

**First Provably Optimal Asynchronous SGD
for Homogeneous and Heterogeneous Data**

Dissertation by

Artavazd Maranjyan

In Partial Fulfillment of the Requirements

For the Degree of

Doctor of Philosophy

King Abdullah University of Science and Technology

Thuwal, Kingdom of Saudi Arabia

©December 4, 2025

Artavazd Maranjyan

All rights reserved

<https://artomaranjyan.github.io>

ABSTRACT

First Provably Optimal Asynchronous SGD for Homogeneous and Heterogeneous Data

Artavazd Maranjyan

Artificial intelligence has achieved remarkable success across domains such as language modeling, vision, and autonomous systems. These breakthroughs are driven by increasingly large neural networks trained on massive datasets using thousands of GPUs or TPUs. Such training runs can occupy entire data centers for weeks to months, consuming vast computational and energy resources.

While advances in hardware and data availability have made this scaling possible, the optimization algorithms for training have not evolved as quickly. Most large-scale training still relies on *synchronous* methods, where all workers must finish their tasks before the next iteration begins. As the number of devices grows, inefficiencies caused by synchronization may grow, too: faster workers remain idle while waiting for slower ones, wasting both compute and energy. In practice, it is nearly impossible for all workers to run at identical speeds—hardware faults and network delays inevitably create *computational heterogeneity*.

At first sight, it may seem that there is an easy fix—removing synchronization—which allows all workers to operate continuously. However, asynchrony introduces *staleness*—some results are produced using outdated versions of the model—making algorithms harder to analyze, especially when delays stem from system-level variability rather than the algorithm itself. Despite extensive research, the *time complexity* of asynchronous methods is poorly understood.

This dissertation addresses this gap. We develop a rigorous framework for asynchronous first-order stochastic optimization, isolating the core challenge these methods target: *heterogeneous worker speeds*. Within this framework, we study stochastic gradient descent (SGD) and show that, with proper design, asynchronous SGD can be *provably optimal* in terms of time complexity, matching optimality results recently achieved by synchronous variants of SGD only.

The first contribution, Ringmaster ASGD, achieves optimal time complexity in the *homogeneous data* setting by selectively discarding stale updates. The second, Ringleader ASGD, extends this result to the *heterogeneous data* regime—typical of federated learning—using a structured gradient-table mechanism that coordinates model updates. Finally, ATA improves resource efficiency by learning workers’ compute-time distributions and allocating tasks adaptively, achieving near-optimal wall-clock time with far less computation.

Together, these results establish asynchronous optimization as a theoretically sound and practically efficient foundation for parallel and distributed learning—showing that coordination without synchronization is not only possible, but that such a strategy enjoys optimal time complexity in theory, while outperforming competing synchronous methods in practice.

ACKNOWLEDGEMENTS

I would like to begin by expressing my deepest gratitude to my advisor, Professor Peter Richtárik. His continuous support, encouragement, and insightful guidance made this dissertation possible. Every discussion with him was a source of motivation and clarity, pushing me forward throughout this challenging journey. From him, I learned how to think about complex ideas from first principles—to seek precision, simplicity, and mathematical elegance in research. He taught me that doing impactful research often begins by asking simple yet foundational questions. His creativity and depth of thought have profoundly influenced the way I approach problems.

I am also deeply grateful to my coauthors—Abdurakhmon Sadiev, Alexander Tyurin, El Mehdi Saad, Francesco Orabona, Laurent Condat, Mher Safaryan, and Omar Shaikh Omar—with whom I have been fortunate to collaborate closely during my PhD. Special thanks to Professor Francesco Orabona—his *Online Learning* course at KAUST remains one of my favorites, and I have greatly enjoyed many insightful discussions with him.

Beyond my direct coauthors, I have been fortunate to work alongside many brilliant colleagues in our research group at KAUST and to share countless technical and non-technical discussions that have shaped my research interests and made my time here truly enjoyable: Ahmad Rammal, Alina Abdikarimova, Alexander Gaponov, Avetik Karagulyan, Egor Shulgin, Elnur Gasanov, Grigory Malinovsky, Hanmin Li, Hussein Rammal, Igor Klimczak, Igor Sokolov, Ivan Ilin, Kai Yi, Kaja Gruntkowska, Konstantin Burlachenko, Majied Ammar Mahran, Samuel Horváth, Sarit Khirirat, Slavomír Hanzely, Yassine Maziane, Yury Demidovich, and Zhirayr Tovmasyan.

I would also like to thank Professor Yi-Shuai Niu for hosting me during my research visit to Beijing, China. That visit gave me the privilege of meeting inspiring professors and exploring several leading universities in Beijing.

I am especially thankful to Hrant Khachatrian and Mher Safaryan, without whose help and guidance I would not have known about KAUST—and perhaps would never have started my PhD journey here. My sincere appreciation also goes to KAUST for providing an exceptional research environment and continuous institutional support.

Last but not least, I am deeply grateful to my family and friends for their constant love and encouragement throughout this journey. Special thanks to my mother, whose endless support and selfless sacrifices made all of this possible.

Contents

Abstract	2
Acknowledgements	3
List of Figures	8
List of Tables	11
1 Introduction	14
1.1 Synchronous Training: Meta-Algorithm	15
1.1.1 Challenges of Synchronization in Practice	15
1.2 Asynchronous Training: Meta-Algorithm	16
1.3 Dissertation Focus	17
1.3.1 Scope: Data Parallelism	18
1.4 Problem Formulation	18
1.4.1 Optimization Problem	18
1.4.2 Assumptions	20
1.4.3 Worker Heterogeneity Model	21
1.5 Worker and Server Dynamics	23
1.5.1 Worker Actions	23
1.5.2 Server Actions	23
1.6 Synchronous vs. Asynchronous Algorithms:	
Formal Definitions	24
1.6.1 Synchronous Algorithms	24
1.6.2 Asynchronous Algorithms	25
1.7 Algorithms for the Homogeneous Setting with Fixed Computation Times	25
1.7.1 Hero SGD	26
1.7.2 Naive Minibatch SGD	27
1.7.3 Naive Asynchronous SGD	28
1.7.4 Rennala SGD	29
1.8 The Beginning of This Work	31
1.9 Contributions and Structure	31

1.9.1	Chapter 2 – Ringmaster ASGD: The First Asynchronous SGD with Optimal Time Complexity	32
1.9.2	Chapter 3 – Ringleader ASGD: The First Asynchronous SGD with Optimal Time Complexity under Data Heterogeneity	34
1.9.3	Chapter 4 – ATA: Adaptive Task Allocation for Efficient Resource Management in Distributed Machine Learning	36
1.10	Publications Not Included in This Dissertation	39
1.11	Common Notations	40
2	Ringmaster ASGD: The First Asynchronous SGD with Optimal Time Complexity	42
2.1	Introduction	42
2.1.1	Assumptions	44
2.1.2	Related Work	44
2.1.3	Contributions	46
2.2	Preliminaries and Naive Method	46
2.2.1	A Naive Optimal Asynchronous SGD	47
2.2.2	Why Is Algorithm 2.2.1 Referred to as “Naive”?	48
2.3	Ringmaster ASGD	49
2.3.1	Description	49
2.3.2	Delay Threshold	50
2.3.3	Why Do We Ignore the Old Stochastic Gradients?	50
2.3.4	Comparison to Rennala SGD	50
2.3.5	Comparison to Previous Asynchronous SGD Variants	51
2.3.6	Stopping the Irrelevant Computations	51
2.4	Theoretical Analysis	52
2.4.1	The Choice of Threshold	54
2.4.2	Proof Techniques	54
2.5	Optimality Under Arbitrary Computation Dynamics	55
2.6	Experiments	57
2.6.1	Neural Network Experiment	58
2.7	Additional Related Work	58
2.8	Conclusion and Future Work	58
3	Ringleader ASGD: The First Asynchronous SGD with Optimal Time Complexity under Data Heterogeneity	60
3.1	Introduction	60
3.1.1	Contributions	63

3.2	Related Work	64
3.3	Problem Setup	65
3.3.1	Worker Heterogeneity Model	65
3.3.2	Notations	66
3.3.3	Assumptions	66
3.4	Background and Motivation	67
3.4.1	Naive Minibatch SGD	67
3.4.2	Malenia SGD	68
3.4.3	Toward Asynchronous Methods	70
3.4.4	IA ² SGD	71
3.4.5	Motivation	72
3.5	Ringleader ASGD	72
3.5.1	Detailed Description	74
3.5.2	Delay Control Analysis	76
3.5.3	Comparison to IA ² SGD	77
3.5.4	Comparison to Malenia SGD	77
3.6	Theoretical Results	78
3.6.1	Iteration Complexity	78
3.6.2	Time Complexity	81
3.7	Experiments	82
3.8	Conclusion	83
4	ATA: Adaptive Task Allocation for Efficient Resource Management in Distributed Machine Learning	85
4.1	Introduction	85
4.1.1	Greedy Task Allocation	85
4.1.2	Wastefulness of GTA	86
4.1.3	Goal of this work	86
4.1.4	A Motivating Example: Optimal Parallel SGD	86
4.1.5	Contributions	87
4.2	Related Work	87
4.3	Problem Setup	88
4.3.1	Task Allocation Protocol	88
4.3.2	Modeling Assumptions	88
4.3.3	Objective of the Allocation Algorithm	89
4.4	Adaptive Task Allocation	90
4.4.1	Reduction to Multi-Armed Bandit and Proxy Loss	90
4.4.2	Comparison with the Combinatorial Bandits Setting	91
4.4.3	Adaptive Task Allocation Algorithm	91

4.4.4	Upper-Bound on the Total Computation Time	93
4.5	Empirical Adaptive Task Allocation	94
4.6	Theoretical Results	94
4.6.1	Guarantees for ATA	95
4.6.2	Guarantees for ATA-Empirical	96
4.7	Experiments	97
5	Concluding Remarks	101
5.1	Summary of Contributions	101
5.1.1	Ringmaster ASGD (Chapter 2)	101
5.1.2	Ringleader ASGD (Chapter 3)	102
5.1.3	Adaptive Task Allocation (Chapter 4)	102
5.2	Future Directions	103
5.2.1	Algorithm-Specific Extensions	103
5.2.2	Broader Research Directions	104
5.3	Closing Remarks	105
References		107
Appendices		122
A Appendix for Chapter 2		123
A.1	Proof of Lemma 2.4.2	123
A.2	Proof of Lemma 2.5.1	124
A.3	Proof of Theorem 2.4.1	125
A.3.1	Proof of Lemma A.3.1	127
A.3.2	Proof of Lemma A.3.2	128
A.4	Proof of Theorem 2.5.2	129
A.5	Derivations for the Example from Section 2.2	131
A.6	When the Initial Point is an ε -Stationary Point	132
B Appendix for Chapter 3		133
B.1	Arbitrarily Changing Computation Times	133
B.1.1	Universal Computation Model	133
B.1.2	Toward an Optimal Method	134
B.1.3	An Optimal Method	134
B.2	Auxiliary Lemmas	136
B.2.1	Proof of Lemma 3.3.4	136
B.2.2	Variance Term	137
B.2.3	Proof of Lemma 3.6.1	138
B.2.4	Proof of Lemma 3.6.2	142

B.3 Time Complexity of IA ² SGD	143
B.4 Improved Malenia SGD	144
C Appendix for Chapter 4	146
C.1 Additional Experiments	146
C.1.1 Linear Noise	146
C.1.2 Heterogeneous Time Distributions	146
C.1.3 Regret	147
C.1.4 Real Dataset	148
C.1.5 Impact of Prior Knowledge on Time Distributions	148
C.2 Concrete Optimization Methods	149
C.2.1 Stochastic Gradient Descent	149
C.2.2 Asynchronous SGD	152
C.3 Recursive Allocation Selection Algorithm	153
C.3.1 Optimality	155
C.3.2 Minimal Cardinality	156
C.4 Proofs of Theorem 4.6.1, Theorem 4.6.3, and Theorem 4.4.2 . . .	157
C.4.1 Proof of Theorem 4.6.1	159
C.4.2 Proof of Theorem 4.6.3	162
C.4.3 Proof of Theorem 4.4.2	166
C.5 Technical Lemmas	166

LIST OF FIGURES

<p>1.7.1 Illustration of Hero SGD (Algorithm 1.7.1) with 3 workers. Only the fastest worker (bottom) performs updates, while the others remain idle. Their idle computation capacity could, in principle, be utilized to compute additional gradients.</p> <p>1.7.2 Illustration of Naive Minibatch SGD (Algorithm 1.7.2) with 3 workers. The shaded regions indicate idle periods when faster devices must wait for the slowest one—wasting compute and doing nothing useful.</p> <p>1.7.3 Illustration of Naive Asynchronous SGD (Algorithm 1.7.3) with 3 workers. Note the absence of shaded regions: all workers remain active, and every computed gradient contributes to a model update.</p> <p>1.7.4 Illustration of Rennala SGD (Algorithm 1.7.4) with 3 workers. The shaded regions indicate idle periods or gradient computations that are eventually discarded during synchronization.</p> <p>2.6.1 Experiment with $n = 6174$ and $d = 1729$ showing the convergence of Ringmaster ASGD, Delay-Adaptive ASGD, and Rennala SGD.</p> <p>2.6.2 We run an experiment on a small 2-layer neural network with ReLU activation on the MNIST dataset, showing that our method, Ringmaster ASGD, is more robust and outperforms Delay-Adaptive ASGD and Rennala SGD.</p> <p>3.7.1 Convergence comparison showing median squared gradient norm $\ \nabla f(x^k)\ ^2$ (solid lines) with interquartile ranges (shaded regions) plotted against wall-clock time, averaged over 30 random seeds. Setup: A two-layer MLP with architecture $\text{Linear}(d, 128) \rightarrow \text{ReLU} \rightarrow \text{Linear}(128, 10)$ is trained on (a) MNIST and (b) Fashion-MNIST datasets. Client delays: Heterogeneous delays are simulated as $\tau_i = i + \eta_i$, where $\eta_i \sim \mathcal{N}(0, i)$ for each client $i \in [n]$, with $n = 100$. Results: With optimally tuned stepsizes, Ringleader ASGD converges faster than both Malenia SGD and IA²SGD, even though Ringleader ASGD and Malenia SGD share the same theoretical time-complexity guarantees.</p>	<p>26</p> <p>27</p> <p>28</p> <p>30</p> <p>58</p> <p>59</p> <p>84</p>
--	---

4.7.1 Each row increases the number of workers by a factor of 3, starting from 17, that is, $n = 17, 51, 153, 459$ from top to bottom. The first column shows runtime vs. suboptimality. The second column also plots suboptimality, but against total worker time, i.e., $\sum_{i=1}^n T_i^k$ in Algorithm 4.4.1. The third column presents the average iteration time, given by C_k/k over all iterations k . The last column displays the averaged cumulative regret, as defined in (4.9).	98
C.1.1 Each row increases the number of workers by a factor of 3, starting from 17, that is, $n = 17, 51, 153, 459$ from top to bottom. The first column shows runtime vs. suboptimality. The second column also plots suboptimality, but against total worker time, i.e., $\sum_{i=1}^n T_i^k$ in Algorithm 4.4.1. The third column presents the average iteration time, given by C_k/k over all iterations k . The last column displays the averaged cumulative regret, as defined in (4.9).	147
C.1.2 Each row corresponds to an increasing number of workers, with $n = 15, 45, 150$ from top to bottom. We consider five distributions—Exponential, Uniform, Half-Normal, Lognormal, and Gamma—grouping them to have the same mean and then varying the mean across different groups. The results demonstrate that the algorithms remain robust across different distributions. The columns represent the same as in Figure C.1.1.	149
C.1.3 Regret growth over iterations.	150
C.1.4 We use the CIFAR-100 dataset (Krizhevsky et al., 2009). The model is a CNN with three convolutional layers and two fully connected layers, totaling 160k parameters. The Adam optimizer (Kingma & Ba, 2014) is used with a constant step size of $8 \cdot 10^{-5}$. The computation time of the workers follows the same setup as in Figure C.1.1, where the mean time increases linearly. The batch size remains the same at $B = 23$. Each row corresponds to a different number of workers, with $n = 17, 51, 153$ from top to bottom.	151
C.1.5 We use the same optimization setup as in Figure C.1.4, with $B = 23$ and $n = 51$. The number of prior iterations, P , varies across rows, starting from the top with $P = 0, 5 \cdot 10^5, 5 \cdot 10^6$. As the number of prior iterations increases, we observe that the training of ATA and ATA-Empirical accelerates, bringing their performance closer to the optimal performance of OFTA.	152

LIST OF TABLES

<p>1.1 Each chapter of this dissertation introduces a main algorithmic contribution, listed in the second column together with its reference. The third column indicates whether the method achieves optimal time complexity, i.e., whether it attains the theoretical lower bound on convergence time. The fourth column specifies the data regime in which each method operates—homogeneous or heterogeneous. The fifth column describes the worker computation time model used in the analysis, while the sixth column indicates whether the method is “resource efficient”. The first two methods focus solely on minimizing wall-clock time and therefore utilize all available computational resources, whereas the third method, ATA, improves computational efficiency while losing only constant factors in the wall-clock time.</p> <p>1.2 Publications not included in this dissertation.</p> <p>2.1 The time complexities of asynchronous stochastic gradient methods, which perform the step $x^{k+1} = x^k - \gamma^k \nabla f(x^{k-\delta^k}; \xi_{i^k}^{k-\delta^k})$, to get an ε-stationary point in the nonconvex setting. In this table, we consider the <i>fixed computation model</i> from Section 2.2. Abbr.: σ^2 is defined as $\mathbb{E}_\xi[\ \nabla f(x; \xi) - \nabla f(x)\ ^2] \leq \sigma^2$ for all $x \in \mathbb{R}^d$, L is the smoothness constant of f, $\Delta := f(x^0) - f^*$, $\tau_i \in [0, \infty]$ is the time bound to compute a single stochastic gradient by worker i^k.</p>	<p>32</p> <p>39</p> <p>43</p>
---	-------------------------------

3.1 Comparison of time complexities for parallel first-order methods under the fixed computation time model, where each worker i takes a fixed time τ_i to compute a stochastic gradient, with the times ordered so that τ_n is the largest (3.2). We denote by $\tau_{\text{avg}} := \frac{1}{n} \sum_{i=1}^n \tau_i$ the average computation time across all workers. The table shows how the time complexity of different algorithms depends on key problem parameters: the initial function suboptimality $\Delta := f(x^0) - f^*$ (Assumption 3.3.5), the target stationarity ε , the variance bound of the stochastic gradients σ^2 (Assumption 3.3.1), and smoothness constants. Specifically, L_f is the smoothness constant of f (Definition 3.3.3); $L_{\max} := \max_{i \in [n]} L_{f_i}$ with L_{f_i} the smoothness constant of f_i ; and L is a constant associated with our new smoothness-type assumption (Assumption 3.3.2). They satisfy $L_f \leq L \leq L_{\max}$ (Lemma 3.3.4). All stated time complexities hide universal constant factors. Each column indicates whether a method satisfies the following desirable properties: **Optimal**: achieves the theoretical lower bound derived by Tyurin & Richtárik (2024) for parallel first-order stochastic methods in heterogeneous data setting. **No sync.**: does not require synchronization and is therefore *asynchronous*. **No idle workers**: all workers remain busy without waiting, so computational resources are fully utilized. **No discarded work**: no computation is wasted, and no worker is stopped mid-computation. Our new method, Ringleader ASGD, is the first asynchronous method to achieve optimal time complexity, while also ensuring full resource utilization (no idle workers) and no discarded computations / work.

62

4.1 Ratios of total worker times and runtimes required to achieve $f(x) - f^* < 10^{-5}$. For total worker time, we divide the total worker time of **GTA** by the corresponding total worker times of the other algorithms listed. For runtime, we do the opposite, dividing the runtime of the other algorithms by the runtime of **GTA**, since **GTA** is the fastest. To simplify the naming, we refer to **ATA-Empirical** as **ATA-E**.

99

C.1 Ratios of total worker times and runtimes required to achieve $f(x) - f^* < 10^{-5}$. For total worker time, we divide the total worker time of GTA by the corresponding total worker times of the other algorithms listed. For runtime, we do the opposite, dividing the runtime of the other algorithms by the runtime of GTA , since GTA is the fastest.	148
---	-----

Chapter 1

Introduction

Artificial Intelligence (AI), once primarily a research topic, now shapes applications that billions use daily. The most visible examples today are conversational AI systems such as ChatGPT and other chatbots (Brown et al., 2020; Achiam et al., 2023; Team et al., 2023). Beyond chat, AI also powers recommendation systems (Covington et al., 2016; Zhang et al., 2019), autonomous driving (Bojarski et al., 2016; Chib & Singh, 2023), image recognition (Krizhevsky et al., 2012; Maurício et al., 2023), and medical diagnostics (Esteva et al., 2017; Rajpoot et al., 2024). What unites these applications is the reliance on large-scale deep learning models trained on vast amounts of data (Kaplan et al., 2020).

These models are so computationally demanding that they cannot be trained or even deployed efficiently on a single machine (Dean et al., 2012; Li et al., 2020a). They require distributed training on a cluster of specialized hardware, often involving thousands of interconnected GPUs or TPUs (Jouppi et al., 2017; Narayanan et al., 2021). Large-scale data centers—massive facilities that interconnect this hardware—have therefore become essential infrastructure, built by companies such as Google, Meta, and Amazon. State-of-the-art AI training runs today often consume weeks or months of computation on these data centers (Narayanan et al., 2021; Grattafiori et al., 2024).

As both models and datasets continue to grow, so do the computational and energy demands of AI training. Data centers are already among the most energy-intensive building types, and their demand is increasing faster than that of most other sectors, posing a major challenge for sustainable energy systems (Strubell et al., 2020; Agency, 2025). Recent estimates suggest that by 2030, data center energy consumption could reach levels comparable to Japan’s current annual electricity usage (Agency, 2025).

Given the growing computational and energy demands of AI training, improving the efficiency of training large models has become a critical and highly active research area (Shoeybi et al., 2019; Rajbhandari et al., 2020; Narayanan et al., 2021; Hu et al., 2022; Wan et al., 2024). Despite rapid advances in hardware and software, current large-scale training pipelines still have substantial room for improvement. For example, the training of LLaMA-3 achieved only about 38-43% Model FLOPs Utilization (a standard metric for measuring training efficiency (Chowdhery et al., 2023)) (Grattafiori et al., 2024). This means that for much of the time, expensive GPUs were idle, yet still consuming power, rather than performing useful computation. This underutilization is like reserving an entire fleet of airplanes for a week, but flying only a few of them each day while the rest sit idling on the runway with engines running. The cost of wasted resources in such a scenario is enormous—and in the case of large AI training runs, this inefficiency translates into billions of dollars in infrastructure expenses and energy consumption (De Vries, 2023; Cottier et al., 2024; O’Donnell & Crownhart, 2025;

Elsworth et al., 2025).

The underutilization of resources in large-scale AI training comes from several sources, including the limits of current hardware and the communication overhead of transferring data between machines. A particularly important source of inefficiency, however, lies on the algorithmic side. Most existing training algorithms were not originally designed with today’s massive computing clusters in mind, and they fail to make *optimal* use of such large-scale systems. In particular, they rely on synchronization across machines, an approach that becomes increasingly costly as the system grows.

1.1 Synchronous Training: Meta-Algorithm

Historically, large-scale AI training has relied on synchronous iterative algorithms, where all workers must complete their assigned tasks before the system can advance to the next iteration. This strategy has remained the dominant approach for training large AI models (Goyal et al., 2017; Li et al., 2020a; Douillard et al., 2024).

To illustrate the principle behind these methods, we abstract away the details of the underlying optimization problem and present a general *meta-algorithm* (Algorithm 1.1.1) that captures the structure common to many synchronous approaches. A precise definition of “synchronous algorithm” is deferred to Section 1.6; here, the meta-algorithm is intentionally informal and serves to highlight common behavior and its bottlenecks.

Algorithm 1.1.1 Synchronous Meta-Algorithm (Server Perspective)

```

1: for iteration  $k = 0, 1, 2, \dots$  do
2:   Broadcast the current model  $x^k$  to all workers
3:   Instruct all workers to compute their local results using  $x^k$ 
4:   Wait until all workers have finished, and their results have been received
5:   Aggregate the collected results
6:   Update the model to obtain  $x^{k+1}$ 
7: end for

```

The defining feature of these methods is the *synchronization barrier*: the server must *wait* for all workers to finish (Line 4) before it can aggregate their results (Line 5), compute the new model (Line 6), and broadcast it again to the workers (Line 2). We highlight Line 4 because it is the bottleneck: faster workers cannot proceed with new computations and remain idle until the slowest worker finishes. This idleness is the central drawback of synchronous methods and a major source of underutilization in large-scale data centers.

1.1.1 Challenges of Synchronization in Practice

One might hope to solve the synchronization problem by building data centers with identical hardware, ensuring all workers run at the same speed. In practice, however, this strategy faces several fundamental obstacles.

Stragglers Persist Even with Identical Hardware

Even when all machines in a cluster are identical, *stragglers*—workers that are slow due to compute or communication delays—still arise because of unpredictable issues such as hardware faults (e.g., overheating) and network problems (Dean & Barroso, 2013; Ananthanarayanan et al., 2013; Maranjyan et al., 2025a). A single malfunctioning component can cause one worker to lag significantly behind others, forcing the entire cluster to wait. In other words, identical hardware specifications do not guarantee identical performance in practice.

Hardware Upgrades Become Prohibitively Expensive

Enforcing hardware homogeneity creates a second problem: it makes upgrading the data center extremely difficult. When newer, more powerful processors become available, adding them to an existing cluster introduces heterogeneity. Since faster machines must still wait for slower ones in synchronous training, much of the performance benefit of the upgrade is lost. This forces operators into an expensive dilemma: either replace all machines simultaneously (which is extremely costly) or delay upgrades until a complete replacement is feasible. Such constraints make it difficult to gradually expand capacity or adopt new technology as it becomes available.

Federated and Edge Settings

Finally, there are important scenarios where ensuring identical hardware is simply impossible. In federated learning, training occurs across everyday devices such as smartphones, sensors, or other personal electronics (Konečný et al., 2016c,a; McMahan et al., 2017). These devices vary dramatically in their computational power, battery life, internet connectivity, and availability. Participants may join or leave the training process unpredictably, and connection quality can fluctuate significantly. In such environments, waiting for the slowest device to complete each round is not merely inefficient—it may be impossible, as some devices might disconnect or fail to complete their work within a reasonable timeframe.

1.2 Asynchronous Training: Meta-Algorithm

Given the fundamental limitations of synchronization discussed above, researchers have proposed an algorithmic solution: simply remove the synchronization requirement altogether (Tsitsiklis et al., 1986; Recht et al., 2011; Agarwal & Duchi, 2011). If the synchronization barrier is the source of inefficiency, then eliminating it should allow workers to operate continuously without idle time.

To illustrate this approach, we again present a general meta-algorithm (Algorithm 1.2.1) that captures the structure common to many asynchronous methods. As before, we defer a precise definition of “asynchronous algorithm” to Section 1.6.

Algorithm 1.2.1 Asynchronous Meta-Algorithm (Server Perspective)

- 1: Initialize model and send to all workers
 - 2: **while** not terminated **do**
 - 3: Receive result from some worker i
 - 4: **No waiting – proceed immediately**
 - 5: Optionally update the model (based on some criterion)
 - 6: Send current model to worker i
 - 7: **Worker i starts computing again**
 - 8: **end while**
-

The key distinction between synchronous and asynchronous methods appears in Line 4, where the server proceeds as soon as it receives a result from any worker, without waiting for the others. Similarly, in Line 7, each worker continues computing immediately—without waiting for other workers. This design eliminates the idle time that plagues synchronous approaches, allowing all workers to operate independently and continuously using their current model versions.

However, this independence comes at a cost. When a worker begins its computation and later sends back its result, the model may have already been updated multiple times by other workers in the meantime. This means the worker’s result is based on an outdated, or *stale*, version of the model. Such staleness can degrade optimization progress, potentially slowing convergence or leading to suboptimal solutions.

Moreover, Line 5 reveals that asynchronous methods offer additional flexibility—the server need not update the model after every received result. These observations raise fundamental design questions: how should one design an asynchronous method to maximize its computational benefits while mitigating the effects of staleness?

1.3 Dissertation Focus

While asynchronous methods eliminate the key inefficiency of synchronous methods—keeping workers idle while waiting for slow workers—they introduce a new challenge: staleness in model updates. This creates a fundamental paradox. Staleness is not merely a side effect of asynchrony—it is the very mechanism that enables continuous worker utilization. Workers can continue computing precisely because they do not wait for the model to be fully up-to-date. Yet this same staleness means that updates are computed using outdated information, which can interfere with optimization progress and potentially harm convergence. If workers operate on severely outdated models, their updates may no longer be beneficial and could actively degrade the training process.

Predicting and managing staleness is challenging on multiple fronts. It is difficult to anticipate how much staleness a given system will experience, how that staleness will affect convergence, and which design choices lead to algorithms that work well in practice while maintaining theoretical guarantees. Moreover, staleness makes the analysis considerably more difficult: the classical convergence analysis of synchronous methods no longer applies, and new theoretical machinery must be developed to understand asynchronous behavior.

The central goal of this dissertation is to develop a rigorous understanding of asynchronous optimization and to design provably efficient asynchronous algorithms for large-scale machine learning.

1.3.1 Scope: Data Parallelism

Modern distributed training systems employ multiple forms of parallelism to handle large models and datasets. These include *data parallelism*, where workers process different subsets of data; *model parallelism*, where different parts of the model are distributed across workers; and *pipeline parallelism*, where different layers of the model are assigned to different stages of a pipeline (Narayanan et al., 2021; Grattafiori et al., 2024). In practice, state-of-the-art training systems often combine several or even all of these strategies simultaneously.

This dissertation focuses exclusively on *data parallelism*, the most fundamental and widely applicable form of parallelism. In data parallelism, all workers maintain a copy of the complete model but process different batches of training data. We restrict our attention to this setting for two reasons. First, data parallelism is the simplest form of parallelism and serves as a natural starting point for developing theoretical understanding. Second, despite its apparent simplicity, even the asynchronous data-parallel setting remains poorly understood—fundamental questions about convergence, optimal algorithm design, and the effects of staleness remain open.

1.4 Problem Formulation

Having outlined the general focus of this dissertation, we now provide a concrete formulation of the problem we study. We begin by presenting the optimization problem that lies at the core of distributed machine learning. Then, since our primary interest is in the convergence time of algorithms, we formalize which operations take time and how these quantities are modeled in our analysis.

1.4.1 Optimization Problem

We consider a distributed learning setup with n clients, where each client i has access to its own local data distribution \mathcal{D}_i . The goal is to learn a global model, parameterized by a vector $x \in \mathbb{R}^d$, where d denotes the dimensionality of the model parameters, by solving the optimization problem

$$\underset{x \in \mathbb{R}^d}{\text{minimize}} \left\{ f(x) := \frac{1}{n} \sum_{i=1}^n f_i(x) \right\}, \quad (1.1)$$

where $f_i: \mathbb{R}^d \rightarrow \mathbb{R}$ denotes the local objective function of client i , and the global objective function f is defined as their average.

The local objective functions can generally be written as

$$f_i(x) := \mathbb{E}_{\xi_i \sim \mathcal{D}_i} [f_i(x; \xi_i)], \quad (1.2)$$

where $f_i(x; \xi_i)$ is the loss function of the model parameterized by x evaluated on a data sample ξ_i . Thus, the local objective is the expectation of this loss over the

entire local data distribution \mathcal{D}_i . This formulation corresponds to minimizing the population risk.

When the local data distribution is finite, i.e., $|\mathcal{D}_i| < \infty$, the local objective (1.2) reduces to the average loss over the local training data points,

$$f_i(x) = \frac{1}{|\mathcal{D}_i|} \sum_{j=1}^{|\mathcal{D}_i|} f_i(x; \xi_{i,j}) .$$

Homogeneous vs. Heterogeneous Data

In general, the local objective functions f_i can differ significantly across clients depending on their local data distributions \mathcal{D}_i . This heterogeneity poses a fundamental challenge for solving the distributed optimization problem and is a key characteristic that distinguishes different practical settings.

Heterogeneous setting. In federated learning scenarios, data heterogeneity is unavoidable. For example, when clients are personal devices such as smartphones, each device contains user-specific data that reflects the habits, preferences, and environments of a particular individual. Such data cannot be redistributed or centralized without violating privacy constraints. As a result, the local data distributions \mathcal{D}_i may differ drastically across clients—ranging from text typed in different languages to photos taken under different conditions—making the optimization problem inherently more challenging. We refer to this as the *heterogeneous data setting*.

Homogeneous setting. In contrast, data center environments offer significantly more freedom in organizing and distributing data. In such settings, data can be freely shuffled, replicated, or centralized without privacy concerns. It is therefore common practice to ensure that all workers have access to data drawn from the same underlying distribution, either by sharing a common dataset or by partitioning the data so that the resulting subsets are approximately statistically identical across workers. Formally, this corresponds to assuming $\mathcal{D}_i = \mathcal{D}$ for all clients i .

In addition, we typically employ the same loss function across all clients, i.e., $f_i(x; \xi_i) \equiv f(x; \xi)$. Under these assumptions, the optimization problem (1.1) simplifies to

$$\underset{x \in \mathbb{R}^d}{\text{minimize}} \{f(x) := \mathbb{E}_{\xi \sim \mathcal{D}} [f(x; \xi)]\} . \quad (1.3)$$

We refer to this as the *homogeneous data setting*.

Both settings are of practical importance and present distinct algorithmic challenges. The homogeneous setting, while more structured, still requires the design of efficient distributed algorithms. The heterogeneous setting, being more general, demands methods that are robust to statistical diversity across clients. Importantly, these settings need to be studied separately: analyzing only the general heterogeneous case does not necessarily yield optimal solutions for the homogeneous case. In the homogeneous setting, one can exploit the fact that all clients share the same data distribution, which enables sharper guarantees and

often simpler algorithms. In this dissertation, we develop and analyze algorithms tailored to both settings.

1.4.2 Assumptions

To make the problem setup concrete, we introduce several standard assumptions on the class of functions we study.

Assumption 1.4.1 (Smoothness). The function f is differentiable, and its gradient is Lipschitz continuous. That is, there exists a constant $L > 0$ such that, for all $x, y \in \mathbb{R}^d$,

$$\|\nabla f(x) - \nabla f(y)\| \leq L \|x - y\|.$$

This assumption, often referred to as *L-smoothness*, controls how quickly the gradient can change.

Assumption 1.4.2 (Lower boundedness). The objective function is bounded below: there exists $f^* > -\infty$ such that $f(x) \geq f^*$ for all $x \in \mathbb{R}^d$.

This assumption ensures that the optimization problem is well-posed: without it, function values could decrease without bound, making convergence guarantees meaningless.

We define $\Delta := f(x^0) - f^*$, where x^0 is the initial point of the optimization algorithm. The quantity Δ represents the initial suboptimality and serves as a measure of the inherent difficulty of the problem.

Convergence Criterion

Under these assumptions, the standard goal in nonconvex optimization is to find an ε -stationary point, that is, a (possibly random) vector x satisfying

$$\mathbb{E} [\|\nabla f(x)\|^2] \leq \varepsilon. \quad (1.4)$$

This criterion is natural in the nonconvex setting: while finding global minima is generally intractable, first-order stationary points are attainable and often correspond to practically useful solutions.

Stochastic First-Order Methods

We focus on stochastic first-order methods, which rely solely on gradient information. The earliest and most fundamental example is stochastic gradient descent (SGD), introduced by [Robbins & Monro \(1951\)](#) and later popularized in machine learning by [Bottou et al. \(2018\)](#). Building on this foundation, adaptive methods such as Adam ([Kingma & Ba, 2014](#)) and its variant AdamW ([Loshchilov & Hutter, 2019](#)) have become standard in training large-scale deep learning models nowadays. In our analysis, we begin with SGD, both for its simplicity and because it serves as the theoretical basis for many of these modern variants. To study its behavior in the distributed setting, we impose additional assumptions on the stochastic gradients computed by the workers.

Assumption 1.4.3. For each $i \in \{1, \dots, n\}$ and every ξ_i , the function $f_i(x; \xi_i)$ is differentiable with respect to its first argument x . Moreover, the stochastic gradients are unbiased and have bounded variance $\sigma^2 \geq 0$, that is,

$$\begin{aligned}\mathbb{E}_{\xi_i \sim \mathcal{D}_i} [\nabla f_i(x; \xi_i)] &= \nabla f_i(x), \quad \forall x \in \mathbb{R}^d, \quad \forall i \in \{1, \dots, n\}, \\ \mathbb{E}_{\xi_i \sim \mathcal{D}_i} [\|\nabla f_i(x; \xi_i) - \nabla f_i(x)\|^2] &\leq \sigma^2, \quad \forall x \in \mathbb{R}^d, \quad \forall i \in \{1, \dots, n\}.\end{aligned}$$

The first condition ensures that stochastic gradients are unbiased estimators of the true gradient, while the second condition bounds their variance. The parameter σ quantifies the level of stochastic noise: when $\sigma = 0$, the gradients are exact.

1.4.3 Worker Heterogeneity Model

We are interested in the *time complexity* of solving problem (1.1), that is, the wall-clock time required to reach a desired level of stationarity (1.4). To analyze time complexity in distributed systems, we model the computational characteristics of individual workers. In this section, we formalize the notion of worker computational heterogeneity through their computation times.

Fixed Computation Times

We begin with the case where workers operate at constant computation speeds. This setting provides an intuitive foundation for understanding the dynamics of asynchronous distributed optimization and isolates the effects of heterogeneity from other sources of variability.

Following the *fixed computation model* of Mishchenko et al. (2022a), we assume:

Each worker i requires τ_i seconds¹ to compute one stochastic gradient $\nabla f_i(x; \xi_i)$. (1.5)

Without loss of generality, we assume

$$0 < \tau_1 \leq \tau_2 \leq \dots \leq \tau_n.$$

Differences in computation times $\{\tau_i\}_{i=1}^n$ may arise from diverse sources: heterogeneous hardware (different GPU generations), variation in local dataset sizes, or competition for shared computational resources. This model captures the central challenge of asynchronous optimization: designing algorithms that efficiently exploit updates from workers that progress at different speeds.

Time-Varying Computation Times

In practice, computation times are rarely constant. Even in data centers, workers may slow down due to shared resource usage, background processes, overheating, or occasional hardware failures. This variability is even more pronounced in federated learning, where clients can face unstable network connections, limited battery life, or interruptions from user activity.

To capture these effects, we also consider models with time-varying computation times (Tyurin, 2024). A formal treatment of this more general setting

¹Throughout this dissertation, we use “seconds” as a generic unit of time. The analysis applies to any consistent time unit.

is deferred to later chapters (see, for example, Section B.1.1). Crucially, we assume that variations in computation times are *non-adversarial*: they may depend on system conditions or external factors, but not on the optimization algorithm itself. This assumption reflects realistic scenarios where delays are caused by environment and hardware limitations rather than by deliberate interference.

Stochastic Computation Times

While the time-varying model introduced above can capture virtually any computation behavior, it provides no meaningful theoretical guarantees—its performance depends entirely on how the computation times evolve over time. To obtain more interpretable results, we also consider a stochastic setting, where each worker’s computation time is modeled as a random variable drawn from an underlying probability distribution. This formulation allows us to derive guarantees in expectation, explicitly characterizing how the distributional parameters (e.g., mean, variance) affect the algorithm’s behavior. We adopt this stochastic assumption in parts of this dissertation; the precise formulation is given in Section 4.3.2.

Zero Communication and Auxiliary Costs

For clarity of exposition, we assume that the only operation that consumes time is the computation of stochastic gradients. All other operations—including worker-to-server and server-to-worker communication, server-side model updates, and memory reads or writes—are assumed to take zero seconds. Of course, this assumption is not realistic, as communication delays and server overhead are non-negligible in real systems.

Memory Locking and Inconsistent Reads

A line of work initiated by Recht et al. (2011) studies *lock-free* updates on a shared parameter vector, where multiple workers read and write at the same time. In that setting, while a worker is reading the server’s parameter vector, other workers can change yet-unread coordinates; the result can be a local copy of the model that mixes values taken before and after those updates.

We do not consider such effects in our analysis, since we assume that communication, reads, and writes are instantaneous (zero-time) and *atomic*. Consequently, each read returns a single, well-defined model vector and each write completes indivisibly. Under these assumptions, such inconsistencies cannot arise, and locking is unnecessary.

Rationale Behind These Simplifications

The assumptions introduced above intentionally eliminate all sources of time other than the workers’ computation times. We adopt this simplified setting because the primary challenge that asynchronous methods aim to address is *heterogeneity in worker speeds*. Even under such idealized conditions, the behavior of asynchronous algorithms is not yet fully understood: in many cases, they provide no advantage over their synchronous counterparts. Establishing a rigorous

understanding in this controlled environment is therefore a necessary first step before incorporating additional system-level complexities.

1.5 Worker and Server Dynamics

Before describing specific algorithms, let us outline the basic set of actions available to the workers and to the server. All algorithms we construct later will be built from these basic operations.

1.5.1 Worker Actions

Each worker can perform only two types of actions at any given time:

- **Idle:** The worker simply does nothing, waiting for instructions from the server.
- **Compute and send gradient:** The worker samples a new iid data point, computes a stochastic gradient at the model it was assigned, and immediately sends this gradient to the server. These stochastic gradients are unbiased, and their variance is bounded as assumed in Assumption 1.4.3.

These are the only worker actions we consider. One could imagine allowing workers to store gradients locally before sending them, but in our setting this makes no difference, since we assume communication is instantaneous.

A related approach is to perform local model updates, as commonly done in federated learning (McMahan et al., 2017). While this dissertation does not consider local steps, such extensions are possible (Tyurin & Sivtsov, 2025; Fradin et al., 2025).

1.5.2 Server Actions

The server has a richer set of possible actions, since it must coordinate the overall training process. When a gradient arrives from a worker, the server can:

- **Store the gradient:** Keep it in memory for potential use in a future update.
- **Discard the gradient:** For example, if it decides the gradient is outdated or of poor quality.

In addition to reacting to incoming gradients, the server can independently perform the following actions at any time:

- **Update the model:** Update the model using stored gradients. This includes both immediate updates upon gradient arrival and delayed updates using multiple stored gradients.
- **Discard stored gradients:** Remove some of the previously stored gradients, for example if they have already been used sufficiently or are no longer needed.

- **Request gradient:** Send the current model to one or more workers, instructing them to compute gradients. In practice, the server may sometimes ask a worker to recompute a gradient at the same model it was already using, in which case resending the model is unnecessary. However, since we assume communication is instantaneous, we simply model requests as always sending the model.
- **Terminate one or more workers’ computations:** If the server anticipates that the results of certain ongoing computations will be discarded, it may cancel them early to avoid wasted effort. The affected workers then simply return to the idle state until they receive new instructions.

Although these actions could, in principle, occur at any time, in practice an algorithm will specify precisely when they take place (e.g., upon receiving a gradient, after an update, etc.). Thus, the server’s behavior is structured rather than arbitrary.

The methods that operate within this framework will be referred to as *distributed* or *parallel* methods. Both synchronous and asynchronous algorithms can be expressed in terms of these actions. In the next section, we formalize how the distinction between synchronous and asynchronous methods arises within this framework.

1.6 Synchronous vs. Asynchronous Algorithms: Formal Definitions

We now formally define what we mean by *synchronous* and *asynchronous* algorithms.

1.6.1 Synchronous Algorithms

We call an algorithm *synchronous* if, whenever the server performs a model update from stored stochastic gradients, it uses only gradients computed at the *current* server model. Equivalently, all gradients in an update correspond to the same model state. Consequently, the update mirrors a standard SGD step with an unbiased gradient estimator.

Formally, a synchronous update can be written as

$$x^{k+1} = x^k - \gamma^k \sum_{i=1}^n w_i^k \sum_{j=1}^{b_i^k} \nabla f_i(x^k; \xi_i^{k,j}), \quad (1.6)$$

where $\gamma^k > 0$ is a stepsize, $w_i^k \geq 0$ are aggregation weights, and $b_i^k \in \{0, 1, 2, \dots\}$ is the number of gradients contributed by worker i in round k (by convention, if $b_i^k = 0$, then we take the sum as 0). All synchronous algorithms considered in this dissertation fit this template via different choices of (γ^k, w_i^k, b_i^k) —including Hero SGD, Naive Minibatch SGD, and Rennala SGD (see Section 1.7).

Because updates must use only gradients computed at the *same* model state, synchronous methods have inherent limitations. When worker speeds differ, they either (i) force faster workers to wait until the slowest finishes, or (ii) discard late computations that miss the synchronization point. Of course, this is not an issue

if worker speeds are roughly the same, but this is hard to guarantee, as already discussed in Section 1.1.1.

1.6.2 Asynchronous Algorithms

By contrast, we call a method *asynchronous* if its updates may use gradients evaluated at *stale* model copies—that is, earlier versions of the model than the one currently held by the server. While this need not occur in every iteration, asynchronous methods must generally account for such delays.

For simplicity, we assume that in each round, all gradients contributed by a given worker are computed using the *same* model iterate. This assumption covers all asynchronous methods analyzed in this dissertation, and relaxing it to allow mixtures of stale points per worker per round would be straightforward but unnecessary for our purposes.

Let $\delta_i^k \in \{0, 1, 2, \dots\}$ denote the *delay* at iteration k : the number of server updates that have occurred since worker i received its model iterate. With this notation, an asynchronous update takes the form

$$x^{k+1} = x^k - \gamma^k \sum_{i=1}^n w_i^k \sum_{j=1}^{b_i^k} \nabla f_i \left(x^{k-\delta_i^k}; \xi_i^{k-\delta_i^k, j} \right),$$

where $\gamma^k > 0$ is the stepsize, $w_i^k \geq 0$ are aggregation weights, and $b_i^k \in \{0, 1, 2, \dots\}$ is the number of gradients contributed by worker i in round k .

Note that when all $\delta_i^k = 0$, we recover the synchronous update (1.6); some algorithms occasionally realize such synchronous steps. However, the goal is to provide guarantees for any admissible delay pattern. An illustrative instance of an asynchronous method is **Naive Asynchronous SGD**, presented in Section 1.7.

1.7 Algorithms for the Homogeneous Setting with Fixed Computation Times

Now that we have formalized the problem setting, we can begin exploring algorithmic solutions. Recall that we have n machines, each capable of computing unbiased stochastic gradients with bounded variance. We assume that each machine’s computation time is fixed, as specified by the fixed computation model in (1.5), and that the server knows these times.

Our goal is to design algorithms that efficiently solve the optimization problem in this distributed setting. We begin by focusing on the *homogeneous data setting*, where we aim to solve problem (1.3).

If we were not concerned with time complexity, we could simply apply standard SGD and achieve optimal iteration complexity (Ghadimi & Lan, 2013; Arjevani et al., 2022). Consequently, in the single-machine setting, plain SGD is optimal. This observation motivates a natural starting point: begin with SGD and investigate whether it can be adapted to achieve optimal time complexity in the distributed setting.

1.7.1 Hero SGD

The simplest approach to applying SGD in our setting is to ignore the fact that we have n machines available and use only a single one. In that case, it is natural to select the fastest machine—the one with computation time τ_1 —and run standard SGD on it. We refer to this baseline as **Hero SGD**,² and the corresponding algorithm is given in Algorithm 1.7.1. We also illustrate its behavior using a simple example with three workers in Figure 1.7.1.

Algorithm 1.7.1 Hero SGD

- 1: **Input:** initial point $x^0 \in \mathbb{R}^d$, stepsizes $\gamma^k > 0$
 - 2: **for** $k = 0, \dots, K - 1$ **do**
 - 3: Request the fastest worker to compute a stochastic gradient $\nabla f(x^k; \xi^k)$
 - 4: Wait until the gradient arrives
 - 5: Update the model:

$$x^{k+1} = x^k - \gamma^k \nabla f(x^k; \xi^k)$$
 - 6: **end for**
-

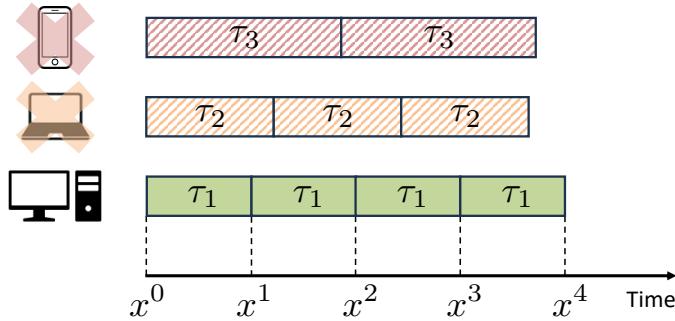


Figure 1.7.1: Illustration of Hero SGD (Algorithm 1.7.1) with 3 workers. Only the fastest worker (bottom) performs updates, while the others remain idle. Their idle computation capacity could, in principle, be utilized to compute additional gradients.

This algorithm requires knowing which worker is the fastest. If this information is unavailable, one could adopt a *greedy* strategy: request all workers to compute a gradient and use the first result that arrives, discarding the rest. Alternatively, one may attempt to learn the computation-time distributions of the workers and allocate tasks accordingly; we discuss such adaptive strategies later in Section 1.9.3. In this section, however, we assume that computation times are fixed and known, so such considerations are unnecessary.

Since each iteration of this method takes τ_1 seconds, and the iteration complexity of SGD is known to be (Ghadimi & Lan, 2013)

$$K = \mathcal{O}\left(\frac{L\Delta}{\varepsilon} + \frac{\sigma^2 L\Delta}{\varepsilon^2}\right),$$

²The name is taken from the work by Fradin et al. (2025).

the total time complexity becomes

$$T = \tau_1 K .$$

While simple, this approach completely fails to exploit the computational power of the remaining $n - 1$ machines, which remain idle throughout the training process.

1.7.2 Naive Minibatch SGD

A natural next step is to utilize all n machines. The most straightforward way to do this is to have each machine compute one stochastic gradient per iteration and then update the model using their average. This gives the Naive Minibatch SGD algorithm, formalized in Algorithm 1.7.2. Similar to the previous case, we illustrate the behavior of Naive Minibatch SGD using a three-worker example in Figure 1.7.2.

Algorithm 1.7.2 Naive Minibatch SGD

- 1: **Input:** initial point $x^0 \in \mathbb{R}^d$, stepsizes $\gamma^k > 0$
- 2: **for** $k = 0, \dots, K - 1$ **do**
- 3: Broadcast the current model x^k to all workers
- 4: Each worker i computes a stochastic gradient $\nabla f(x^k; \xi_i^k)$
- 5: Wait until all n gradients have been received
- 6: Aggregate the gradients and update the model:

$$x^{k+1} = x^k - \gamma^k \frac{1}{n} \sum_{i=1}^n \nabla f(x^k; \xi_i^k)$$

- 7: **end for**
-

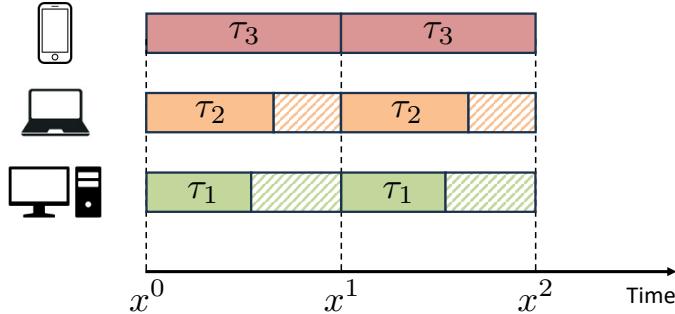


Figure 1.7.2: Illustration of Naive Minibatch SGD (Algorithm 1.7.2) with 3 workers. The shaded regions indicate idle periods when faster devices must wait for the slowest one—wasting compute and doing nothing useful.

The iteration complexity of this method is similar to that of Hero SGD. Let $K(n)$ denote the number of iterations required by Naive Minibatch SGD to reach an ε -stationary point; following Ghadimi & Lan (2013), we have

$$K(n) = \mathcal{O} \left(\frac{L\Delta}{\varepsilon} + \frac{\sigma^2 L\Delta}{n\varepsilon^2} \right). \quad (1.7)$$

The key difference is that the variance term is now divided by n , reflecting the variance reduction achieved by averaging n independent gradient estimates.

However, each iteration requires τ_n seconds—the time needed for the slowest worker to complete its computation. As a result, all faster workers remain idle until the slowest machine finishes before the next iteration can begin. Therefore, the total time complexity is

$$\tau_n K(n).$$

This synchronization overhead can be substantial in heterogeneous environments: when worker speeds differ greatly, the slowest device dictates the overall pace of training. In extreme cases, when τ_n is significantly larger than the others, the advantage of using multiple workers is almost entirely lost.

1.7.3 Naive Asynchronous SGD

To address the inefficiency of idle workers in synchronous methods, researchers proposed the idea of *asynchronous SGD* (Recht et al., 2011; Agarwal & Duchi, 2011). The key idea is simple: allow all workers to operate continuously and independently, without waiting for others. Each worker computes stochastic gradients on its own copy of the model, and the server performs an update immediately whenever any worker finishes its computation. We refer to this simple and naive algorithm, which updates the model as soon as any gradient arrives, as **Naive Asynchronous SGD**, and its pseudocode is given in Algorithm 1.7.3. As before, we provide a three-worker illustration of Naive Asynchronous SGD in Figure 1.7.3.

Algorithm 1.7.3 Naive Asynchronous SGD

- 1: **Input:** initial point $x^0 \in \mathbb{R}^d$, stepsizes $\gamma^k > 0$
 - 2: Workers start computing stochastic gradients at x^0
 - 3: **for** $k = 0, \dots, K - 1$ **do**
 - 4: A gradient $\nabla f(x^{k-\delta^k}; \xi_{i^k}^{k-\delta^k})$ arrives from some worker i^k
 - 5: Update the model: $x^{k+1} = x^k - \gamma^k \nabla f(x^{k-\delta^k}; \xi_{i^k}^{k-\delta^k})$
 - 6: Worker i^k begins calculating $\nabla f(x^{k+1}; \xi_{i^k}^{k+1})$
 - 7: **end for**
-

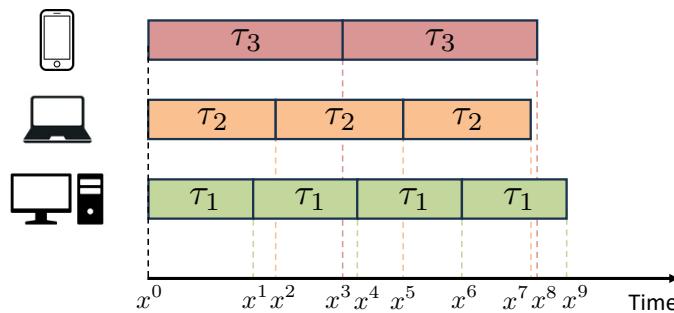


Figure 1.7.3: Illustration of Naive Asynchronous SGD (Algorithm 1.7.3) with 3 workers. Note the absence of shaded regions: all workers remain active, and every computed gradient contributes to a model update.

In Algorithm 1.7.3, the term δ^k denotes the *delay* of the gradient received at iteration k . Specifically, δ^k counts how many model updates have occurred on

the server since worker i^k started computing its gradient. Delays arise naturally because workers operate asynchronously—while one worker is computing, others may complete their gradients and perform model updates on the server.

While this algorithm ensures that all workers remain busy at all times, eliminating idle computation, it introduces a new challenge: *staleness*. Gradients computed on outdated models (large δ^k) may misguide the updates, causing the optimization to progress in the wrong direction or converge more slowly.

Over the past decade, numerous variants of asynchronous SGD have been proposed to mitigate this effect, primarily by improving how convergence bounds depend on the maximum delay (Cohen et al., 2021; Koloskova et al., 2022; Mishchenko et al., 2022a). Despite this progress, theoretical guarantees often remain pessimistic when delays are large.

A different line of research—initiated by Tyurin & Richtárik (2024)—took an alternative path. Rather than attempting to mitigate the delay issues inherent to asynchronous methods, this work turned attention back to the synchronous setting, aiming to squeeze as much performance as possible from it. They proposed a method called Rennala SGD.

1.7.4 Rennala SGD

The Rennala SGD algorithm addresses the inefficiency of Naive Minibatch SGD via a key insight: since it does not matter which worker computes a particular stochastic gradient, there is no need to collect exactly one gradient per worker. Instead, the server continuously collects gradients as they arrive and performs updates as soon as “enough” have been gathered. The algorithm is summarized in Algorithm 1.7.4. Following the same setup, we show the behavior of Rennala SGD with three workers in Figure 1.7.4.

Algorithm 1.7.4 Rennala SGD (Tyurin & Richtárik, 2024)

- 1: **Input:** initial point $x^0 \in \mathbb{R}^d$, stepsizes $\gamma^k > 0$, batch size $B \in \{1, 2, \dots\}$
 - 2: **for** $k = 0, \dots, K - 1$ **do**
 - 3: Broadcast x^k to all workers to compute stochastic gradients
 - 4: Initialize $g^k = 0$ and $b = 0$
 - 5: **while** $b < B$ **do**
 - 6: A gradient $\nabla f(x^k; \xi_{i^{k,b}}^{k,b})$ arrives from worker $i^{k,b}$
 - 7: $g^k = g^k + \nabla f(x^k; \xi_{i^{k,b}}^{k,b})$
 - 8: Worker $i^{k,b}$ immediately begins computing a new gradient at x^k
 - 9: $b = b + 1$
 - 10: **end while**
 - 11: Update the model:

$$x^{k+1} = x^k - \gamma^k \frac{g^k}{B}$$
 - 12: **end for**
-

The parameter B denotes the number of gradients that the server aims to collect before performing an update. These gradients are gathered asynchronously—as soon as they arrive—until B in total have been received. However, because Rennala SGD is ultimately a *synchronous* method, workers must synchronize once

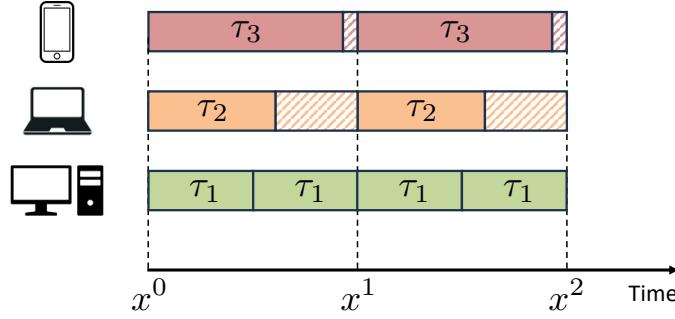


Figure 1.7.4: Illustration of Rennala SGD (Algorithm 1.7.4) with 3 workers. The shaded regions indicate idle periods or gradient computations that are eventually discarded during synchronization.

the server updates the model. Specifically, as shown in Line 3, when the server broadcasts the new model, all workers must *discard* any ongoing gradient computations and restart from the updated point. Alternatively, if the server knows the computation times of all workers, it can schedule the tasks so that the required B gradients are collected in the minimum possible time; in that case, some workers may remain *idle* once their assigned gradients are completed, waiting for the next synchronization.

The iteration complexity is analogous to the Naive Minibatch SGD case (1.7), since this is fundamentally minibatch SGD with asynchronous batch collection:

$$K(B) = \mathcal{O}\left(\frac{L\Delta}{\varepsilon} + \frac{\sigma^2 L\Delta}{B\varepsilon^2}\right).$$

Because B gradients are averaged, the variance term now scales with B instead of n .

The time per iteration, however, needs more reasoning. As shown by Tyurin & Richtárik (2024) (see also the alternative proof in Section A.1), an upper bound on the time needed to collect B gradients is given by

$$t(B) := 2 \min_{m \in \{1, \dots, n\}} \left(\frac{1}{m} \sum_{i=1}^m \frac{1}{\tau_i} \right)^{-1} \left(1 + \frac{B}{m} \right).$$

The total time complexity therefore becomes

$$t(B)K(B).$$

Remarkably, Tyurin & Richtárik (2024) proved that by choosing

$$B = \max \left\{ 1, \left\lceil \frac{\sigma^2}{\varepsilon} \right\rceil \right\},$$

Rennala SGD attains the *optimal time complexity*. They established this result by deriving lower bounds for all first-order stochastic parallel methods applied to smooth nonconvex objectives—the exact setting considered in this dissertation—and showing that Rennala SGD’s complexity matches these bounds.

One important observation is that Rennala SGD exhibits behavior similar to an

asynchronous algorithm in that it reduces idle time by allowing workers to proceed without waiting for stragglers. It almost fits the *Asynchronous Meta-Algorithm* framework in Algorithm 1.2.1. Since the update in Algorithm 1.2.1 is optional, the server can simply accumulate gradients—just as Rennala SGD does—and perform an update once B gradients have been collected. The only difference is that Algorithm 1.2.1 cannot broadcast the updated model to all workers simultaneously. However, instead of broadcasting to all workers at once, the server can send the updated model to each worker individually when they submit their next gradient. In fact, this is how Rennala SGD was originally introduced by Tyurin & Richtárik (2024).

Nevertheless, we do not classify Rennala SGD as an asynchronous algorithm due to its update rule, as described in Section 1.6.

1.8 The Beginning of This Work

Beyond establishing that Rennala SGD achieves optimal time complexity, Tyurin & Richtárik (2024) also showed—perhaps surprisingly—that no existing asynchronous SGD algorithm attains this optimality. These results raise a fundamental question:

Are asynchronous algorithms fundamentally flawed?

If the optimal solution lies within the class of synchronous algorithms, should the community abandon asynchronous SGD and focus exclusively on synchronous methods? Was the long-standing enthusiasm for asynchrony misplaced?

This question forms the central motivation of this dissertation. As we demonstrate in the chapters that follow, the answer is *no*. It is indeed possible to design asynchronous SGD algorithms that achieve optimal time complexity.

Moreover, these asynchronous methods not only attain theoretically optimal time complexity but also outperform synchronous algorithms in practice. As discussed in Section 1.6, synchronous algorithms—even the optimal ones, such as Rennala SGD—suffer from some computational waste: some devices remain idle, or their work is discarded during synchronization. Asynchronous methods can potentially eliminate this inefficiency, which may lead to additional performance gains in real-world settings.

In this dissertation, we restore the theoretical and practical significance of asynchrony by introducing new asynchronous algorithms that are both provably optimal and empirically efficient.

1.9 Contributions and Structure

This dissertation focuses on the design of asynchronous SGD algorithms that achieve *theoretically optimal time complexity*. We develop optimal algorithms for both the homogeneous and heterogeneous data settings. For the homogeneous case, we propose Ringmaster ASGD (Chapter 2); for the heterogeneous case, we introduce Ringleader ASGD (Chapter 3). The works of Maranjyan et al. (2025d); Maranjyan & Richtárik (2025) introduced these algorithms, which, to the best of our knowledge, are the first *theoretically optimal asynchronous SGD methods*.

Table 1.1: Each chapter of this dissertation introduces a main algorithmic contribution, listed in the second column together with its reference. The third column indicates whether the method achieves optimal time complexity, i.e., whether it attains the theoretical lower bound on convergence time. The fourth column specifies the data regime in which each method operates—homogeneous or heterogeneous. The fifth column describes the worker computation time model used in the analysis, while the sixth column indicates whether the method is “resource efficient”. The first two methods focus solely on minimizing wall-clock time and therefore utilize all available computational resources, whereas the third method, ATA, improves computational efficiency while losing only constant factors in the wall-clock time.

Chapter	Algorithm and Reference	Optimal time complexity	Data Heterogeneity	Worker Computation Dynamics	Resource Efficient
2	Ringmaster ASGD (Maranjyan et al., 2025d)	✓	✗	Fixed and Arbitrarily Changing	✗
3	Ringleader ASGD (Maranjyan & Richtárik, 2025)	✓	✓	Fixed and Arbitrarily Changing	✗
4	ATA ^(a) (Maranjyan et al., 2025b)	✓ ^(b)	✗	Stochastic	✓

^(a) ATA (Adaptive Task Allocation) is a meta-algorithm that can be applied on top of other methods—such as Ringmaster ASGD, Rennala SGD, or Hero SGD—to improve resource efficiency.

^(b) Here, optimality is defined in a different sense: we compare the time complexity to that of a fixed competitor that knows the distribution of worker times, and show that ATA achieves performance within a constant multiplicative factor of the best fixed allocation in expectation.

The key contributions of these chapters are summarized and discussed in more detail in Section 1.9.1 and Section 1.9.2, respectively.

Having established optimal methods in both settings, we next focus on improving their *resource efficiency*. While the optimal methods minimize the total *time to convergence*, they do so by utilizing all available computational resources. We show, however, that it is possible to slightly relax time optimality while achieving significant savings in resource usage. To address this trade-off, we develop the *Adaptive Task Allocation* (ATA) algorithm, presented in Chapter 4, which introduces strategies that make Ringmaster ASGD and related methods in the homogeneous data regime more *resource efficient*. Further discussion of these ideas is provided in Section 1.9.3.

Finally, Table 1.1 summarizes the main contributions and distinctions between the chapters. As summarized in Table 1.1, the dissertation progresses from optimal asynchronous methods (Ringmaster ASGD, Ringleader ASGD) toward resource-aware extensions (ATA). The columns emphasize the distinct aspects of each method—optimality, heterogeneity, computation-time models, and efficiency. In the following subsections, we provide a detailed overview of each chapter, clarifying the meaning of the table’s columns and elaborating on specific contributions.

1.9.1 Chapter 2 – Ringmaster ASGD: The First Asynchronous SGD with Optimal Time Complexity

In this chapter, we focus on the easier case—the homogeneous data setting. As discussed earlier in Section 1.4.1, although this setting may seem simpler, it is by

no means less important. It is not merely a simplifying theoretical assumption; rather, it represents a distinct and practically relevant regime that deserves separate study. Indeed, this setting closely reflects how most large-scale data centers operate today.

Our goal in this chapter is to design an asynchronous SGD method that achieves *optimal time complexity*—that is, whose upper bound matches the theoretical lower bound established by [Tyurin & Richtárik \(2024\)](#). In other words, we aim to construct an algorithm that is as fast as theoretically possible. Previously, such optimality had only been achieved by a *synchronous* method—[Rennala SGD](#) ([Tyurin & Richtárik, 2024](#)).

We find that it is possible to achieve such optimality in an asynchronous setting by making a simple yet crucial modification to the [Naive Asynchronous SGD](#) algorithm. Several recent works have attempted to improve [Naive Asynchronous SGD](#), for example by introducing delay-adaptive stepsizes ([Mishchenko et al., 2022a](#); [Koloskova et al., 2022](#)). While these methods outperform the synchronous Mini-batch SGD baseline, they remain suboptimal in general. Our redesign takes a different approach: it leverages one of the possible server actions introduced in Section 1.5—the ability to *discard* certain incoming stochastic gradients.

The key idea is the following. The main issue in asynchronous methods is *staleness*—some stochastic gradients are computed using outdated model parameters, which can harm convergence if the model is updated using them. The vanilla [Naive Asynchronous SGD](#) algorithm (Algorithm 1.7.3) greedily applies all received gradients, regardless of their delay. Instead, we propose to *discard* gradients that are excessively stale, i.e., those whose delay exceeds a certain threshold. By selecting this threshold carefully, we can provably attain optimal time complexity.

We call the resulting method [Ringmaster ASGD](#), inspired by the metaphor of a circus ringmaster who brings order to chaos—taming the “wild” behavior of asynchronous workers and delayed gradients that characterizes [HOGWILD!](#) ([Recht et al., 2011](#)). The algorithm is formalized as Algorithm 1.9.1.

Algorithm 1.9.1 Ringmaster ASGD

```

1: Input: initial point  $x^0 \in \mathbb{R}^d$ , stepsizes  $\gamma^k > 0$ , delay threshold  $R \geq 1$ 
2: Set  $k = 0$ 
3: Workers start computing stochastic gradients at  $x^0$ 
4: while not terminated do
5:   Gradient  $\nabla f(x^{k-\delta^k}; \xi_i^{k-\delta^k})$  arrives from worker  $i$ 
6:   if  $\delta^k < R$  then
7:     Update the model:  $x^{k+1} = x^k - \gamma^k \nabla f(x^{k-\delta^k}; \xi_i^{k-\delta^k})$ 
8:     Worker  $i$  immediately begins calculating  $\nabla f(x^{k+1}; \xi_i^{k+1})$ 
9:     Increment the iteration counter:  $k = k + 1$ 
10:    else
11:      Discard the outdated gradient  $\nabla f(x^{k-\delta^k}; \xi_i^{k-\delta^k})$ 
12:      Worker  $i$  begins calculating  $\nabla f(x^k; \xi_i^k)$ 
13:    end if
14: end while
```

Note that one can implement this more efficiently—instead of waiting for

outdated gradients to arrive and then discarding them, the server can *terminate* (another possible action described in Section 1.5) the computation of workers that are operating on excessively stale models.

Even though Ringmaster ASGD attains the same theoretical time complexity as the synchronous algorithm Rennala SGD (up to constant factors), it is often faster in practice due to two main factors:

- **No synchronization waste.** In Rennala SGD, after each model update, all devices must synchronize with the server, discarding any partial computations that were still in progress. Ringmaster ASGD avoids this waste by synchronizing only those workers whose gradients have become stale.
- **More frequent updates.** Whereas Rennala SGD performs a single update after collecting B gradients, Ringmaster ASGD updates the model whenever a fresh gradient arrives. This leads to faster progress, particularly in *sparse models*, where each worker’s gradient affects only a small subset of parameters with limited overlap across workers.

We validate this improvement through toy simulations presented in Section 2.6. This chapter is based on the work of Maranjyan et al. (2025d):

Artavazd Maranjyan, Alexander Tyurin, and Peter Richtárik. Ringmaster ASGD: The first Asynchronous SGD with optimal time complexity. In *International Conference on Machine Learning*, 2025d

1.9.2 Chapter 3 – Ringleader ASGD: The First Asynchronous SGD with Optimal Time Complexity under Data Heterogeneity

In this chapter, we turn from the homogeneous setting to the more general and more challenging *heterogeneous data* case. This setting is particularly important in applications such as *federated learning*, where data is distributed across many clients (e.g., mobile devices or organizations), and each local dataset can follow a different distribution. Moreover, these datasets are typically *private* and cannot be shared or centralized due to privacy constraints. Heterogeneity is therefore the rule rather than the exception in *federated learning*.

Let us now discuss why this setting is more difficult than the homogeneous one. Recall that our objective is to minimize the average of n local functions:

$$\underset{x \in \mathbb{R}^d}{\text{minimize}} \left\{ f(x) := \frac{1}{n} \sum_{i=1}^n f_i(x) \right\}.$$

Suppose we apply the Naive Asynchronous SGD algorithm (Algorithm 1.7.3) in this setting. The server updates the model whenever a gradient arrives from a worker. However, in the heterogeneous case, these gradients correspond to different local functions f_i , and therefore may not accurately represent the global objective f . As a result, naive asynchronous updates can introduce bias and lead to unstable training.

To address this, prior work on asynchronous optimization under data heterogeneity has typically used *similarity assumptions* between the local objectives f_i

and the global objective f (see, e.g., (Mishchenko et al., 2022a; Koloskova et al., 2022; Nguyen et al., 2022; Islamov et al., 2024)). In practice, however, such assumptions rarely hold—especially in federated learning, where data of different users or devices can vary significantly (for example, in personalized text or image data).

The only known method in this heterogeneous data setting that achieves optimal time complexity *without any similarity assumptions* is the synchronous method **Malenia SGD** (Tyurin & Richtárik, 2024)—much like the situation in the homogeneous case. Thus, the goal of this chapter is to design an *asynchronous* method that achieves optimal time complexity *without requiring any similarity assumptions*. The hope is that such an *asynchronous* method, while matching the optimal time complexity of the synchronous method **Malenia SGD**, will perform better in practice by avoiding the computational waste caused by synchronization.

We accomplish this by leveraging another possible server action described in Section 1.5—the ability to *store* received gradients rather than applying them immediately. Specifically, we maintain a *gradient table*, whose entries correspond to each worker’s most recent gradients. At each model update, the server averages all stored gradients, thereby incorporating information from all workers simultaneously. This strategy eliminates the need for similarity assumptions, since the update uses a balanced view of the entire system.

This idea builds on the concept of maintaining and aggregating stored gradients, as in the *Incremental Aggregated Gradient* (IAG) (Blatt et al., 2007; Gurubuzbalaban et al., 2017; Vanli et al., 2018) and *Stochastic Averaged Gradient* (SAG) (Roux et al., 2012; Schmidt et al., 2017) methods. An asynchronous extension of IAG, called IA²SGD, was recently proposed by Wang et al. (2025), but their algorithm does not attain optimal time complexity.

While the gradient table idea removes the need for similarity assumptions, achieving optimal time complexity requires additional care. As in the homogeneous setting, the main challenge lies in controlling *staleness*. Here, however, staleness arises because some entries in the gradient table may remain outdated if certain workers have not sent stochastic gradients for a long time. If many iterations occur without refreshing these entries, the corresponding gradients become stale and degrade performance.

To address this issue, we modify the update mechanism to avoid performing a model update upon every received stochastic gradient. Instead, we organize the *updates* into *rounds*. In each round, the server performs n updates—one per worker—and sends each worker the corresponding new model. This ensures that, in the following round, the gradients computed by the workers are based on recent models. Before the next round begins, the server clears the gradient table and starts collecting new gradients computed at these updated models. This design ensures that the delays remain bounded, resulting in an optimal method. We call the resulting algorithm **Ringleader ASGD**—a natural successor to **Ringmaster ASGD**.

A nice property of our algorithm is that it never discards any gradients: all received gradients are stored in the table and eventually used for model updates. In contrast, the optimal synchronous method **Malenia SGD** (Tyurin & Richtárik, 2024) discards some workers’ computations due to synchronization. Therefore, although **Ringleader ASGD** achieves the same optimal time complexity (up to constants), it can be faster in practice—a result we demonstrate in the experimental

section of the chapter (Section 3.7).

This chapter is based on the work of Maranjyan & Richtárik (2025):

Artavazd Maranjyan and Peter Richtárik. Ringleader ASGD: The first Asynchronous SGD with optimal time complexity under data heterogeneity. *arXiv preprint arXiv:2509.22860*, 2025

1.9.3 Chapter 4 – ATA: Adaptive Task Allocation for Efficient Resource Management in Distributed Machine Learning

In this chapter, we begin with the observation that the optimal parallel methods in the homogeneous data setting—Rennala SGD (Tyurin & Richtárik, 2024) and Ringmaster ASGD—may result in significant resource waste. The goal of these methods is purely to minimize training time, and to that end, they utilize all available resources at all times. However, this often leads to redundant computations: a large portion of the work performed by slower machines may never be used. We show that by slightly relaxing the objective of minimizing wall-clock training time, one can achieve a substantial reduction in resource waste. The saved resources can then be allocated to other jobs or tasks.

Where the Waste Comes From

To illustrate this, consider Rennala SGD, which provides a convenient example. In each iteration, the server must collect exactly B gradients before performing a model update (see Algorithm 1.7.4). All n workers continuously compute gradients until B gradients have been received. Once B gradients arrive, the server updates the model and discards any ongoing computations.

For instance, consider a scenario with $n = 1,010$ workers and $B = 10$. In this case, at least 1,000 workers’ computations will be discarded—since the server synchronizes after the first 10 gradients arrive. Ideally, we would avoid assigning tasks to those slower workers whose results will eventually be discarded. However, this would require knowing in advance how long each worker will take to compute its gradient. If these computation times were fixed and known, we could simply select the optimal subset of workers to minimize total computation time. We instead consider a more challenging setup, in which the computation times are random variables. Let us now formalize this idea.

Problem Formulation

We consider a system of n workers, each capable of repeatedly performing the same task (e.g., computing a stochastic gradient). In each round, the algorithm has a budget of B tasks to distribute among the workers. Allocating one unit of budget to a worker corresponds to assigning it one task.

We define the set of all feasible allocations as

$$\mathcal{A} := \left\{ a \in \mathbb{Z}_+^n : \sum_{i=1}^n a_i = B \right\},$$

where $a = (a_1, \dots, a_n)$ specifies how the total budget of B tasks is distributed among the n workers—that is, the i -th worker receives a_i tasks—and \mathbb{Z}_+ denotes the set of nonnegative integers.

Fixed Computation Times

Under the fixed-time model described in (1.5), where each worker i requires τ_i seconds per task, the time to receive B completed gradients under allocation $a \in \mathcal{A}$ is

$$C(a) := \max_{i \in \{1, \dots, n\}} a_i \tau_i .$$

Hence, the optimal allocation can be found by solving the combinatorial problem

$$a^* \in \arg \min_{a \in \mathcal{A}} C(a) .$$

This allocation achieves the same wall-clock time as the greedy asynchronous collection strategy used in Algorithm 1.7.4, but with the minimum possible total computation, avoiding any wasted work.

Random Computation Times

In practice, however, worker computation times are not fixed and may change during the training. We model this by assuming that the computation time of worker i follows an unknown distribution ν_i . In round k , the time to complete the u -th task of worker i is denoted by $X_i^{k,u}$, with $X_i^{k,u} \sim \nu_i$ iid across u . If worker i performs a_i^k tasks in round k , the total computation time for that worker is

$$\sum_{u=1}^{a_i^k} X_i^{k,u}$$

with the convention that this sum equals 0 when $a_i^k = 0$.

Thus, the time to collect B completed gradients under allocation a^k is

$$C(a^k) := \max_{i \in \{1, \dots, n\}} \sum_{u=1}^{a_i^k} X_i^{k,u} .$$

Over K rounds, the total expected computation time is

$$\mathcal{C}_K := \mathbb{E} \left[\sum_{k=1}^K C(a^k) \right] = \sum_{k=1}^K \mathbb{E} [C(a^k)] .$$

If the distributions ν_i were known, the optimal allocation would be obtained by solving

$$a^* \in \arg \min_{a \in \mathcal{A}} \mathbb{E} [C(a)] ,$$

and this same allocation would then be used in every round. We refer to this strategy as the Optimal Fixed Task Allocation (OFTA).

Goal: Adaptive Strategy without Knowing the Distributions

In practice, the distributions ν_i are unknown. Our goal is to design an adaptive allocation strategy whose total computation time \mathcal{C}_K satisfies

$$\mathcal{C}_K \leq c \cdot \mathcal{C}_K^* + \mathcal{E}_K , \quad (1.8)$$

where $c \geq 1$ is a constant close to 1, and \mathcal{E}_K is a negligible term compared to \mathcal{C}_K^* when $K \rightarrow \infty$. This guarantee ensures that, asymptotically, our adaptive method performs nearly as well as the optimal strategy that knows all ν_i .

The ATA Algorithm

We achieve this with our algorithm, Adaptive Task Allocation (ATA), which formulates the problem as a non-linear stochastic multi-armed bandit (MAB) problem (Lattimore & Szepesvári, 2020). In the standard MAB setting, an agent repeatedly chooses among several “arms” (here, workers), each associated with an unknown random outcome. The agent must balance two objectives:

- **Exploration:** learning the distribution of each arm.
- **Exploitation:** favoring the arms currently believed to yield better outcomes (faster workers, in our case).

In the MAB literature, this balance is often achieved using confidence bounds as estimators of each arm’s mean reward (or cost). In particular, algorithms based on Lower Confidence Bounds (LCBs) select arms using pessimistic estimates, ensuring sufficient exploration while minimizing regret (Auer, 2002).

Following this principle, ATA maintains LCBs for each worker’s expected computation time, updating them from observed samples. At each iteration, given the LCB vector $s^k = (s_1^k, \dots, s_n^k)$ and total budget B , the algorithm selects the next allocation by solving

$$a^k \in \arg \min_{a \in \mathcal{A}} \max_{i \in \{1, \dots, n\}} a_i^k s_i^k .$$

This rule balances exploration and exploitation in a principled way and yields the performance guarantee stated in (1.8).

The Asynchronous Case

Since Ringmaster ASGD is the asynchronous version of Rennala SGD, the ATA algorithm can also be applied on top of it. In this setting, the allocation computed by ATA determines how many gradients each worker contributes, but instead of waiting to collect all gradients before updating the model, the server performs a model update immediately upon receiving each gradient. At the same time, the server ensures that delays remain bounded by sending fresh models to any workers whose local copies become outdated during the process.

This chapter is based on the work of Maranjyan et al. (2025b):

Artavazd Maranjyan, El Mehdi Saad, Peter Richtárik, and Francesco Orabona. ATA: Adaptive task allocation for efficient resource management in distributed machine learning. In *International Conference on Machine Learning*, 2025b

1.10 Publications Not Included in This Dissertation

To maintain a coherent narrative and focus, several papers that I wrote during my PhD were not included in this dissertation. For completeness, I also list earlier publications from before the PhD period. All such works are summarized in Table 1.2.

Table 1.2: Publications not included in this dissertation.

Reference and Title	During PhD
Maranjyan et al. (2025a) MindFlayer SGD: Efficient Parallel SGD in the Presence of Heterogeneous and Random Worker Compute Times	✓
Condat et al. (2025) LoCoDL: Communication-Efficient Distributed Learning with Local Training and Compression	✓
Maranjyan et al. (2024) Differentially Private Random Block Coordinate Descent	✓
Maranjyan et al. (2025c) GradSkip: Communication-Accelerated Local Gradient Methods with Better Computational Complexity	✗
Grigoryan et al. (2023) Menshov-Type Theorem for Divergence Sets of Sequences of Localized Operators	✗
Grigoryan & Maranjyan (2021a) On the divergence of Fourier series in the general Haar system	✗
Grigoryan & Maranjyan (2021b) On the unconditional convergence of Faber-Schauder series in L^1	✗

Among these, let me briefly highlight one relevant work. In Maranjyan et al. (2025a), we studied Rennala SGD (Tyurin & Richtárik, 2024)—the optimal synchronous SGD algorithm in the homogeneous data setting—under the assumption that each worker’s computation time is a random variable. We showed that Rennala SGD can become significantly slower when the distributions of the workers’ computation times have heavy tails. To address this issue, we proposed a new algorithm, MindFlayer SGD, which mitigates the effect of extreme delays by imposing a threshold on the maximum time allowed for each worker to compute its gradient. Our analysis demonstrated that MindFlayer SGD consistently outperforms Rennala SGD and can achieve arbitrarily large speedups as the skewness of these distributions increases.

1.11 Common Notations

This section summarizes the common mathematical notations used throughout the dissertation.

Basic Sets

We denote by \mathbb{R} the set of real numbers and by $\mathbb{R}_+ := [0, \infty)$ the set of nonnegative reals. The set of natural numbers is denoted by $\mathbb{N} := \{1, 2, \dots\}$, and the set of nonnegative integers by $\mathbb{Z}_+ := \{0, 1, 2, \dots\}$. For a positive integer n , we use the shorthand notation $[n] := \{1, 2, \dots, n\}$.

Vectors and Norms

Let $d \in \mathbb{N}$ denote the dimension of the vectors. For vectors $a, b \in \mathbb{R}^d$, the standard inner product is defined as

$$\langle a, b \rangle := \sum_{i=1}^d a_i b_i ,$$

where a_i and b_i are coordinates and the corresponding Euclidean norm as

$$\|a\| := \sqrt{\langle a, a \rangle} .$$

We also use $\|\cdot\|_1$ and $\|\cdot\|_\infty$ to denote the ℓ_1 and ℓ_∞ norms, respectively:

$$\|a\|_1 := \sum_{i=1}^d |a_i| , \quad \|a\|_\infty := \max_{1 \leq i \leq d} |a_i| .$$

The symbol \odot denotes the element-wise (Hadamard) product of two vectors. For any scalar $x \in \mathbb{R}$, we write $(x)_+ := \max\{x, 0\}$.

For integers $a, b \in \mathbb{Z}$, we write $a \bmod b$ to denote the remainder when a is divided by b , that is,

$$a \bmod b := a - b \left\lfloor \frac{a}{b} \right\rfloor .$$

Probability and Expectation

Expectation with respect to a random variable is denoted by $\mathbb{E}[\cdot]$. For an event \mathcal{E} , we write $\neg\mathcal{E}$ for its complement (i.e., the event that \mathcal{E} does not occur). We denote by $\mathbb{1}(\cdot)$ the indicator function, which takes the value 1 if the condition inside the parentheses is true and 0 otherwise.

Asymptotic Notation

For functions $\phi, \psi : \mathcal{X} \rightarrow \mathbb{R}$, we use the standard asymptotic symbols:

- $\phi = \mathcal{O}(\psi)$ means that there exists a constant $C > 0$ such that

$$\phi(x) \leq C \psi(x), \quad \forall x \in \mathcal{X}.$$

- $\phi = \Omega(\psi)$ means that there exists a constant $C > 0$ such that

$$\phi(x) \geq C \psi(x), \quad \forall x \in \mathcal{X}.$$

- $\phi = \Theta(\psi)$ means that both $\phi = \mathcal{O}(\psi)$ and $\phi = \Omega(\psi)$ hold, i.e., ϕ and ψ are of the same order up to constant factors.

Chapter 2

Ringmaster ASGD: The First Asynchronous SGD with Optimal Time Complexity

This chapter is based on the work of Maranjyan et al. (2025d):

Artavazd Maranjyan, Alexander Tyurin, and Peter Richtárik. Ringmaster ASGD: The first Asynchronous SGD with optimal time complexity. In *International Conference on Machine Learning*, 2025d

2.1 Introduction

We consider stochastic nonconvex optimization problems of the form

$$\underset{x \in \mathbb{R}^d}{\text{minimize}} \{f(x) := \mathbb{E}_{\xi \sim \mathcal{D}} [f(x; \xi)]\},$$

where $f : \mathbb{R}^d \times \mathbb{S}_\xi \rightarrow \mathbb{R}$, \mathbb{R}^d is a linear space, and \mathbb{S}_ξ is a sample space. In machine learning, $f(x; \xi)$ denotes the loss of a model parameterized by x on a data sample ξ , and \mathcal{D} denotes the distribution of the training dataset. In nonconvex optimization, our goal is to find an ε -stationary point, i.e., a (random) vector $x \in \mathbb{R}^d$ such that $\mathbb{E}[\|\nabla f(x)\|^2] \leq \varepsilon$.

We consider a setup involving n workers (e.g., CPUs, GPUs, servers), each with access to the same distribution \mathcal{D} . Each worker is capable of computing independent, unbiased stochastic gradients with bounded variance (Assumption 2.1.3). We consider a setup with asynchronous, heterogeneous, and varying computation speeds. We aim to account for all potential scenarios, such as random outages, varying computational performance over time, and the presence of slow or straggling workers (Dean & Barroso, 2013).

This setup is common in both data center environments (Dean et al., 2012) and federated learning (Konečný et al., 2016b; McMahan et al., 2016; Kairouz et al., 2021) for distributed training. Although parallelism facilitates rapid convergence, variations in worker speeds make effective coordination more challenging.

Asynchronous Stochastic Gradient Descent is a popular approach for parallelization in such distributed settings. We begin with a simple variant that keeps all workers continuously active and performs an update whenever a gradient arrives; we refer to this method as **Naive Asynchronous SGD**. The method is outlined in Algorithm 2.1.1.

Table 2.1: The time complexities of asynchronous stochastic gradient methods, which perform the step $x^{k+1} = x^k - \gamma^k \nabla f(x^{k-\delta^k}; \xi_{i^k}^{k-\delta^k})$, to get an ε -stationary point in the nonconvex setting. In this table, we consider the *fixed computation model* from Section 2.2. Abbr.: σ^2 is defined as $\mathbb{E}_\xi[\|\nabla f(x; \xi) - \nabla f(x)\|^2] \leq \sigma^2$ for all $x \in \mathbb{R}^d$, L is the smoothness constant of f , $\Delta := f(x^0) - f^*$, $\tau_i \in [0, \infty]$ is the time bound to compute a single stochastic gradient by worker i^k .

Algorithm	Worst-Case Time Complexity	Optimal	Adaptive to Varying Speeds
Delay-Adaptive Asynchronous SGD (Koloskova et al., 2022) (Mishchenko et al., 2022a)	$\left(\frac{1}{n} \sum_{i=1}^n \frac{1}{\tau_i}\right)^{-1} \left(\frac{L\Delta}{\varepsilon} + \frac{\sigma^2 L\Delta}{n\varepsilon^2}\right)$	✗	✓
Naive Optimal ASGD (new) (Algorithm 2.2.1; Theorem 2.2.1)	$\min_{m \in [n]} \left(\frac{1}{m} \sum_{i=1}^m \frac{1}{\tau_i}\right)^{-1} \left(\frac{L\Delta}{\varepsilon} + \frac{\sigma^2 L\Delta}{m\varepsilon^2}\right)$	✓	✗
Ringmaster ASGD (new) (Algorithms 2.3.1 or 2.3.2; Theorem 2.4.3)	$\min_{m \in [n]} \left(\frac{1}{m} \sum_{i=1}^m \frac{1}{\tau_i}\right)^{-1} \left(\frac{L\Delta}{\varepsilon} + \frac{\sigma^2 L\Delta}{m\varepsilon^2}\right)$	✓	✓
Lower Bound (Tyurin & Richtárik, 2024)	$\min_{m \in [n]} \left(\frac{1}{m} \sum_{i=1}^m \frac{1}{\tau_i}\right)^{-1} \left(\frac{L\Delta}{\varepsilon} + \frac{\sigma^2 L\Delta}{m\varepsilon^2}\right)$	—	—

Algorithm 2.1.1 Naive Asynchronous SGD

- 1: **Input:** point $x^0 \in \mathbb{R}^d$, stepsizes $\gamma^k \geq 0$
 - 2: Workers start computing stochastic gradients at x^0
 - 3: **for** $k = 0, \dots, K - 1$ **do**
 - 4: A gradient $\nabla f(x^{k-\delta^k}; \xi_{i^k}^{k-\delta^k})$ arrives from worker i^k
 - 5: Update: $x^{k+1} = x^k - \gamma^k \nabla f(x^{k-\delta^k}; \xi_{i^k}^{k-\delta^k})$
 - 6: Worker i^k begins calculating $\nabla f(x^{k+1}; \xi_{i^k}^{k+1})$
 - 7: **end for**
-

This is a greedy and asynchronous method. Once a worker finishes computation of the stochastic gradient, it immediately sends the gradient to the server, which updates the current iterate without waiting for other workers. Notice that, unlike vanilla SGD, the update is performed using the stochastic gradient calculated at the point $x^{k-\delta^k}$, where the index $k - \delta^k$ corresponds to the iteration when the worker started computing the gradient, which can be significantly outdated. The sequence $\{\delta^k\}$ is a sequence of delays of asynchronous SGD, where $\delta^k \geq 0$ is defined as the difference between the iteration when worker i^k started computing the gradient and iteration k , when it was applied.

Asynchronous SGD methods have a long history, originating in 1986 (Tsitsiklis et al., 1986) and regaining prominence with the seminal work of Recht et al. (2011). The core idea behind asynchronous SGD is simple: to achieve fast convergence, all available resources are utilized by keeping all workers busy at all times. This principle has been validated in numerous studies, showing that asynchronous SGD can outperform naive synchronous SGD methods (Feyzmahdavian et al., 2016; Dutta et al., 2018; Nguyen et al., 2018; Arjevani et al., 2020; Cohen et al., 2021; Mishchenko et al., 2022a; Koloskova et al., 2022; Islamov et al., 2024; Feyzmahdavian & Johansson, 2023).

2.1.1 Assumptions

In this paper, we consider the standard assumptions from the nonconvex world.

Assumption 2.1.1 (Smoothness). The function f is differentiable, and its gradient is Lipschitz continuous. That is, there exists a constant $L > 0$ such that, for all $x, y \in \mathbb{R}^d$,

$$\|\nabla f(x) - \nabla f(y)\| \leq L \|x - y\|.$$

This assumption, often referred to as *L-smoothness*, controls how quickly the gradient can change.

Assumption 2.1.2 (Lower boundedness). The objective function is bounded below: there exists $f^* > -\infty$ such that $f(x) \geq f^*$ for all $x \in \mathbb{R}^d$.

This assumption ensures that the optimization problem is well-posed: without it, function values could decrease without bound, making convergence guarantees meaningless.

We define $\Delta := f(x^0) - f^*$, where x^0 is the initial point of the optimization algorithm. The quantity Δ represents the initial suboptimality and serves as a measure of the inherent difficulty of the problem.

Assumption 2.1.3. For every ξ the function $f(x; \xi)$ is differentiable with respect to its first argument x . Moreover, the stochastic gradients are unbiased and have bounded variance $\sigma^2 \geq 0$, that is,

$$\begin{aligned}\mathbb{E}_\xi [\nabla f(x; \xi)] &= \nabla f(x), \quad \forall x \in \mathbb{R}^d, \\ \mathbb{E}_\xi [\|\nabla f(x; \xi) - \nabla f(x)\|^2] &\leq \sigma^2, \quad \forall x \in \mathbb{R}^d.\end{aligned}$$

2.1.2 Related Work

Despite the variety of asynchronous SGD algorithms proposed over the years, a fundamental question remained unresolved: *What is the optimal strategy for parallelization in this setting?*

When we have one worker, the optimal number of stochastic gradients required to find an ε -stationary point is

$$\Theta\left(\frac{L\Delta}{\varepsilon} + \frac{\sigma^2 L \Delta}{\varepsilon^2}\right),$$

achieved by the vanilla SGD method (Ghadimi & Lan, 2013; Arjevani et al., 2022). In the parallel setting with many workers, several approaches have been proposed to obtain oracle lower bounds (Scaman et al., 2017; Woodworth et al., 2018; Arjevani et al., 2020; Lu & De Sa, 2021). Recent work by Tyurin & Richtárik (2024); Tyurin (2024) addressed the question by establishing lower bounds for the *time complexity* of asynchronous methods under the *fixed computation model* and the *universal computation model*. Surprisingly, they demonstrated that none of the existing asynchronous SGD methods are optimal. Moreover, they introduced

Algorithm 2.1.2 Rennala SGD (Tyurin & Richtárik, 2024)

```

1: Input: point  $x^0 \in \mathbb{R}^d$ , stepsize  $\gamma > 0$ , batch size  $B \in \mathbb{N}$ 
2: Workers start computing stochastic gradients at  $x^0$ 
3: for  $k = 0, \dots, K - 1$  do
4:   Initialize  $g^k = 0$  and  $b = 0$ 
5:   while  $b < B$  do
6:     A gradient  $\nabla f(x^{k-\delta^{k,b}}; \xi^{k,b})$  arrives from worker  $i^{k,b}$ 
7:     if  $\delta^{k,b} = 0$  then
8:        $g^k = g^k + \nabla f(x^{k-\delta^{k,b}}; \xi^{k,b})$ 
9:     end if
10:    Worker  $i^{k,b}$  immediately begins computing a new gradient at  $x^k$ 
11:     $b = b + 1$ 
12:   end while
13:   Update the model:

$$x^{k+1} = x^k - \gamma \frac{g^k}{B}$$

14: end for

```

a minimax optimal method, Rennala SGD, which achieves the theoretical lower bound for time complexity.

Rennala SGD is *semi-asynchronous* and can be viewed as Minibatch SGD (which takes *synchronous* iteration/model updates) combined with an *asynchronous* mini-batch collection mechanism. Let us now explain how Rennala SGD works, in the notation of Algorithm 2.1.2, which facilitates comparison with further methods described in this work. Due to the condition $\delta^{k,b} = 0$, which ignores all stochastic gradients calculated at the previous points, Rennala SGD performs the step

$$x^{k+1} = x^k - \gamma \frac{1}{B} \sum_{j=1}^B \nabla f(x^k; \xi^{k,j}) ,$$

where $\xi^{k,1}, \dots, \xi^{k,B}$ are independent samples from \mathcal{D} collected asynchronously across all workers. Note that the workers compute the stochastic gradients at the *same* point x^k , with worker i computing $B_i \geq 0$ gradients such that $\sum_{i=1}^n B_i = B$.

This approach has at least two fundamental drawbacks:

- Once the fastest worker completes the calculation of the first stochastic gradient, $\nabla f(x^k; \xi^{k,1})$, it begins computing another stochastic gradient at the same point x^k , even though it already possesses additional information from $\nabla f(x^k; \xi^{k,1})$. Rennala SGD does not update iterate x^k immediately.
- Once Rennala SGD finishes the inner while loop, it will ignore all stochastic gradients that were being calculated before the loop ended, even if a worker started the calculation just a moment before. In contrast, Naive Asynchronous SGD avoids these issues by fully utilizing all currently available information when asking a worker to calculate the next stochastic gradient and not ignoring any stochastic gradients.

These revelations raise an intriguing question: *Is asynchronous parallelization fundamentally flawed?* If the optimal solution lies in synchronous approaches,

should the community abandon asynchronous SGD and redirect its focus to developing synchronous methods? Perhaps the widespread enthusiasm for asynchronous SGD methods was misplaced.

Alternatively, could there be a yet-to-be-discovered variant of asynchronous SGD that achieves optimal time complexity? In this work, we answer this question affirmatively. We reestablish the prominence of asynchronous SGD by proposing a novel asynchronous optimization method that attains optimal time complexity.

2.1.3 Contributions

Our contributions are summarized as follows:

- We introduce a novel asynchronous stochastic gradient descent method, **Ringmaster ASGD**, described in Algorithm 2.3.1 and Algorithm 2.3.2. This is the first asynchronous method to achieve optimal time complexity under arbitrary heterogeneous worker compute times (see Table 2.1). Specifically, in Theorems 2.4.3 and 2.5.2, we establish time complexities that match the lower bounds developed by Tyurin & Richtárik (2024); Tyurin (2024).
- Our work begins with another new optimal method, **Naive Optimal ASGD** (Algorithm 2.2.1). We demonstrate that Naive Optimal ASGD achieves optimality under the fixed computation model. However, we find that it is overly simplistic and lacks robustness in scenarios where worker computation times are chaotic and dynamic. To address this limitation, we designed **Ringmaster ASGD**, which combines the strengths of Naive Optimal ASGD, previous non-optimal versions of Naive Asynchronous SGD methods (Cohen et al., 2021; Koloskova et al., 2022; Mishchenko et al., 2022a), and the semi-synchronous Rennala SGD (Tyurin & Richtárik, 2024).
- All our claims are supported by rigorous theoretical analysis showing the optimality of the method under virtually any computation scenario, including unpredictable downtimes, fluctuations in computational performance over time, delays caused by slow or straggling workers, and challenges in maintaining synchronization across distributed systems (see Sections 2.4 and 2.5). Using numerical experiments, we demonstrate that **Ringmaster ASGD** outperforms existing methods (see Section 2.6).

2.2 Preliminaries and Naive Method

To compare methods, we consider the *fixed computation model* (Mishchenko et al., 2022a). In this model, it is assumed that

Each worker i requires τ_i seconds to compute one stochastic gradient $\nabla f(x; \xi)$. (2.1)

Without loss of generality, we assume

$$0 < \tau_1 \leq \tau_2 \leq \dots \leq \tau_n .$$

However, in Section 2.5, we will discuss how one can easily generalize our result to arbitrary computational dynamics, e.g., when the computation times are not bounded by the fixed values $\{\tau_i\}$, and can change in arbitrary/chaotic manner

in time. Under the fixed computation model, Tyurin & Richtárik (2024) proved that the optimal time complexity lower bound is

$$T_R := \Theta \left(\min_{m \in [n]} \left(\frac{1}{m} \sum_{i=1}^m \frac{1}{\tau_i} \right)^{-1} \left(\frac{L\Delta}{\varepsilon} + \frac{\sigma^2 L\Delta}{m\varepsilon^2} \right) \right) \quad (2.2)$$

seconds achieved by Rennala SGD (Algorithm 2.1.2). However, the best analysis of Naive Asynchronous SGD (Koloskova et al., 2022; Mishchenko et al., 2022a) with appropriate stepsizes achieves the time complexity (see Sec. L of Tyurin & Richtárik (2024))

$$T_A := \Theta \left(\left(\frac{1}{n} \sum_{i=1}^n \frac{1}{\tau_i} \right)^{-1} \left(\frac{L\Delta}{\varepsilon} + \frac{\sigma^2 L\Delta}{n\varepsilon^2} \right) \right). \quad (2.3)$$

Note that $T_R \leq T_A$; this is because $\min_{m \in [n]} g(m) \leq g(n)$ for any function $g : \mathbb{N} \rightarrow \mathbb{R}$. Moreover, T_R can *arbitrarily* smaller. To illustrate the difference, consider an example with $\tau_i = \sqrt{i}$ for all $i \in [n]$. Then,

$$T_R = \Theta \left(\max \left\{ \frac{\sigma L\Delta}{\varepsilon^{3/2}}, \frac{L\Delta\sigma^2}{\sqrt{n}\varepsilon^2} \right\} \right)$$

and

$$T_A = \Theta \left(\max \left\{ \frac{\sqrt{n}L\Delta}{\varepsilon}, \frac{L\Delta\sigma^2}{\sqrt{n}\varepsilon^2} \right\} \right),$$

see the derivations in Section A.5. If n is large, as is often encountered in modern large-scale training scenarios, T_A can be arbitrarily larger than T_R . Thus, the best-known variants of asynchronous SGD are not robust to the scenarios when the number of workers is large and computation times are heterogeneous/chaotic.

2.2.1 A Naive Optimal Asynchronous SGD

We now introduce our first simple and effective strategy to improve the time complexity T_A . Specifically, we hypothesize that selecting a *subset* of workers at the beginning of the optimization process, instead of utilizing all available workers, can lead to a more efficient and stable approach. As we shall show, this adjustment not only simplifies the computational dynamics, but also proves sufficient to achieve the optimal time complexity.

The idea is to select the fastest $[m] := \{1, 2, \dots, m\}$ workers, thereby ignoring the slow ones and eliminating delayed gradient updates. We demonstrate that the optimal algorithm involves finding the ideal number of workers m and running delay-adaptive version of Naive Asynchronous SGD (Koloskova et al., 2022; Mishchenko et al., 2022a) on those workers. The method is formalized in Algorithm 2.2.1.

Algorithm 2.2.1 Naive Optimal ASGD

1: Find

$$m_* \in \arg \min_{m \in [n]} \left(\frac{1}{m} \sum_{i=1}^m \frac{1}{\tau_i} \right)^{-1} \left(1 + \frac{\sigma^2}{m\varepsilon} \right)$$

2: Run Naive Asynchronous SGD (Algorithm 2.1.1) on $[m_*]$ workers

The choice of m_* in Algorithm 2.2.1 effectively selects the fastest m_* workers only. Note that it is possible for m_* to be equal to n , meaning that all workers participate, which occurs when all workers are nearly equally fast. In this case, the harmonic mean in Line 1 of Algorithm 2.2.1 remains unchanged if all τ_i s are equal, but the right-hand side decreases. However, if some workers experience large delays, the harmonic mean in Line 1 increases as m grows, introducing a trade-off between the two factors. Conversely, if most workers are very slow, it may be optimal to have as few as one worker participating.

Next, we can easily prove that our algorithm, Naive Optimal ASGD (Algorithm 2.2.1), is optimal in terms of time complexity.

Theorem 2.2.1. Consider the *fixed computation model* (2.1). Let Assumptions 2.1.1, 2.1.2, and 2.1.3 hold. Then Naive Optimal ASGD (Algorithm 2.2.1) with m_* workers achieves the optimal time complexity (2.2).

Proof. The proof is straightforward. Indeed, the time complexity (2.3) of Algorithm 2.1.1 with m_* workers is

$$\Theta \left(\left(\frac{1}{m_*} \sum_{i=1}^{m_*} \frac{1}{\tau_i} \right)^{-1} \left(\frac{L\Delta}{\varepsilon} + \frac{\sigma^2 L \Delta}{m_* \varepsilon^2} \right) \right),$$

which equals to (2.2) due to the definition of m_* in Algorithm 2.2.1. □

To the best of our knowledge, this is the first variant of asynchronous SGD that provides guarantees for achieving the optimal time complexity.

2.2.2 Why Is Algorithm 2.2.1 Referred to as “Naive”?

Note that determining the optimal m_* requires the knowledge of the computation times τ_1, \dots, τ_n . If the workers’ computation times were indeed static in time, this would not be an issue, as these times could be obtained by querying a single gradient from each worker before the algorithm is run. However, in real systems, computation times are rarely static, and can vary with from iteration to iteration (Dean & Barroso, 2013; Chen et al., 2016a; Dutta et al., 2018; Maranjyan et al., 2025a), or even become infinite at times, indicating down-time.

Naively selecting the fastest m_* workers at the start of the method and keeping this selection unchanged may therefore lead to significant issues in practice. The computational environment may exhibit adversarial behavior, where worker speeds change over time. For instance, initially, the first worker may be the fastest, while the last worker is the slowest. In such cases, Naive Optimal ASGD would exclude the slowest worker. However, as time progresses, their performance

may reverse, causing the initially selected m_* workers, including the first worker, to become the slowest. This exposes a critical limitation of the strategy: it lacks robustness to time-varying worker speeds.

2.3 Ringmaster ASGD

We are now ready to present our new versions of asynchronous SGD, called Ringmaster ASGD (Algorithm 2.3.1 and Algorithm 2.3.2), which guarantee the *optimal time complexity* without knowing the computation times a priori. Both methods are equivalent, up to a minor detail that we shall discuss later. Let us first focus on Algorithm 2.3.1.

Algorithm 2.3.1 Ringmaster ASGD (without calculation stops)

```

1: Input: point  $x^0 \in \mathbb{R}^d$ , stepsize  $\gamma > 0$ , delay threshold  $R \in \mathbb{N}$ 
2: Set  $k = 0$ 
3: Workers start computing stochastic gradients at  $x^0$ 
4: while not terminated do
5:   Gradient  $\nabla f(x^{k-\delta^k}; \xi_i^{k-\delta^k})$  arrives from worker  $i$ 
6:   if  $\delta^k < R$  then
7:     Update the model:  $x^{k+1} = x^k - \gamma \nabla f(x^{k-\delta^k}; \xi_i^{k-\delta^k})$ 
8:     Worker  $i$  begins calculating  $\nabla f(x^{k+1}; \xi_i^{k+1})$ 
9:     Update the iteration number  $k = k + 1$ 
10:    else
11:      Ignore the outdated gradient  $\nabla f(x^{k-\delta^k}; \xi_i^{k-\delta^k})$ 
12:      Worker  $i$  begins calculating  $\nabla f(x^k; \xi_i^k)$ 
13:    end if
14: end while
```

Algorithm 2.3.2 Ringmaster ASGD (with calculation stops)

```

1: Input: point  $x^0 \in \mathbb{R}^d$ , stepsize  $\gamma > 0$ , delay threshold  $R \in \mathbb{N}$ 
2: Set  $k = 0$ 
3: Workers start computing stochastic gradients at  $x^0$ 
4: while not terminated do
5:   Stop calculating stochastic gradients with delays  $\geq R$ , and start computing
   new ones at  $x^k$  instead
6:   Gradient  $\nabla f(x^{k-\delta^k}; \xi_i^{k-\delta^k})$  arrives from worker  $i$ 
7:   Update the model:  $x^{k+1} = x^k - \gamma \nabla f(x^{k-\delta^k}; \xi_i^{k-\delta^k})$ 
8:   Worker  $i$  begins calculating  $\nabla f(x^{k+1}; \xi_i^{k+1})$ 
9:   Update the iteration number  $k = k + 1$ 
10: end while
```

(The core and essential modifications to Alg. 2.1.1 are highlighted. Alternatively, Alg. 2.3.1 is Alg. 2.1.1 with a specific choice of adaptive stepsizes defined by (2.4))

2.3.1 Description

Let us compare Algorithm 2.3.1 and Algorithm 2.1.1: the only difference, highlighted, lies in the fact that Algorithm 2.3.1 disregards “very old stochastic gradi-

ents”, which are gradients computed at points with significant delay. Specifically, Algorithm 2.3.1 receives $\nabla f(x^{k-\delta^k}; \xi_i^{k-\delta^k})$, compares δ^k to our *delay threshold* hyperparameter R . If $\delta^k \geq R$, then Algorithm 2.3.1 completely ignores this very outdated stochastic gradient and requests the worker to compute a new stochastic gradient at the most relevant point x^k .

Notice that Ringmaster ASGD (Algorithm 2.3.2) is Algorithm 2.1.1 with the following adaptive stepsize rule:

$$\begin{aligned}\gamma^k &= \begin{cases} \gamma, & \text{if } \bar{\delta}_i^k < R, \\ 0, & \text{if } \bar{\delta}_i^k \geq R, \end{cases} \\ \bar{\delta}_j^{k+1} &= \begin{cases} 0, & \text{if } j = i, \\ \bar{\delta}_j^k + 1, & \text{if } j \neq i \text{ and } \bar{\delta}_i^k < R, \\ \bar{\delta}_j^k, & \text{if } j \neq i \text{ and } \bar{\delta}_i^k \geq R, \end{cases}\end{aligned}\tag{2.4}$$

where i is the index of the worker whose stochastic gradient is applied at iteration k , and where we initialize the *virtual sequence of delays* $\{\bar{\delta}_j^k\}_{k,j}$ by setting $\bar{\delta}_j^0 = 0$ for all $j \in [n]$.

2.3.2 Delay Threshold

Returning to Algorithm 2.3.1, note that we have a parameter R called the *delay threshold*. When $R = 1$, the algorithm reduces to the classical SGD method, i.e.,

$$x^{k+1} = x^k - \gamma \nabla f(x^k; \xi_i^k),$$

since $\delta^k = 0$ for all $k \geq 0$. In this case, the algorithm becomes highly conservative, ignoring all stochastic gradients computed at earlier points x^{k-1}, \dots, x^0 . Conversely, if $R = \infty$, the method incorporates stochastic gradients with arbitrarily large delays, and becomes classical Naive Asynchronous SGD. Intuitively, there should be a balance—a “proper” value of R that would (i) prevent the method from being overly conservative, while (ii) ensuring stability by making sure that only informative stochastic gradients are used to update the model. We formalize these intuitions by proposing an optimal R in Section 2.4. Interestingly, the value of R does *not* depend on the computation times.

2.3.3 Why Do We Ignore the Old Stochastic Gradients?

The primary reason is that doing so enables the development of the first optimal asynchronous SGD method that achieves the lower bounds (see Sections 2.4 and 2.5). Ignoring old gradients allows us to establish tighter convergence guarantees. Intuitively, old gradients not only fail to provide additional useful information about the function f , but they can also negatively impact the algorithm’s performance. Therefore, disregarding them in the optimization process is essential for achieving our goal of developing an optimal asynchronous SGD method.

2.3.4 Comparison to Rennala SGD

Unlike Rennala SGD, which combines a synchronous Minibatch SGD update with an asynchronous minibatch collection strategy, Ringmaster ASGD is fully asyn-

chronous, which, as we show in our experiments, provided further practical advantages:

- Ringmaster ASGD updates the model immediately upon receiving a new and relevant (i.e., not too outdated) stochastic gradient. This immediate update strategy is particularly advantageous for sparse models in practice, where different gradients may update disjoint parts of the model only, facilitating faster processing. This concept aligns with the ideas of [Recht et al. \(2011\)](#), where lock-free asynchronous updates were shown to effectively leverage sparsity for improved performance.
- Ringmaster ASGD ensures that stochastic gradients computed by fast workers are never ignored. In contrast, Rennala SGD may discard stochastic gradients, even if they were recently initiated and would be computed quickly (see discussion in Section [2.1](#)).

At the same time, Ringmaster ASGD adheres to the same principles that make Rennala SGD optimal. The core philosophy of Rennala SGD is to prioritize fast workers by employing asynchronous batch collection, effectively ignoring slow workers. This is achieved by carefully choosing the batch size B in Algorithm [2.1.2](#): large enough to allow fast workers to complete their calculations, but small enough to disregard slow workers and excessively delayed stochastic gradients. Similarly, Ringmaster ASGD implements this concept using the delay threshold R .

In summary, while both Rennala SGD and Ringmaster ASGD are optimal from a theoretical perspective, the complete absence of synchronization in the latter method intuitively makes it more appealing in practice, and allows it to collect further gains which are not captured in theory. As we shall see, this intuition is supported by our experiments.

2.3.5 Comparison to Previous Asynchronous SGD Variants

[Koloskova et al. \(2022\)](#) and [Mishchenko et al. \(2022a\)](#) provided the previous state-of-the-art analysis of Naive Asynchronous SGD using delay-adaptive stepsizes. However, as demonstrated in Section [2.2](#), their analysis does not guarantee optimality. This is due to at least two factors:

- their approaches do *not* discard old stochastic gradients, instead attempting to utilize all gradients, even those with very large delays;
- although they select stepsizes γ^k that decrease as the delays increase, this adjustment may not be sufficient to ensure optimal performance, and the choice of their stepsize may be suboptimal compared to [\(2.4\)](#).

2.3.6 Stopping the Irrelevant Computations

If stopping computations is feasible, we can further enhance Algorithm [2.3.1](#) by introducing Algorithm [2.3.2](#). Instead of waiting for workers to complete the calculation of outdated stochastic gradients with delays larger than R which would not be used anyway, we propose to *stop/terminate* these computations immediately and reassign the workers to the most relevant point x^k . This adjustment

provides workers with an opportunity to catch up, as they may become faster and perform computations more efficiently at the updated point.

2.4 Theoretical Analysis

We are ready to present the theoretical analysis of Ringmaster ASGD. We start with an *iteration complexity* bound:

Theorem 2.4.1 (Proof in Section A.3). Under Assumptions 2.1.1, 2.1.2, and 2.1.3, let the stepsize in Ringmaster ASGD (Algorithm 2.3.1 or Algorithm 2.3.2) be

$$\gamma = \min \left\{ \frac{1}{2RL}, \frac{\varepsilon}{4L\sigma^2} \right\}.$$

Then

$$\frac{1}{K+1} \sum_{k=0}^K \mathbb{E} \left[\|\nabla f(x^k)\|^2 \right] \leq \varepsilon,$$

as long as

$$K \geq \frac{8RL\Delta}{\epsilon} + \frac{16\sigma^2 L\Delta}{\epsilon^2}, \quad (2.5)$$

where $R \in \{1, 2, \dots\}$ is an arbitrary delay threshold.

The classical analysis of Naive Asynchronous SGD achieves the same convergence rate, with R defined as $R \equiv \max_{k \in [K]} \delta^k$ (Stich & Karimireddy, 2020; Arjevani et al., 2020). This outcome is expected, as setting $R = \max_{k \in [K]} \delta^k$ in Ringmaster ASGD makes it equivalent to classical Naive Minibatch SGD, since no gradients are ignored. Furthermore, the analyses by Cohen et al. (2021); Koloskova et al. (2022); Mishchenko et al. (2022a) yield the same rate with $R = n$. However, it is important to note that R is a free parameter in Ringmaster ASGD that can be chosen arbitrarily. While setting $R = \max_{k \in [K]} \delta^k$ or $R = n$ effectively recovers the earlier *iteration complexities*, we show that there exists a *different* choice of R leading to optimal time complexities (see Theorems 2.4.3 and 2.5.2).

It is important to recognize that the *iteration complexity* (2.5) does *not* capture the actual “runtime” performance of the algorithm. To select an optimal value for R , we must shift our focus from *iteration complexity* to *time complexity*, which measures the algorithm’s runtime. We achieve the best practical and effective choice by optimizing the *time complexity* over R . In order to find the *time complexity*, we need the following lemma.

Lemma 2.4.2 (Proof in Appendix A.1). Let the workers’ computation times satisfy the *fixed computation model* (2.1). Let R be the delay threshold of Algorithm 2.3.1 or Algorithm 2.3.2. The time required to complete any R consecutive iterate updates of Algorithm 2.3.1 or Algorithm 2.3.2 is at most

$$t(R) := 2 \min_{m \in [n]} \left(\frac{1}{m} \sum_{i=1}^m \frac{1}{\tau_i} \right)^{-1} \left(1 + \frac{R}{m} \right). \quad (2.6)$$

Combining Theorem 2.4.1 and Lemma 2.4.2, we provide our main result.

Theorem 2.4.3 (Optimality of Ringmaster ASGD). Let Assumptions 2.1.1, 2.1.2, and 2.1.3 hold. Let the stepsize in Ringmaster ASGD (Algorithm 2.3.1 or Algorithm 2.3.2) be

$$\gamma = \min \left\{ \frac{1}{2RL}, \frac{\varepsilon}{4L\sigma^2} \right\}.$$

Then, under the *fixed computation model* (2.1), Ringmaster ASGD achieves the optimal time complexity

$$\mathcal{O} \left(\min_{m \in [n]} \left(\frac{1}{m} \sum_{i=1}^m \frac{1}{\tau_i} \right)^{-1} \left(\frac{L\Delta}{\varepsilon} + \frac{\sigma^2 L\Delta}{m\varepsilon^2} \right) \right) \quad (2.7)$$

with the delay threshold

$$R = \max \left\{ 1, \left\lceil \frac{\sigma^2}{\varepsilon} \right\rceil \right\}. \quad (2.8)$$

Note that the value of R does not in any way depend on the computation times $\{\tau_1, \dots, \tau_n\}$.

Proof. From Theorem 2.4.1, the iteration complexity of Ringmaster ASGD is

$$K = \left\lceil \frac{8RL\Delta}{\epsilon} + \frac{16\sigma^2 L\Delta}{\epsilon^2} \right\rceil. \quad (2.9)$$

Using Lemma 2.4.2, we know that Ringmaster ASGD requires at most $t(R)$ seconds to finish any R consecutive updates of the iterates. Therefore, the total time is at most

$$t(R) \times \left\lceil \frac{K}{R} \right\rceil.$$

Without loss of generality, we assume¹ that $L\Delta > \varepsilon/2$. Therefore, using (2.9), we get

$$t(R) \times \left\lceil \frac{K}{R} \right\rceil = \mathcal{O} \left(t(R) \left(\frac{L\Delta}{\epsilon} + \frac{\sigma^2 L\Delta}{R\epsilon^2} \right) \right). \quad (2.10)$$

It is left to substitute our choice (2.8) into (2.10) to get (2.7). We take (2.8), noticing that this choice of R minimizes (2.10) up to a universal constant. \square

To the best of our knowledge, this is the first result in the literature to establish the optimality of a fully asynchronous variant of SGD: the time complexity (2.7) is optimal and aligns with the lower bound established by Tyurin & Richtárik (2024).

The derived time complexity (2.7) has many nice and desirable properties.

¹Otherwise, using L -smoothness, $\|\nabla f(x^0)\|^2 \leq 2L\Delta \leq \varepsilon$, and the initial point is an ε -stationary point. See Section A.6.

First, it is robust to slow workers: if $\tau_n \rightarrow \infty$, the expression equals

$$\min_{m \in [n-1]} \left(\frac{1}{m} \sum_{i=1}^m \frac{1}{\tau_i} \right)^{-1} \left(\frac{L\Delta}{\varepsilon} + \frac{\sigma^2 L\Delta}{m\varepsilon^2} \right),$$

effectively disregarding the slowest worker. Next, assume that m_* is the smallest index that minimizes (2.7). In this case, (2.7) simplifies further to

$$\left(\frac{1}{m_*} \sum_{i=1}^{m_*} \frac{1}{\tau_i} \right)^{-1} \left(\frac{L\Delta}{\varepsilon} + \frac{\sigma^2 L\Delta}{m_* \varepsilon^2} \right).$$

This shows that the method operates effectively as if only the fastest m_* workers participate in the optimization process, resembling the idea from Algorithm 2.2.1. What is important, however, the method determines m_* adaptively and automatically.

2.4.1 The Choice of Threshold

Another notable property of the method and the time complexity result in Theorem 2.4.3 is that the threshold R is independent of the individual computation times τ_i . As a result, the method can be applied across heterogeneous distributed systems (where workers have varying speeds) while still achieving the optimal time complexity given in (2.10), up to a constant factor.

If a tighter, constant-level expression for the optimal threshold is desired, R can be computed explicitly. In that case, it will depend on the values of τ_i . The optimal R solves

$$\arg \min_{R \geq 1} t(R) \left(1 + \frac{\sigma^2}{R\varepsilon} \right),$$

which follows from (2.10) after discarding constants independent of R . Here, $t(R)$ denotes the total time required for R consecutive iterations, and is upper bounded by (2.6).

Substituting this bound yields the following expression for the optimal R :

$$R = \max \left\{ \sigma \sqrt{\frac{m^*}{\varepsilon}}, 1 \right\},$$

where

$$m^* = \arg \min_{m \in [n]} \left(\frac{1}{m} \sum_{i=1}^m \frac{1}{\tau_i} \right)^{-1} \left(1 + 2\sqrt{\frac{\sigma^2}{m\varepsilon}} + \frac{\sigma^2}{m\varepsilon} \right).$$

As the expression shows, the optimal threshold R depends on m^* , which in turn is determined by the τ_i values.

2.4.2 Proof Techniques

Several mathematical challenges had to be addressed to achieve the final result. Lemmas 2.4.2 and 2.5.1 are novel, as estimating the time complexity of the *asynchronous* Ringmaster ASGD method requires distinct approaches compared

to the *semi-synchronous* Rennala SGD method because Ringmaster ASGD is more chaotic and less predictable. Compared to the works of Koloskova et al. (2022); Mishchenko et al. (2022a), the proof of Theorem 2.4.1 is tighter and more refined, as we more carefully analyze the sum

$$\sum_{k=0}^K \mathbb{E} \left[\|x^k - x^{k-\delta^k}\|^2 \right]$$

in Lemma A.3.2. We believe that the simplicity of Ringmaster ASGD, combined with the new lemmas, the refined analysis, and the novel choice of R in Theorem 2.4.3, represents a set of non-trivial advancements that enable us to achieve the optimal time complexity.

2.5 Optimality Under Arbitrary Computation Dynamics

In the previous sections, we presented the motivation, improvements, comparisons, and theoretical results within the framework of the *fixed computation model* (2.1). We now prove Ringmaster ASGD is optimal under virtually any computation behavior of the workers. One way to formalize these behaviors is to use the *universal computation model* (Tyurin, 2024).

For each worker $i \in [n]$, we associate a *computation power* function

$$p_i : \mathbb{R}_+ \rightarrow \mathbb{R}_+$$

The number of stochastic gradients computed by worker i in $[T^0, T^1]$ is given by the integral of its computation power p_i , followed by a floor operation:

$$\text{"\# of stoch. grad. in } [T^0, T^1] = \left\lfloor \int_{T^0}^{T^1} p_i(\tau) d\tau \right\rfloor. \quad (2.11)$$

The computational power p_i characterizes the behavior of workers, accounting for potential disconnections due to hardware or network delays, variations in processing capacity over time, and fluctuations or trends in computation speeds. The integral of p_i over a given interval represents the *computation work* performed. If p_i is small within $[T^0, T^1]$, the integral is correspondingly small, indicating that worker i performs less computation. Conversely, if p_i is large over $[T^0, T^1]$, the worker is capable of performing more computation.

For our analysis, we only assume that p_i is non-negative and continuous almost everywhere². We use this assumption non-explicitly when applying the Riemann integral. The computational power p_i can even vary randomly, and all the results discussed hold conditional on the randomness of $\{p_i\}$.

Let us examine some examples. If worker i remains inactive for the first t seconds and then becomes active, this corresponds to $p_i(\tau) = 0$ for all $\tau \leq t$ and $p_i(\tau) > 0$ for all $\tau > t$. Furthermore, we allow p_i to exhibit periodic or even chaotic behavior³. The *universal computation model* reduces to the *fixed*

²Thus, it can be discontinuous and “jump” on a countable set.

³For example, $p_i(t)$ might behave discontinuously as follows: $p_i(t) = 0.5t + \sin(10t)$ for $t \leq 10$, $p_i(t) = 0$ for $10 < t \leq 20$, and $p_i(t) = \max\{80 - 0.5t, 0\}$.

computation model when $p_i(t) = 1/\tau_i$ for all $t \geq 0$ and $i \in [n]$. In this case, the number of stochastic gradients computed in the interval $[T^0, T^1]$, given by (2.11), becomes $\lfloor (T^1 - T^0)/\tau_i \rfloor$, indicating that worker i computes one stochastic gradient after $T^0 + \tau_i$ seconds, two stochastic gradients after $T^0 + 2\tau_i$ seconds, and so forth.

Note that Theorem 2.4.1 is valid under any computation model. Next, we introduce an alternative to Lemma 2.4.2 for the *universal computation model*.

Lemma 2.5.1 (Proof in Appendix A.2). Let the workers' computation times satisfy the *universal computation model*. Let R be the delay threshold of Algorithm 2.3.1 or Algorithm 2.3.2. Assume that some iteration starts at time T^0 . Starting from this iteration, the R consecutive iterate updates of Algorithm 2.3.1 or Algorithm 2.3.2 will be performed before the time

$$T(R, T^0) := \min \left\{ T \geq 0 : \sum_{i=1}^n \left\lfloor \frac{1}{4} \int_{T^0}^T p_i(\tau) d\tau \right\rfloor \geq R \right\}.$$

This result extends Lemma 2.4.2. Specifically, if $p_i(t) = 1/\tau_i$ for all $t \geq 0$ and $i \in [n]$, it can be shown that $T(R, T^0) - T^0 = \Theta(t(R))$ for all $T^0 \geq 0$.

Combining Theorem 2.4.1 and Lemma 2.5.1, we are ready to present our main result.

Theorem 2.5.2 (Optimality of Ringmaster ASGD; Proof in Appendix A.4). Let Assumptions 2.1.1, 2.1.2, and 2.1.3 hold. Let the stepsize in Ringmaster ASGD (Algorithm 2.3.1 or Algorithm 2.3.2) be

$$\gamma = \min \left\{ \frac{1}{2RL}, \frac{\varepsilon}{4L\sigma^2} \right\},$$

and delay threshold

$$R = \max \left\{ 1, \left\lceil \frac{\sigma^2}{\varepsilon} \right\rceil \right\}.$$

Then, under the *universal computation model*, Ringmaster ASGD finds an ε -stationary point after at most $T^{\bar{K}}$ seconds, where

$$\bar{K} := \left\lceil \frac{48L\Delta}{\epsilon} \right\rceil$$

and $T^{\bar{K}}$ is the \bar{K} -th element of the following recursively defined sequence:

$$T^k := \min \left\{ T \geq 0 : \sum_{i=1}^n \left\lfloor \frac{1}{4} \int_{T^{k-1}}^T p_i(\tau) d\tau \right\rfloor \geq R \right\}$$

for all $k \geq 1$ and $T^0 = 0$.

Admittedly, the obtained theorem is less explicit than Theorem 2.4.3. However, this is the price to pay for the generality of the result in terms of the computation time dynamics it allows. Determining the time complexity $T^{\bar{K}}$ requires computing T^1, T^2 , and so on, sequentially. However, in certain scenarios, $T^{\bar{K}}$ can be derived explicitly. For example, if $p_i(t) = 1/\tau_i$ for all $t \geq 0$ and $i \in [n]$, then $T^{\bar{K}}$ is given by (2.7). Furthermore, T^1 represents the number of seconds re-

quired to compute the first R stochastic gradients, T^2 represents the time needed to compute the first $2 \times R$ stochastic gradients, and so on. Naturally, to compute T^2 , one must first determine T^1 , which is reasonable given the sequential nature of the process.

The obtained result is optimal, aligns with the lower bound established by Tyurin (2024), and cannot be improved by any asynchronous parallel method.

2.6 Experiments

Asynchronous SGD has consistently demonstrated its effectiveness and practicality, achieving strong performance in various applications (Recht et al., 2011; Lian et al., 2018; Mishchenko et al., 2022a), along with numerous other studies supporting its utility. Our main goal of this paper is to refine the method further and establish that it is not only practical but also *theoretically optimal*.

We conduct a toy experiment with our proposed Ringmaster ASGD method and compare it against two baselines: the previous Naive Asynchronous SGD (referred to as Delay-Adaptive ASGD, following Mishchenko et al. (2022a)) and Rennala SGD. The optimization task is based on a convex quadratic function $f : \mathbb{R}^d \rightarrow \mathbb{R}$ defined as

$$f(x) = \frac{1}{2}x^\top Ax - b^\top x , \quad \forall x \in \mathbb{R}^d ,$$

where

$$A = \frac{1}{4} \begin{bmatrix} 2 & -1 & & 0 \\ -1 & \ddots & \ddots & \\ & \ddots & \ddots & -1 \\ 0 & & -1 & 2 \end{bmatrix} \in \mathbb{R}^{d \times d} , \quad b = \frac{1}{4} \begin{bmatrix} -1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \in \mathbb{R}^d .$$

We set $d = 1729$ and $n = 6174$. Each of the n workers computes unbiased stochastic gradients of the form

$$\nabla f(x, \xi) = \nabla f(x) + \xi ,$$

where $\xi \sim \mathcal{N}(0, 0.01^2)$.

The experiments were implemented in Python. The distributed environment was emulated on machines with Intel(R) Xeon(R) Gold 6248 CPU @ 2.50GHz. The computation times for each worker are simulated as $\tau_i = i + |\eta_i|$ for all $i \in [n]$, where $\eta_i \sim \mathcal{N}(0, i)$. We tuned the stepsize from the set $\{5^p : p \in [-5, 5]\}$. Both the batch size for Rennala SGD and the delay threshold for Ringmaster ASGD were tuned from the set $\{\lceil n/4^p \rceil : p \in \mathbb{Z}_+\}$. The experimental results are shown in Figure 2.6.1.

The obtained result confirms that Ringmaster ASGD is indeed faster than Delay-Adaptive ASGD and Rennala SGD in the considered setting. One can see the numerical experiments support that our theoretical results, and we significantly improve the convergence rate of the previous version of Naive Asynchronous SGD (Delay-Adaptive ASGD).

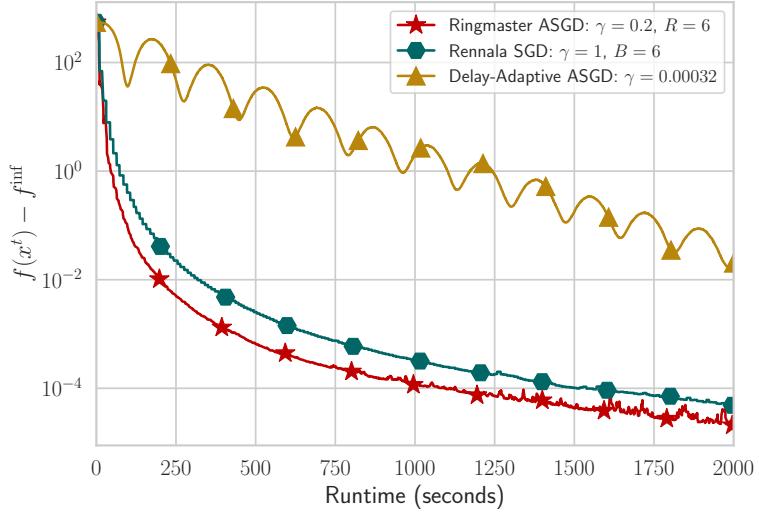


Figure 2.6.1: Experiment with $n = 6174$ and $d = 1729$ showing the convergence of Ringmaster ASGD, Delay-Adaptive ASGD, and Rennala SGD.

2.6.1 Neural Network Experiment

To show that our method also works well for neural networks, we trained a small 20-layer neural network with ReLU activation on the MNIST dataset (LeCun et al., 1998). We used the same number of workers as in the previous experiment ($n = 6174$) and kept the same time distributions. The results are shown in Figure 2.6.2.

2.7 Additional Related Work

In federated learning, communication is often more time-consuming than local computation, which motivates the use of local training strategies (McMahan et al., 2016; Mishchenko et al., 2022b; Malinovsky et al., 2022; Maranjyan et al., 2025c). Similarly, in our setting, instead of repeatedly computing gradients at the same model point, workers can perform several local gradient descent steps and send the updated model to the server when requested. Asynchronous variants of such schemes, including those with explicit time-complexity analyses, have been explored in prior works (Tyurin & Sivtsov, 2025; Fradin et al., 2025).

2.8 Conclusion and Future Work

In this work, we developed the first asynchronous SGD method, named Ringmaster ASGD, that achieves optimal time complexity. By employing a carefully designed algorithmic approach, which can be interpreted as asynchronous SGD with adaptive step sizes (2.4), we successfully reach this goal. By selecting an appropriate delay threshold R in Algorithm 2.3.1, the method attains the theoretical lower bounds established by Tyurin & Richtárik (2024); Tyurin (2024).

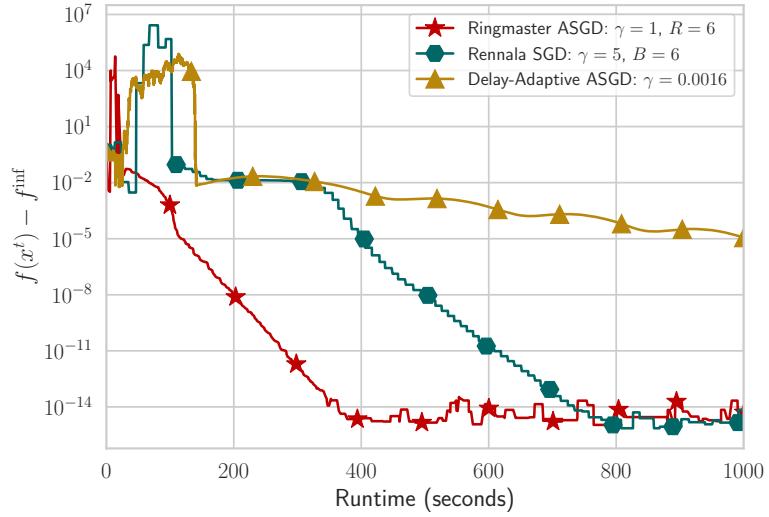


Figure 2.6.2: We run an experiment on a small 2-layer neural network with ReLU activation on the MNIST dataset, showing that our method, Ringmaster ASGD, is more robust and outperforms Delay-Adaptive ASGD and Rennala SGD.

Future work can explore heterogeneous scenarios, where the data on each device comes from different distributions, which are particularly relevant for federated learning (FL) (Konečný et al., 2016b; McMahan et al., 2016; Kairouz et al., 2021). In FL, communication costs also become crucial, making it worthwhile to consider similar extensions (Alistarh et al., 2017; Tyurin et al., 2024b; Tyurin & Richtárik, 2024). It would be also interesting to design a fully asynchronous analog to the method from (Tyurin et al., 2024a). Additionally, as discussed by Maranjyan et al. (2025a), computation and communication times can be treated as random variables. It would be valuable to investigate these cases further and derive closed-form expressions for various distributions.

Chapter 3

Ringleader ASGD: The First Asynchronous SGD with Optimal Time Complexity under Data Heterogeneity

This chapter is based on the work of Maranjyan & Richtárik (2025):

Artavazd Maranjyan and Peter Richtárik. Ringleader ASGD: The first Asynchronous SGD with optimal time complexity under data heterogeneity. *arXiv preprint arXiv:2509.22860*, 2025

3.1 Introduction

Modern machine learning increasingly depends on large-scale distributed training across clusters with hundreds or even thousands of GPUs (Shoeybi et al., 2019; Brown et al., 2020; Narayanan et al., 2021). However, classical synchronous training methods struggle to scale in these settings, as device failures, network instabilities, and synchronization overheads introduce significant inefficiencies (Chen et al., 2016a; Grattafiori et al., 2024). These issues become even more pronounced in environments with heterogeneous computational power, such as Federated Learning (FL), where devices range from high-end data center GPUs to resource-constrained edge hardware (Konečný et al., 2016b; McMahan et al., 2016; Li et al., 2020b; Kairouz et al., 2021). Because synchronous methods are bottlenecked by the slowest participants, faster devices remain idle, leading to severe underutilization of computational resources when stragglers—nodes slowed down by computation or communication—lag significantly behind.

One way to reduce synchronization bottlenecks is to equip data centers with homogeneous GPUs. However, this approach is prohibitively expensive and difficult to scale: upgrading to faster GPUs would require replacing all devices simultaneously, since heterogeneous hardware cannot be combined efficiently. Even then, homogeneity does not eliminate synchronization issues, as hardware failures and device dropouts during training still cause stragglers and idle time. Moreover, this solution applies only to controlled data center environments and is infeasible in FL, where edge devices are outside the server’s control.

A more promising approach is to shift from hardware solutions to algorithmic ones by adopting asynchronous optimization methods. These methods remove the need for synchronization, allowing fast workers to contribute updates without waiting for slower ones (Tsitsiklis et al., 1986; Recht et al., 2011; Agarwal & Duchi, 2011; Dean et al., 2012; Li et al., 2014). Despite their appeal, asynchronous methods are more difficult to analyze. In particular, a meaningful analysis would require studying *time to convergence*, rather than iteration complexity only. While iteration complexity is the traditional metric in optimization, it does not necessarily reflect real-world training speed in parallel settings: a method that performs more iterations may finish faster in wall-clock time if those iterations

can be computed without waiting for slow workers. This distinction raises a fundamental question: *among all parallel methods, which ones are provably fastest in theory?* To make this question precise, we restrict our attention to smooth non-convex problems and to stochastic first-order methods, encompassing algorithms with or without synchronization. This will be the only setting considered in this paper.

Recently, [Tyurin & Richtárik \(2024\)](#) studied this very regime, where they derived lower bounds. They then proposed two algorithms: **Rennala SGD**, designed for the *homogeneous data setting*, where all workers draw samples from the same distribution, and **Malenia SGD**, for the *heterogeneous data setting*, where data distributions differ across workers. They showed that both methods are optimal—achieving the lower bounds—and, perhaps surprisingly, both are synchronous (they periodically synchronize the workers). The key idea in both is to fully utilize the available computational resources by keeping workers continuously busy: each worker computes independently, and synchronization occurs only after a sufficient number of gradient computations have been accumulated.

At first, the result of [Tyurin & Richtárik \(2024\)](#) suggested a rather pessimistic outlook for asynchronous methods: despite their practical appeal, they showed that existing asynchronous methods are not optimal and that the method achieving the lower bound is *synchronous*. This created the view that optimality is inherently tied to synchronization. However, this view was overturned by [Maranjyan et al. \(2025d\)](#), who, in the *homogeneous data setting*, introduced **Ringmaster ASGD**—the first asynchronous SGD method to achieve the same optimal time complexity as the synchronous Rennala SGD. Although both methods share the same theoretical guarantees, Ringmaster ASGD can be faster than Rennala SGD in practice, since it avoids synchronization and benefits from more frequent updates.

Nevertheless, the work of [Maranjyan et al. \(2025d\)](#) established optimality in the *homogeneous data setting* only. The question of whether some variant of a parallel method that does not rely on synchronization (i.e., is asynchronous) can also be optimal in the more general *heterogeneous data setting* remained open. In this work, we close this gap and answer the question affirmatively.

The heterogeneous data setting is both important and practically relevant. In FL, for instance, such heterogeneity arises naturally as participants hold distinct datasets ([Zhao et al., 2018](#); [Li et al., 2020b](#); [Tan et al., 2022](#)). Yet this setting is significantly more challenging than the homogeneous one. The standard philosophy of asynchronous SGD—updating the model after every gradient computation—can be harmful here: fast workers contribute updates more frequently, causing the optimization process to become biased toward their local data. To mitigate this, most existing asynchronous methods address this issue by assuming similarity across client distributions ([Mishchenko et al., 2022a](#); [Koloskova et al., 2022](#); [Nguyen et al., 2022](#); [Islamov et al., 2024](#)). While this assumption simplifies the analysis, it is often unrealistic in practice, where clients may involve fundamentally different populations (e.g., hospitals with distinct demographics, mobile users in different countries, or financial institutions under varied regulations).

Recent work by [Wang et al. \(2025\)](#) took an important step toward removing these restrictive assumptions by proposing Incremental Aggregated Asynchronous SGD (IA²SGD), a method that provably converges without similarity assumptions. However, their method achieves the same time complexity as standard Naive Mini-

Table 3.1: Comparison of time complexities for parallel first-order methods under the fixed computation time model, where each worker i takes a fixed time τ_i to compute a stochastic gradient, with the times ordered so that τ_n is the largest (3.2). We denote by $\tau_{\text{avg}} := \frac{1}{n} \sum_{i=1}^n \tau_i$ the average computation time across all workers. The table shows how the time complexity of different algorithms depends on key problem parameters: the initial function suboptimality $\Delta := f(x^0) - f^*$ (Assumption 3.3.5), the target stationarity ε , the variance bound of the stochastic gradients σ^2 (Assumption 3.3.1), and smoothness constants. Specifically, L_f is the smoothness constant of f (Definition 3.3.3); $L_{\max} := \max_{i \in [n]} L_{f_i}$ with L_{f_i} the smoothness constant of f_i ; and L is a constant associated with our new smoothness-type assumption (Assumption 3.3.2). They satisfy $L_f \leq L \leq L_{\max}$ (Lemma 3.3.4). All stated time complexities hide universal constant factors.

Each column indicates whether a method satisfies the following desirable properties: **Optimal:** achieves the theoretical lower bound derived by Tyurin & Richtárik (2024) for parallel first-order stochastic methods in heterogeneous data setting. **No sync.:** does not require synchronization and is therefore *asynchronous*. **No idle workers:** all workers remain busy without waiting, so computational resources are fully utilized. **No discarded work:** no computation is wasted, and no worker is stopped mid-computation. Our new method, Ringleader ASGD, is the first asynchronous method to achieve optimal time complexity, while also ensuring full resource utilization (no idle workers) and no discarded computations / work.

Algorithm	Time Complexity	Optimal	No Sync.	No Idle Workers	No Discarded Work
Naive Minibatch SGD (Section 3.4.1)	$\frac{L_f \Delta}{\varepsilon} \left(\tau_n + \tau_n \frac{\sigma^2}{n\varepsilon} \right)$	✗	✗	✗	✓
IA ² SGD (Wang et al., 2025) (Section B.3)	$\frac{L_{\max} \Delta}{\varepsilon} \left(\tau_n + \tau_n \frac{\sigma^2}{n\varepsilon} \right)$ (†)	✗	✓	✓	✓
Malenia SGD (Tyurin & Richtárik, 2024)	$\frac{L_f \Delta}{\varepsilon} \left(\tau_n + \tau_{\text{avg}} \frac{\sigma^2}{n\varepsilon} \right)$	✓	✗	✓	✗
Ringleader ASGD (new) (Algorithm 3.5.1; Theorem 3.6.5)	$\frac{L \Delta}{\varepsilon} \left(\tau_n + \tau_{\text{avg}} \frac{\sigma^2}{n\varepsilon} \right)$	✓ (‡)	✓	✓	✓

(†) The analysis presented by Wang et al. (2025) is carried out under the assumption that each f_i is smooth with the same smoothness constant, which is equivalent to requiring that each f_i is L_{f_i} -smooth and then using the upper bound L_{\max} for all L_{f_i} . However, this strict assumption is not necessary: they could instead use our relaxed smoothness-type assumption (Assumption 3.3.2), in which case the constant improves to L , and their analysis remains unchanged.

(‡) The time complexities of Ringleader ASGD and Malenia SGD differ in the smoothness constant only. Since Malenia SGD is optimal, Ringleader ASGD is also optimal whenever L exceeds L_f by at most a universal constant factor, that is, $L = \mathcal{O}(L_f)$.

batch SGD (see the first two rows of Table 3.1)—the simplest synchronous SGD baseline, which waits to collect one gradient from each worker before every update—thus failing to provide the computational advantages that motivate asynchronous approaches in the first place.

To the best of our knowledge, the only method proven to be optimal in the *heterogeneous data setting* is the synchronous algorithm **Malenia SGD** of [Tyurin & Richtárik \(2024\)](#), which, notably, does not rely on similarity assumptions. However, synchronization is a major bottleneck in practice: although synchronous and asynchronous methods can share the same theoretical complexity, asynchronous methods are often faster in practice because they avoid costly synchronization and benefit from more frequent updates, as demonstrated in the homogeneous case by [Maranjyan et al. \(2025d\)](#).

This raises a fundamental question: *Is it possible to design an asynchronous method that requires no similarity assumptions while still achieving optimal time complexity?* In this paper, we answer this question affirmatively by introducing **Ringleader ASGD**, the first asynchronous SGD method that achieves optimal time complexity¹ in the *heterogeneous data setting*. Importantly, Ringleader ASGD attains this without relying on restrictive similarity assumptions.

3.1.1 Contributions

Our main contributions are the following:

- **Optimal asynchronous SGD under data heterogeneity.** We prove that Ringleader ASGD (Algorithm 3.5.1) is, to the best of our knowledge, the first asynchronous method in the heterogeneous setting under the fixed computation model (3.2) that matches the lower bounds for parallel methods established by [Tyurin & Richtárik \(2024\)](#) (Table 3.1). Crucially, it does not rely on any similarity assumptions across clients’ datasets.
- **Additional useful properties.** Beyond achieving optimal time complexity, our method Ringleader ASGD satisfies two additional desirable properties: (i) all workers remain continuously active (*no idle workers*), and (ii) every computed gradient is incorporated into the update (*no discarded work*). These properties are crucial in practice, as they ensure maximum resource utilization: all workers contribute at all times, and their computations are never wasted. Table 3.1 compares Ringleader ASGD against benchmark algorithms with respect to these properties.
- **Parameter-free design.** In contrast to the optimal synchronous method **Malenia SGD** ([Tyurin & Richtárik, 2024](#)), which requires prior knowledge of the gradient variance bound and target accuracy, our method operates in the fixed computation time model without such parameters (except for the stepsize, needed only to match the optimal theoretical rate). This makes it far more practical for real-world deployments, where these quantities are typically unknown or difficult to estimate. The same parameter-free

¹Throughout the paper, we refer to our method as optimal. Formally, this holds whenever the constant L —associated with our new smoothness-type assumption (Assumption 3.3.2)—is at most a constant factor larger than the smoothness constant L_f used in the derived lower bounds ([Tyurin & Richtárik, 2024](#)). See Table 3.1 for details.

improvement can also be extended to Malenia SGD, as we discuss in Section B.4.

- **Universal computation model.** In Section B.1, we extend our analysis beyond the fixed computation time model to the general setting of arbitrarily varying computation times, accommodating virtually any computational behavior, including stochastic or adversarial patterns, while retaining optimality time complexity.
- **Empirical validation.** In Section 3.7, we evaluate Ringleader ASGD against benchmark methods on illustrative toy problems. The results validate our theoretical findings and demonstrate clear practical advantages over the baselines.

3.2 Related Work

Research on asynchronous stochastic gradient methods dates back to the seminal work of Tsitsiklis et al. (1986), and gained renewed momentum with the introduction of the HOGWILD! algorithm (Recht et al., 2011). HOGWILD! is fundamentally an asynchronous coordinate descent method: updates are performed lock-free with inconsistent reads and writes, and its convergence guarantees rely on sparsity assumptions that are rarely realistic in modern large-scale machine learning. Subsequent refinements of this paradigm include (J Reddi et al., 2015; Zhao & Li, 2016; Pan et al., 2016; Mania et al., 2017; Leblond et al., 2018; Nguyen et al., 2018; Zhou et al., 2018), but these works remain tied to the coordinate descent setting with inconsistent memory accesses, and thus differ substantially from our focus.

Closer to our setting are works where updates are based on gradients that are applied consistently. Early contributions, typically under the homogeneous-data assumption (all workers sample from the same distribution), include the work of Agarwal & Duchi (2011), who studied convex objectives, as well as later extensions to the non-convex case such as the work of Lian et al. (2015) and Dutta et al. (2018), the latter analyzing exponentially distributed computation times. Other relevant results in this line include (Feyzmahdavian et al., 2016; Zheng et al., 2017; Arjevani et al., 2020; Feyzmahdavian & Johansson, 2023), all of which assume fixed delays. More recently, delay-adaptive methods have been proposed, aiming to improve performance by down-weighting very stale gradients (Cohen et al., 2021; Wu et al., 2022; Koloskova et al., 2022; Mishchenko et al., 2022a).

Particularly relevant to our work are asynchronous variants of SAGA. Leblond et al. (2018) developed a shared-parameter version in the spirit of HOGWILD!, while Glasgow & Wootters (2022) studied a distributed setting that employs full gradients, in contrast to our stochastic-gradient perspective.

A large body of recent work investigates asynchronous methods in federated learning (FL), where clients hold data from heterogeneous distributions. Notable contributions include (Aytekin et al., 2016; Xie et al., 2019; Mishchenko et al., 2022a; Koloskova et al., 2022; Wang et al., 2022a,b; Glasgow & Wootters, 2022; Fraboni et al., 2023; Zhang et al., 2023; Wang et al., 2024; Islamov et al., 2024; Alahyane et al., 2025).

More broadly, Assran et al. (2020) provide a comprehensive survey of asynchronous optimization methods.

There is another line of work that began with Tyurin & Richtárik (2024), who established lower bounds for parallel methods and proposed optimal synchronous algorithms together with an asynchronous counterpart. Several follow-up papers extended this semi-asynchronous framework to other settings (Tyurin et al., 2024a,b; Tyurin & Richtárik, 2024; Maranjyan et al., 2025a).

Finally, beyond asynchronous approaches, several synchronous methods address system heterogeneity by adapting local training to worker speeds. The canonical method, FedAvg (McMahan et al., 2017), performs multiple local steps on each worker. Variants have adapted the number of local steps to match workers' computation speeds (Li et al., 2020b; Maranjyan et al., 2025c), effectively balancing task assignments across heterogeneous systems. More recently, Maranjyan et al. (2025b) proposed adapting the number of local steps dynamically, without requiring prior knowledge of worker speeds.

3.3 Problem Setup

We consider a distributed learning setting with n workers, where each worker i possesses its own local data distribution \mathcal{D}_i . Our goal is to solve the following distributed optimization problem:

$$\underset{x \in \mathbb{R}^d}{\text{minimize}} \left\{ f(x) := \frac{1}{n} \sum_{i=1}^n f_i(x) \right\}, \quad \text{where } f_i(x) := \mathbb{E}_{\xi_i \sim \mathcal{D}_i} [f_i(x; \xi_i)]. \quad (3.1)$$

Here $f_i: \mathbb{R}^d \rightarrow \mathbb{R}$ denotes the local objective of worker i , defined as the expectation of the sample loss $f_i(x; \xi_i)$ over data points ξ_i drawn from its local distribution \mathcal{D}_i .

3.3.1 Worker Heterogeneity Model

We first focus on the case where workers have constant computation speeds, as this setting is more intuitive and serves as a foundational model for understanding the dynamics of asynchronous distributed optimization. The extension to arbitrary computation times is presented in Section B.1.

Following the *fixed computation model* (Mishchenko et al., 2022a), we formalize:

Each worker i requires τ_i seconds² to compute one stochastic gradient $\nabla f_i(x; \xi_i)$. (3.2)

Without loss of generality, we assume

$$0 < \tau_1 \leq \tau_2 \leq \dots \leq \tau_n.$$

We assume communication is infinitely fast (taking 0 seconds), both from workers to the server and from the server to workers³. This is a modeling choice—arguably

²One could alternatively assume that each worker requires *at most* τ_i seconds. Under this formulation, all of our upper bounds would still hold; however, the lower bound would no longer be valid. For this reason, we adopt the assumption that each worker requires exactly τ_i seconds.

³Alternatively, one could define τ_i as the time required for a worker to both compute a gradient and communicate it to the server, while keeping server-to-worker communication infinitely

the simplest one—and has been the standard assumption in prior work Mishchenko et al. (2022a); Koloskova et al. (2022); Tyurin & Richtárik (2024); Maranjyan et al. (2025d), even if not always stated explicitly. A related study by Tyurin et al. (2024b) considers the case where communication is non-negligible and proposes techniques to address it.

Finally, we denote by $\tau_{\text{avg}} := \frac{1}{n} \sum_{i=1}^n \tau_i$ the average computation time across all workers.

3.3.2 Notations

We denote the standard inner product in \mathbb{R}^d by

$$\langle x, y \rangle = \sum_{i=1}^d x_i y_i ,$$

and the corresponding Euclidean norm by $\|x\| := \sqrt{\langle x, x \rangle}$. We use $[n] := \{1, 2, \dots, n\}$ to denote the index set, and $\mathbb{E}[\cdot]$ for mathematical expectation. For functions $\phi, \psi : \mathcal{Z} \rightarrow \mathbb{R}$, we write $\phi = \mathcal{O}(\psi)$ if there exists a constant $C > 0$ such that $\phi(z) \leq C\psi(z)$ for all $z \in \mathcal{Z}$.

3.3.3 Assumptions

We consider the following standard assumptions for the nonconvex setting.

Assumption 3.3.1. For each $i \in [n]$ and every ξ_i , the function $f_i(x; \xi_i)$ is differentiable with respect to its first argument x . Moreover, the stochastic gradients are unbiased and have bounded variance $\sigma^2 \geq 0$, that is,

$$\begin{aligned} \mathbb{E}_{\xi_i \sim \mathcal{D}_i} [\nabla f_i(x; \xi_i)] &= \nabla f_i(x), \quad \forall x \in \mathbb{R}^d, \quad \forall i \in [n], \\ \mathbb{E}_{\xi_i \sim \mathcal{D}_i} [\|\nabla f_i(x; \xi_i) - \nabla f_i(x)\|^2] &\leq \sigma^2, \quad \forall x \in \mathbb{R}^d, \quad \forall i \in [n]. \end{aligned}$$

Assumption 3.3.2. Each function f_i is differentiable. There exists a constant $L > 0$ such that, for all $x \in \mathbb{R}^d$ and $y_1, \dots, y_n \in \mathbb{R}^d$,

$$\left\| \nabla f(x) - \frac{1}{n} \sum_{i=1}^n \nabla f_i(y_i) \right\|^2 \leq \frac{L^2}{n} \sum_{i=1}^n \|x - y_i\|^2 .$$

Recall the standard definition of smoothness

Definition 3.3.3 (Smoothness). A differentiable function $\phi: \mathbb{R}^d \rightarrow \mathbb{R}$ is called L_ϕ -smooth if

$$\|\nabla \phi(x) - \nabla \phi(y)\| \leq L_\phi \|x - y\|, \quad \forall x, y \in \mathbb{R}^d .$$

By convention, L_ϕ denotes the smallest such constant.

fast. Our upper bounds would still hold under this formulation, but the lower bounds would no longer apply, so we use the simpler model.

Note that Assumption 3.3.2 is stronger than requiring f itself to be L_f -smooth, yet weaker than all f_i being L_{f_i} -smooth. The following lemma establishes the relation among the constants L_f , L , and L_{f_i} .

Lemma 3.3.4 (Smoothness Bounds; Proof in Section B.2.1). Suppose Assumption 3.3.2 holds with constant $L > 0$. Then f is L_f -smooth with $L_f \leq L$. Moreover, if each f_i is L_{f_i} -smooth, then Assumption 3.3.2 is satisfied, and we have

$$L_f \leq L \leq \sqrt{\frac{1}{n} \sum_{i=1}^n L_{f_i}^2} \leq \max_{i \in [n]} L_{f_i} =: L_{\max}.$$

Finally, if all f_i are identical, i.e., $f_i = f$ for all $i \in [n]$, then $L = L_f$.

To the best of our knowledge, prior work on asynchronous SGD under data heterogeneity has always assumed smoothness of each f_i (Koloskova et al., 2022; Nguyen et al., 2022; Wang et al., 2025).

Assumption 3.3.5. There exists $f^* > -\infty$ such that $f(x) \geq f^*$ for all $x \in \mathbb{R}^d$. We define $\Delta := f(x^0) - f^*$, where x^0 is the starting point of the optimization methods.

Under these assumptions; our objective is to find an ε -stationary point—a (possibly random) vector x satisfying $\mathbb{E}[\|\nabla f(x)\|^2] \leq \varepsilon$.

3.4 Background and Motivation

In this section, we review relevant existing methods in distributed optimization and discuss their limitations to motivate our algorithmic design. We begin with Naive Minibatch SGD as the canonical synchronous baseline, then consider Maleenia SGD (Tyurin & Richtárik, 2024), the first synchronous SGD method to attain optimal time complexity. We then turn to the challenges of asynchronous approaches, focusing on the recent IA²SGD (Wang et al., 2025) and its limitations. Finally, we outline how these limitations can be addressed and introduce the core idea behind our algorithm.

3.4.1 Naive Minibatch SGD

Naive Minibatch SGD provides the most straightforward approach to solving problem (3.1) in a distributed setting.

Algorithm Description. At each iteration k , the algorithm performs the following update:

$$x^{k+1} = x^k - \gamma \frac{1}{n} \sum_{i=1}^n \nabla f_i(x^k; \xi_i^k).$$

The algorithm operates synchronously: at each iteration, the server waits to receive one stochastic gradient from each of the n workers, all of which are evaluated at the current iterate x^k . Once all gradients are collected, the server constructs an unbiased minibatch estimator of the full gradient by averaging these stochastic gradients and performs a standard gradient descent step.

Limitations. This synchronous approach has a significant computational bottleneck. Each iteration requires waiting for the slowest worker to complete its gradient computation, resulting in the iteration time $\tau_n := \max_{i \in [n]} \tau_i$ (3.2). Consequently, faster workers remain idle while waiting for stragglers, which leads to inefficient utilization of available computational resources.

Theoretical Performance. In terms of convergence rate, Naive Minibatch SGD achieves the iteration complexity

$$\mathcal{O} \left(\frac{L_f \Delta}{\varepsilon} \left(1 + \frac{\sigma^2}{n \varepsilon} \right) \right)$$

to reach an ε -stationary point (Cotter et al., 2011; Goyal et al., 2017; Gower et al., 2019). The corresponding time complexity becomes

$$\mathcal{O} \left(\frac{\tau_n L_f \Delta}{\varepsilon} \left(1 + \frac{\sigma^2}{n \varepsilon} \right) \right).$$

This motivates the development of methods that can better utilize fast workers without waiting for stragglers.

3.4.2 Malenia SGD

Malenia SGD (Tyurin & Richtárik, 2024) resolves the straggler issue in Naive Minibatch SGD by ensuring continuous worker utilization, rather than waiting for the slowest worker in each iteration. However, it is fundamentally a Minibatch SGD algorithm: in every iteration, it constructs an unbiased gradient estimator from multiple gradients and then performs a synchronous update. The key distinction lies in how the batch is collected—Malenia SGD gathers gradients asynchronously, allowing potentially multiple contributions from the same worker within a single iteration, unlike Naive Minibatch SGD.

Algorithm Description. At each iteration k , all workers continuously compute stochastic gradients at the current point x^k . The server accumulates these gradients until the stopping condition

$$\left(\frac{1}{n} \sum_{i=1}^n \frac{1}{b_i^k} \right)^{-1} \geq \max \left\{ 1, \frac{\sigma^2}{n \varepsilon} \right\} \quad (3.3)$$

is satisfied, where b_i^k denotes the number of gradients received from worker i . Once this condition is met, the server performs the update

$$x^{k+1} = x^k - \gamma \bar{g}^k := x^k - \gamma \frac{1}{n} \sum_{i=1}^n \bar{g}_i^k,$$

where \bar{g}_i^k is the average of the stochastic gradients received from worker i

$$\bar{g}_i^k = \frac{1}{b_i^k} \sum_{j=1}^{b_i^k} \nabla f_i(x^k; \xi_i^{k,j}).$$

Because stochastic gradients are collected asynchronously, but the update is performed only after all required gradients are received, the algorithm can be regarded as semi-asynchronous—asynchronous in gradient collection but synchronous in parameter updates.

Stopping Condition Rationale. The left-hand side of condition (3.3) appears in the variance of the gradient estimator \bar{g}^k . Ensuring that this quantity is sufficiently large allows the algorithm to control the variance at each step. Moreover, condition (3.3) guarantees that every worker computes at least one gradient, which in turn yields a smaller variance than that of the estimator in Naive Minibatch SGD.

Theoretical Performance. This asynchronous gradient collection strategy achieves the optimal time complexity

$$\mathcal{O}\left(\frac{L_f \Delta}{\varepsilon} \left(\tau_n + \tau_{\text{avg}} \frac{\sigma^2}{n\varepsilon}\right)\right),$$

as shown by [Tyurin & Richtárik \(2024\)](#). The key improvement over Naive Minibatch SGD is that the variance term σ^2 is now multiplied by τ_{avg} instead of τ_n . Since $\tau_{\text{avg}} \ll \tau_n$ in computationally heterogeneous environments, Malenia SGD can potentially provide substantial speedup in highly heterogeneous regimes.

The algorithm's benefits are most pronounced in the high-noise settings (where $\sigma^2/n\varepsilon$ is large). In low-noise scenarios or when $\sigma = 0$, Malenia SGD offers no advantage over Naive Minibatch SGD, since collecting multiple gradients per worker provides no benefit in terms of variance reduction.

Limitations. The main limitation of Malenia SGD is its synchronous nature. After collecting the necessary gradients, the server must broadcast the updated model to all workers simultaneously. This all-at-once synchronization creates substantial communication overhead, which can become a major scalability bottleneck in large-scale or bandwidth-limited environments.

Moreover, the synchronization process forces the server to discard ongoing gradient computations from workers who are actively computing when the broadcast occurs. Since workers operate continuously during the gradient collection phase, they must abandon their current computations upon receiving the new model, wasting valuable computational resources that could otherwise contribute to convergence.

Additionally, Malenia SGD requires prior knowledge of the noise level σ and the target accuracy ε to evaluate the stopping condition (3.3). This dependence on problem-specific parameters, which are often unknown or difficult to estimate in practice, significantly limits its practical applicability.

These synchronization bottlenecks motivate the development of asynchronous optimal methods. Beyond avoiding coordination overhead, asynchronous approaches enable immediate model updates upon gradient arrival, which can accelerate convergence through more frequent parameter updates. This immediate processing is particularly beneficial for sparse models where gradients typically affect disjoint parameter subsets, allowing parallel updates without interference (Recht et al., 2011).

3.4.3 Toward Asynchronous Methods

A naive approach to making optimization asynchronous would be to update the model immediately upon receiving any gradient, using

$$x^{k+1} = x^k - \gamma \nabla f_{i^k} \left(x^{k-\delta^k}; \xi_{i^k}^{k-\delta^k} \right),$$

where i^k denotes the worker that sent the gradient at iteration k , and $\delta^k \geq 0$ is its delay, i.e., the number of server updates that occurred while the gradient was being computed. Delays arise naturally in asynchronous execution: fast workers return gradients quickly and proceed with updated models, while slower workers compute on stale iterates; by the time they return their gradients, several server updates may already have occurred. Consequently, δ^k is only determined when the server receives a gradient: at that point, it knows both the model iterate used by the worker and the current server iterate, and δ^k is simply the difference between the two.

Limitations. This approach suffers from a fundamental bias problem when workers have heterogeneous data distributions. Faster workers send updates more frequently, causing their local objectives to dominate the optimization and pull the model toward their own minimizers. Slower workers, when their updates finally arrive, push the model back toward different solutions. As a result, the iterates may oscillate or stagnate, failing to converge to a stationary point.

This bias also makes theoretical analysis difficult. Classical SGD-style proofs rely on one-step progress toward minimizing the global function, but here each update direction reflects a different local objective. Without additional data similarity assumptions (Mishchenko et al., 2022a; Koloskova et al., 2022; Nguyen et al., 2022; Islamov et al., 2024), it becomes impossible to extend the analysis to the global function—yet such assumptions are rarely realistic when data can be arbitrarily heterogeneous across machines or organizations.

The root cause is that each update uses a gradient from one worker only. A better strategy is to incorporate information from all workers, even if some gradients are computed at stale iterates. This idea motivates methods such as Incremental Aggregated Gradient (IAG) (Blatt et al., 2007; Gurbuzbalaban et al., 2017; Vanli et al., 2018) and Stochastic Averaged Gradient (SAG) (Roux et al., 2012; Schmidt et al., 2017), which maintain and aggregate gradients from all workers.

3.4.4 IA²SGD

As discussed above, the key insight is to construct a gradient estimator using information from all workers at each model update. IA²SGD (Wang et al., 2025) achieves this by maintaining a gradient table on the server, similar to SAG or IAG, but with asynchronous table updates.

Algorithm Overview. The server maintains a gradient table $\{g_i\}_{i=1}^n$ that stores the most recent gradient received from each of the n workers. The table is initialized by collecting one gradient from each worker at the starting point x^0 . The server then computes the first model update

$$x^1 = x^0 - \gamma \bar{g} := x^0 - \gamma \frac{1}{n} \sum_{i=1}^n g_i$$

and broadcasts x^1 to all workers. Subsequently, the workers compute stochastic gradients in parallel, and their corresponding table entries are updated asynchronously as soon as the computations finish.

At each iteration k , the server performs the following steps:

1. Receive gradient $\nabla f_{i^k}(x^{k-\delta_{i^k}^k}; \xi_{i^k}^{k-\delta_{i^k}^k})$ from worker i^k
2. Update the gradient table entry: $g_{i^k} = \nabla f_{i^k}(x^{k-\delta_{i^k}^k}; \xi_{i^k}^{k-\delta_{i^k}^k})$
3. Perform model update: $x^{k+1} = x^k - \gamma \bar{g} = x^k - \gamma \frac{1}{n} \sum_{i=1}^n g_i$
4. Send the updated iterate x^{k+1} to worker i^k for its next computation

The gradient estimator \bar{g} combines the most recent gradient from each worker, ensuring that every update reflects information from the entire set of workers despite asynchronous execution. In this way, the method retains the statistical benefits of using all workers' data while allowing them to operate independently, thereby avoiding the synchronization bottlenecks that limit scalability.

Note that, due to asynchrony, the stochastic gradients stored in the table generally correspond to different iterates of the model. We therefore record each worker i 's delay δ_i^k to track the iterate at which its gradient was computed.

Theoretical Performance. The iteration complexity of this algorithm was established by Wang et al. (2025), and by a straightforward conversion to the fixed computation time model (see Section B.3), we obtain the corresponding time complexity

$$\mathcal{O}\left(\frac{\tau_n L_{\max} \Delta}{\varepsilon} \left(1 + \frac{\sigma^2}{n\varepsilon}\right)\right),$$

which matches the complexity of Naive Minibatch SGD. This indicates that asynchronous execution alone does not provide computational advantages over synchronous approaches. Thus, a fundamental challenge lies in how gradient delay affects convergence, which we address next.

3.4.5 Motivation

The primary limitation of asynchronous algorithms stems from gradient delay, which can significantly degrade convergence performance. Large delays can cause the optimization steps to follow suboptimal trajectories, which disrupts convergence.

This delay problem occurs even in homogeneous settings where all functions f_i are equal ($f_i \equiv f$ for all $i \in [n]$). The state-of-the-art solution for this case, **Ringmaster ASGD** (Maranjyan et al., 2025d), achieves optimal time complexity by explicitly controlling delay to prevent it from becoming large. **Ringmaster ASGD** accomplishes this by discarding gradients that arrive with large delays.

Unfortunately, this gradient-discarding strategy fails in the heterogeneous setting of **IA²SGD**. The fundamental issue is that slow workers inevitably experience large delays due to their computational constraints. If we ignore or discard their delayed gradients, the corresponding table entries remain outdated and may never be updated again if subsequent gradients also arrive late and are discarded. This creates a persistent information bottleneck that degrades the quality of the gradient estimator and harms convergence.

This issue suggests we should prevent such situations from occurring by controlling the number of updates performed using fast workers. The simplest approach would be to ignore some updates from fast workers, but this contradicts the core principle of asynchronous methods whereby all workers compute gradients continuously.

Instead, our approach *buffers* the gradients produced by fast workers rather than applying them immediately, similar to the strategy in **Malenia SGD**. By buffering gradients and performing a model update only once a sufficient number have been collected, we control the delays induced by slow workers while keeping all workers continuously active. This buffering mechanism provides an additional advantage: when multiple gradients computed at the same iterate are aggregated, they yield lower-variance estimates, thereby improving convergence.

3.5 Ringleader ASGD

We now formally introduce our algorithm, **Ringleader ASGD**.

Ringleader ASGD builds upon three key insights from existing methods. First, inspired by **IA²SGD**, we maintain a gradient table⁴ to ensure that information from all workers is incorporated into every update, which eliminates the need for data similarity assumptions between workers. Second, following **Ringmaster ASGD**, we recognize that controlling gradient delay is essential for efficient asynchronous optimization. Third, drawing from **Malenia SGD**, we use buffering of stochastic gradients—rather than discarding delayed ones—to control delays while preserving valuable computations, enabling continuous utilization of all resources.

An important property of the algorithm is that all workers remain continuously active, computing stochastic gradients. As soon as a worker finishes computing a gradient, it immediately sends it to the server. Recall that we assumed

⁴It is not strictly necessary to maintain a table on the server. An alternative, as in **IA²SGD** (Wang et al., 2025), is to store one vector on each worker along with an additional vector on the server. However, this can be problematic when workers have limited memory. For clarity and simplicity, we adopt the server-side table formulation in our description.

communication is instantaneous, i.e., takes zero time (Section 3.3.1). When the server receives a gradient, it either buffers it for later use or applies it immediately to perform a model update. If the gradient is buffered, no further action is taken and the worker simply continues computing and sending new gradients. If the server decides to perform an update, it updates the model and sends the updated model back to the worker that provided the gradient. This server-to-worker communication is also assumed instantaneous, after which the worker resumes computing gradients at the new model point, ensuring that workers are never idle.

The algorithm proceeds in rounds. In each round, exactly n model updates are performed—one for each worker. Specifically, when a worker sends a stochastic gradient, the server may apply an update and return the updated model to that worker, but it ensures that each worker receives an updated model at most once per round. Repeating this procedure n times ensures that every worker obtains exactly one fresh model per round, which in turn keeps delays bounded. To avoid discarding the computations of fast workers, the server buffers their gradients and applies them only at the appropriate moment, thereby guaranteeing that each round consists of exactly n updates.

Each round consists of two phases:

- **Phase 1:** Buffer stochastic gradients in a table until at least one gradient from each worker is available.
- **Phase 2:** Perform exactly n updates (one for each worker), then discard the old stochastic gradients from the table and return to Phase 1 to start collecting fresh ones.

The complete algorithm is formalized in Algorithm 3.5.1.

Algorithm 3.5.1 Ringleader ASGD (server algorithm)

```

1: Input: Step size  $\gamma > 0$ , initial point  $x^0 \in \mathbb{R}^d$ 
2: Initialization: Broadcast  $x^0$  to all workers, which then start running Algorithm 3.5.2 in parallel
3: Set  $k = 0$ ,  $S = \emptyset$ ; initialize  $G_i = 0$ ,  $b_i = 0$  for all  $i \in [n]$ 
4: while not terminated do
5:   — Phase 1: await stochastic gradients from all workers —
6:   while  $S \neq [n]$  do
7:     Receive stochastic gradient  $g_j^k$  (at  $x^{k-\delta_j^k}$ ) from some worker  $j \in [n]$ 
8:      $G_j = G_j + g_j^k$ ;  $b_j = b_j + 1$ ;  $S = S \cup \{j\}$ 
9:   end while
10:  — Phase 2: perform exactly one update for every worker —
11:  Update the model:  $x^{k+1} = x^k - \gamma \frac{1}{n} \sum_{i=1}^n G_i / b_i$ 
12:  Broadcast  $x^{k+1}$  to worker  $j$  ◊ Last worker to complete Phase 1
13:   $k = k + 1$ ;  $S = S \setminus \{j\}$ 
14:   $g_i^+ = 0$ ,  $b_i^+ = 0$  for all  $i \in [n]$ ;  $S^+ = \emptyset$  ◊ Initialize temporary buffer
15:  while  $S \neq \emptyset$  do
16:    Receive stochastic gradient  $g_j^k$  (at  $x^{k-\delta_j^k}$ ) from some worker  $j \in [n]$ 
17:    if  $j \in S$  then
18:       $G_j = G_j + g_j^k$ ;  $b_j = b_j + 1$ 
19:      Update the model:  $x^{k+1} = x^k - \gamma \frac{1}{n} \sum_{i=1}^n G_i / b_i$ 
20:      Broadcast  $x^{k+1}$  to worker  $j$ 
21:       $k = k + 1$ ;  $S = S \setminus \{j\}$ 
22:    else
23:       $G_j^+ = G_j^+ + g_j^k$ ;  $b_j^+ = b_j^+ + 1$ ;  $S^+ = S^+ \cup \{j\}$  ◊ Buffer
24:    end if
25:  end while
26:   $G_i = G_i^+$ ;  $b_i = b_i^+$  for all  $i \in [n]$ ;  $S = S^+$ 
27: end while

```

Algorithm 3.5.2 Worker i 's subroutine

```

1: Input: Model  $x$ 
2: while not terminated do
3:   Compute  $g_i = \nabla f_i(x; \xi_i)$  using a freshly sampled data point  $\xi_i \sim \mathcal{D}_i$ 
4:   Send  $g_i$  to the server
5: end while

```

3.5.1 Detailed Description

Initialization. The algorithm begins by broadcasting the initial point $x^0 \in \mathbb{R}^d$ to all workers, which then start executing the worker subroutine (Algorithm 3.5.2). Each worker continuously computes stochastic gradients at its current point and sends them to the server until instructed to stop, at which point the server provides a new point to resume from. This design ensures that workers remain fully utilized and never idle.

The server maintains a gradient table with entries $\{(G_i, b_i)\}_{i=1}^n$, all initialized to zero. Here, G_i accumulates gradients, while b_i tracks the number of stochastic

gradients received from worker i to form proper minibatch estimators, with $b_i = 0$ for all $i \in [n]$ at the start.

Before Phase 1 begins, we also initialize the set $S = \emptyset$, which tracks which table entries contain at least one stochastic gradient. Since no gradients have yet arrived, S is empty.

Phase 1 — Gradient Collection. In this phase, the server continuously receives stochastic gradients from the workers and stores them in the gradient table $\{(G_i, b_i)\}_{i=1}^n$. We denote by g_j^k the stochastic gradient sent by worker j at iteration k , which is computed at point $x^{k-\delta_j^k}$ using an i.i.d. sample $\xi_j \sim D_j$.

We do not specify how the delays δ_j^k evolve, since this information is not needed to run the algorithm: whenever necessary, a delay can be obtained as the difference between the current model iterate and the iterate at which the stochastic gradient was computed. The delays will only play a role in the analysis, not in the execution of the method.

Upon receiving g_j^k from worker j (Line 7), the server updates the corresponding table entry and the stochastic gradient counter as follows (Line 8)

$$G_j = G_j + g_j^k, \quad b_j = b_j + 1, \quad S = S \cup \{j\}.$$

This process continues until $S = [n]$, i.e., the server has collected at least one gradient from every worker. No model updates are performed during this phase, and workers do not receive new points; hence, all stochastic gradients from a given worker are computed at the same point.

Phase 2 — Sequential Updates. In this phase, the server performs exactly one model update for each worker i , for a total of n updates. Phase 2 starts with the last worker that completed Phase 1, i.e., the worker whose gradient made the table complete. The server first computes an update by averaging the accumulated stochastic gradients in the table $\{(G_i, b_i)\}_{i=1}^n$ and taking a descent step with this estimate (Line 11). The updated model is then sent to this worker (Line 12), the worker is removed from the set S , and the iteration counter is incremented (Line 13).

Next, the server must update the remaining $n - 1$ workers. These updates are performed sequentially as soon as each worker finishes its current computation. During this waiting period, new gradients may arrive from workers not in S —e.g. for example, the last updated worker may send another stochastic gradient before the other workers complete their computation. Since discarding these gradients would waste information, they are instead buffered for the next round.

Temporary Table Management. To achieve this, the server maintains a temporary table $\{(G_i^+, b_i^+)\}_{i=1}^n$, initialized to zero (Line 14), together with a set S^+ that records which workers have contributed to the table. Whenever a gradient arrives from a worker not in S , it is stored in the temporary table (Line 23).

If instead the gradient comes from a worker $j \in S$ —i.e., one of the workers whose model we still need to update—the server first updates the main table $\{(G_i, b_i)\}_{i=1}^n$ with this new stochastic gradient (Line 18). It then performs a model update by again averaging the accumulated stochastic gradients in the

table (Line 19), broadcasts the new model to worker j (Line 20), increments the iteration counter, and removes j from the set $S = S \setminus \{j\}$ (Line 21).

Preparing for the Next Round. Once all workers in S have been updated and Phase 2 is complete ($S = \emptyset$), the server prepares for the next round by copying the contents of the temporary table $\{(G_i^+, b_i^+)\}_{i=1}^n$ into the main table $\{(G_i, b_i)\}_{i=1}^n$ (Line 26). The set S is also reset to $S = S^+$, since these workers already contributed gradients at their updated models. Entries in the main table corresponding to workers not in S^+ remain zero, as the temporary table was initialized with zeros at the start of Phase 2 (Line 14).

The server can now proceed to the next round by returning to Phase 1 and beginning a new gradient collection phase.

3.5.2 Delay Control Analysis

The structure of Ringleader ASGD, with its two phases, is specifically designed to prevent the unbounded delays that arise in standard asynchronous methods. To understand why this works, consider that in each round we perform exactly n updates—one per worker—before moving to the next round. This ensures that no worker can fall more than one full round behind the current iteration. The precise bound on delays is given in the following lemma

Lemma 3.5.1 (Bounded Delay). In Ringleader ASGD (Algorithm 3.5.1), the delays δ_i^k satisfy

$$\delta_i^k \leq 2n - 2 ,$$

for any worker $i \in [n]$ and any iteration $k \geq 0$.

Proof. We prove this by analyzing the structure of Ringleader ASGD. The algorithm operates in rounds, where each round consists of Phase 1 (gradient collection) followed by Phase 2 (sequential updates). In each Phase 2, the server performs exactly n updates, one for each worker. Phase 2 begins at iterations $0, n, 2n, 3n, \dots$, i.e., at multiples of n .

First round (iterations 0 to $n - 1$): Initially, all workers compute gradients at the point x^0 , so during iterations $0, 1, \dots, n - 1$, the server receives gradients computed at x^0 . For any iteration k in this range, the server processes stochastic gradients computed at point $x^{k-\delta_i^k}$, so $\delta_i^k = k \leq n - 1$. Thus, delays simply increment during this first Phase 2.

At the end of this round, each worker i has received a new model x^j for some $j \in \{1, 2, \dots, n\}$, and these update iterations are distinct across workers.

Second round (iterations n to $2n - 1$): At the start of the second Phase 2 (at iteration n), the gradient table contains gradients computed at points $x^{n-\delta_i^n}$ for worker i , where $\delta_i^n \in \{0, 1, \dots, n - 1\}$. These delay values are distinct across workers since each worker received its update at a different iteration in the previous round.

During iterations n to $2n - 1$, these delays increase by 1 at each iteration for the same reason as in the first Phase 2, giving $\delta_i^{2n-1} \in \{n - 1, n, \dots, 2n - 2\}$ at the

end of this round. At the same time, all workers receive new points to compute gradients from, so during the next Phase 2, the delays will again be distinct for all workers and in $\{0, 1, \dots, n - 1\}$.

General pattern: By induction, at the beginning of each round starting at iteration cn (for integer $c \geq 1$), the delays δ_i^{cn} take distinct values in $\{0, 1, \dots, n - 1\}$. During each Phase 2, these delays increase by at most $n - 1$, giving the bound

$$\delta_i^k \leq (n - 1) + (n - 1) = 2n - 2 .$$

□

3.5.3 Comparison to IA²SGD

Our method is a delay-controlled version of IA²SGD. We can recover IA²SGD by removing Phase 1 (gradient collection) and Phase 2 (structured updates), and thus perform updates naively—immediately upon gradient arrival. In contrast, our algorithm operates in structured rounds, performing exactly one update per worker in each round, which provides the crucial delay control that IA²SGD lacks.

In IA²SGD, delays for slow workers can grow unboundedly because the server continuously updates the model using gradients from fast workers, causing slow workers to fall increasingly behind. Our method prevents this issue by buffering the gradients from fast workers rather than immediately applying these gradients, to ensure that all workers receive updated models within n subsequent iterations.

3.5.4 Comparison to Malenia SGD

Malenia SGD also operates as an algorithm with two phases. In Phase 1, Malenia SGD collects gradients using a similar method to our approach, but uses a different termination condition (3.3) that requires knowledge of the noise parameter σ and the target stationarity level ε , making it impractical. In Phase 2, Malenia SGD performs a *synchronous* update by averaging all collected gradients and broadcasting the new model to *all* workers simultaneously before returning to Phase 1. This synchronization forces Malenia SGD to discard ongoing gradient computations from workers that are active during the broadcast.

In contrast, our method performs Phase 2 *asynchronously*: we update workers sequentially as they complete their current gradient computations, which ensures that no computational work is wasted.

Regarding Malenia SGD’s termination condition (3.3), in Section B.4 we demonstrate that this condition can be replaced with our simpler requirement of obtaining at least one gradient from every worker. With this modification, Malenia SGD remains optimal in the fixed-time regime (3.2) while becoming parameter-free, which eliminates the need for prior knowledge of σ and ε . In this parameter-free variant, the only difference between Malenia SGD and Ringleader ASGD lies in Phase 2: we perform updates asynchronously without discarding gradients, while Malenia SGD operates synchronously.

3.6 Theoretical Results

Before presenting the theoretical results, we first write the algorithm in a compact form. The gradients for each worker in the table are all computed at the same point; for worker i at iteration k , the point is $x^{k-\delta_i^k}$. The update rule can be written compactly as

$$x^{k+1} = x^k - \gamma \bar{g}^k ,$$

where the gradient estimator \bar{g}^k is defined by

$$\bar{g}^k := \frac{1}{n} \sum_{i=1}^n \bar{g}_i^k := \frac{1}{n} \sum_{i=1}^n \frac{1}{b_i^k} \sum_{j=1}^{b_i^k} g_i^{k,j} .$$

Since multiple gradients may be received from the same worker, we denote by $g_i^{k,j}$ the j -th gradient from worker i at iteration k . Here the index j corresponds to the i.i.d. sampled data point, and more concretely

$$g_i^{k,j} := \nabla f_i \left(x^{k-\delta_i^k}; \xi_i^{k-\delta_i^k, j} \right) .$$

The quantity b_i^k denotes the number of gradients from worker i stored in the table at iteration k , i.e., the value of b_i in Lines 11 and 19. Thus, the pair (b_i^k, δ_i^k) fully determines the method's behavior at iteration k .

Note that the sequence $\{b_i^k\}$ depends only on the computation times $\{\tau_i\}$ and the algorithm design (i.e., the stopping rule for collecting gradients). Once these are fixed, all b_i^k for every $i \in [n]$ and iteration k are determined. Crucially, the values of b_i^k do not depend on the method's hyperparameters γ , x^0 , or on the variance parameter σ or the stationarity level ε .

3.6.1 Iteration Complexity

Our convergence analysis follows the structured approach employed by Maranjyan et al. (2025d), which decomposes the proof into two key components: a descent lemma that tracks the progress of the objective function and a residual estimation lemma that controls the accumulated delays in the system.

We begin by establishing notation for the harmonic means of the batch sizes across rounds:

$$B^k = \left(\frac{1}{n} \sum_{i=1}^n \frac{1}{b_i^k} \right)^{-1} , \quad \text{and} \quad B = \inf_{k \geq 0} B^k .$$

Note that $B \geq 1$, since by the algorithm's design each $b_i^k \geq 1$. A sharper bound on B will be established later in Lemma 3.6.4.

The first lemma quantifies how much the objective function decreases at each iteration, accounting for both the standard gradient descent progress and the additional complexities introduced by asynchronous updates.

Lemma 3.6.1 (Descent Lemma; Proof in Section B.2.3). Under Assumptions 3.3.1 and 3.3.2, if the stepsize in Algorithm 3.5.1 satisfies $\gamma \leq 1/4L$, then the following

inequality holds

$$\begin{aligned} \mathbb{E}[f(x^{k+1})] &\leq \mathbb{E}[f(x^k)] - \frac{\gamma}{2}\mathbb{E}\left[\|\nabla f(x^k)\|^2\right] - \frac{\gamma}{4}\mathbb{E}\left[\left\|\frac{1}{n}\sum_{i=1}^n \nabla f_i(x^{k-\delta_i^k})\right\|^2\right] \\ &\quad + \frac{\gamma L^2}{2n}\sum_{i=1}^n \mathbb{E}\left[\|x^k - x^{k-\delta_i^k}\|^2\right] + \frac{3\gamma^2 L\sigma^2}{2B} \\ &\quad + \gamma^2 L \sum_{\ell=k-(k \bmod n)}^{k-1} \mathbb{E}\left[\left\|\frac{1}{n}\sum_{i=1}^n \nabla f_i(x^{\ell-\delta_i^\ell})\right\|^2\right]. \end{aligned}$$

This descent lemma shares a similar structure with its counterpart in the homogeneous setting analyzed by Maranjyan et al. (2025d), but with a crucial additional term. The final summation term in the upper bound captures the effect of using stale gradients from the gradient table—a phenomenon we refer to as “table delay”. This term is absent in the homogeneous case because no gradient table is maintained. Indeed, when $n = 1$, our setting reduces to the homogeneous case, the gradient table becomes unnecessary, and this additional term vanishes, recovering the original descent lemma established by Maranjyan et al. (2025d).

Next, similar to the work of Maranjyan et al. (2025d), we derive a lemma to bound the term involving the difference between current and old points.

Lemma 3.6.2 (Residual Estimation; Proof in Section B.2.4). Under Assumption 3.3.1, the iterates of Ringleader ASGD (Algorithm 3.5.1) with stepsize $\gamma \leq 1/(4nL)$ satisfy the following bound

$$\frac{1}{K}\sum_{k=0}^{K-1} \frac{1}{n}\sum_{i=1}^n \mathbb{E}\left[\|x^k - x^{k-\delta_i^k}\|^2\right] \leq \frac{2\gamma n}{LK}\sum_{k=0}^{K-1} \mathbb{E}\left[\left\|\frac{1}{n}\sum_{j=1}^n \nabla f_j(x^{k-\delta_j^k})\right\|^2\right] + \frac{2\gamma\sigma^2}{LB}.$$

Finally, we get the iteration complexity combining these two lemmas.

Theorem 3.6.3 (Iteration Complexity). Under Assumptions 3.3.1, 3.3.2, and 3.3.5, let the stepsize in Ringleader ASGD (Algorithm 3.5.1) be

$$\gamma = \min\left\{\frac{1}{8nL}, \frac{\varepsilon B}{10L\sigma^2}\right\}.$$

Then,

$$\frac{1}{K}\sum_{k=0}^{K-1} \mathbb{E}\left[\|\nabla f(x^k)\|^2\right] \leq \varepsilon,$$

for

$$K \geq \frac{32nL\Delta}{\varepsilon} + \frac{40L\Delta\sigma^2}{B\varepsilon^2} = \mathcal{O}\left(\frac{nL\Delta}{\varepsilon}\left(1 + \frac{\sigma^2}{Bn\varepsilon}\right)\right).$$

Proof. We start by averaging the inequality from Lemma 3.6.1 over K iterations

and dividing both sides by $\gamma/2$

$$\begin{aligned}
& \frac{1}{K} \sum_{k=0}^{K-1} \mathbb{E} \left[\|\nabla f(x^k)\|^2 \right] + \frac{1}{2K} \sum_{k=0}^{K-1} \mathbb{E} \left[\left\| \frac{1}{n} \sum_{i=1}^n \nabla f_i(x^{k-\delta_i^k}) \right\|^2 \right] \\
& \leq \frac{2\Delta}{\gamma K} + \frac{3\gamma L\sigma^2}{B} \\
& + \frac{L^2}{n} \frac{1}{K} \sum_{k=0}^{K-1} \sum_{i=1}^n \mathbb{E} \left[\|x^k - x^{k-\delta_i^k}\|^2 \right] \\
& + 2\gamma L \frac{1}{K} \sum_{k=0}^{K-1} \sum_{\ell=k-(k \bmod n)}^{k-1} \mathbb{E} \left[\left\| \frac{1}{n} \sum_{i=1}^n \nabla f_i(x^{\ell-\delta_i^\ell}) \right\|^2 \right].
\end{aligned}$$

We now bound the third term on the right using Lemma 3.6.2

$$\begin{aligned}
& \frac{1}{K} \sum_{k=0}^{K-1} \mathbb{E} \left[\|\nabla f(x^k)\|^2 \right] + \frac{1}{2K} \sum_{k=0}^{K-1} \mathbb{E} \left[\left\| \frac{1}{n} \sum_{i=1}^n \nabla f_i(x^{k-\delta_i^k}) \right\|^2 \right] \\
& \leq \frac{2\Delta}{\gamma K} + \frac{3\gamma L\sigma^2}{B} + \frac{2\gamma L\sigma^2}{B} \\
& + 2\gamma L n \frac{1}{K} \sum_{k=0}^{K-1} \mathbb{E} \left[\left\| \frac{1}{n} \sum_{j=1}^n \nabla f_j(x^{k-\delta_j^k}) \right\|^2 \right] \\
& + 2\gamma L \frac{1}{K} \sum_{k=0}^{K-1} \sum_{\ell=k-(k \bmod n)}^{k-1} \mathbb{E} \left[\left\| \frac{1}{n} \sum_{i=1}^n \nabla f_i(x^{\ell-\delta_i^\ell}) \right\|^2 \right] \\
& \leq \frac{2\Delta}{\gamma K} + \frac{5\gamma L\sigma^2}{B} \\
& + 2\gamma L n \frac{1}{K} \sum_{k=0}^{K-1} \mathbb{E} \left[\left\| \frac{1}{n} \sum_{j=1}^n \nabla f_j(x^{k-\delta_j^k}) \right\|^2 \right] \\
& + 2\gamma L n \frac{1}{K} \sum_{k=0}^{K-1} \mathbb{E} \left[\left\| \frac{1}{n} \sum_{i=1}^n \nabla f_i(x^{k-\delta_i^k}) \right\|^2 \right].
\end{aligned}$$

Now, using the bound $\gamma \leq 1/(8nL)$, we obtain

$$\frac{1}{K} \sum_{k=0}^{K-1} \mathbb{E} \left[\|\nabla f(x^k)\|^2 \right] \leq \frac{2\Delta}{\gamma K} + \frac{5\gamma L\sigma^2}{B}.$$

Finally, plugging in the stepsize and the expression for K ensures the right-hand side is bounded by ε . \square

For parallel and asynchronous methods, iteration complexity is less important than time complexity. What truly matters is how quickly we can finish training. We are willing to perform more iterations and extra computation if it means completing the process faster. Having established the iteration complexity, we

now turn to the time complexity.

3.6.2 Time Complexity

Since the algorithm operates in rounds with n steps per round, and its iteration complexity is already known, it remains to determine the duration of each round. We have the following lemma

Lemma 3.6.4. Each block of n consecutive iterations (each round) of Algorithm 3.5.1 takes at most $2\tau_n$ seconds. Moreover, we have

$$B \geq \frac{\tau_n}{2} \left(\frac{1}{n} \sum_{i=1}^n \tau_i \right)^{-1} = \frac{\tau_n}{2\tau_{\text{avg}}} .$$

Proof. The upper bound of $2\tau_n$ follows from the structure of the algorithm, which consists of two phases. In the first phase, the server waits until all workers complete at least one gradient computation, which takes at most τ_n seconds. In the second phase, the server applies the received gradients and waits for all ongoing computations to finish—which again takes at most τ_n seconds. Thus, the total time for n iterations is bounded by $2\tau_n$.

We now prove the second part of the lemma. Recall that

$$B = \inf_{k \geq 0} B^k = \inf_{k \geq 0} \left(\frac{1}{n} \sum_{i=1}^n \frac{1}{b_i^k} \right)^{-1} \geq \left(\frac{1}{n} \sum_{i=1}^n \frac{1}{b_i} \right)^{-1} ,$$

where we define

$$b_i = \inf_{k \geq 0} b_i^k .$$

We are interested in the number of gradients stored in the table at iteration k . This count includes gradients computed during Phase 1 plus one additional gradient from Phase 2 (except for the worker that finished Phase 1 last).

Since every worker needs to compute at least one gradient during Phase 1, the slowest worker will take τ_n seconds to complete single gradient computation. During this τ_n -second interval, faster workers $i < n$ may still be finishing gradients from the previous round’s Phase 2 before starting their Phase 1 computations for the current round.

After completing any remaining Phase 2 work (which takes at most τ_i seconds), worker i has at least $\tau_n - \tau_i$ seconds remaining to compute additional gradients for the current round’s Phase 1. The number of gradients that worker i can compute satisfies

$$b_i \geq \max \left\{ 1, \left\lceil \frac{\tau_n - \tau_i}{\tau_i} \right\rceil \right\} \geq \max \left\{ 1, \frac{\tau_n}{\tau_i} - 1 \right\} .$$

For workers i where $\tau_n \geq 2\tau_i$, we have

$$\frac{\tau_n}{\tau_i} - 1 \geq \frac{\tau_n}{2\tau_i} ,$$

and hence

$$b_i \geq \frac{\tau_n}{2\tau_i}.$$

Plugging this bound into the expression for B gives the claimed result. \square

Based on this lemma, we derive the time complexity guarantee of our algorithm

Theorem 3.6.5. Let Assumptions 3.3.2, 3.3.5, and 3.3.1 hold. Let the stepsize in Ringleader ASGD (Algorithm 3.5.1) be $\gamma = \min\{1/(8nL), \varepsilon B/(10L\sigma^2)\}$. Then, under the *fixed computation model* (3.2), Ringleader ASGD achieves the optimal time complexity

$$\mathcal{O}\left(\frac{L\Delta}{\varepsilon}\left(\tau_n + \tau_{\text{avg}}\frac{\sigma^2}{n\varepsilon}\right)\right).$$

Proof. We start with the iteration complexity from Theorem 3.6.3

$$K \geq \frac{32nL\Delta}{\varepsilon} + \frac{40L\Delta\sigma^2}{B\varepsilon^2} = \mathcal{O}\left(\frac{nL\Delta}{\varepsilon}\left(1 + \frac{\sigma^2}{Bn\varepsilon}\right)\right).$$

The time to do n steps is at most $2\tau_n$ from Lemma 3.6.4. Then the time complexity is

$$2\tau_n \times \frac{K}{n} = \mathcal{O}\left(\tau_n \frac{L\Delta}{\varepsilon}\left(1 + \frac{\sigma^2}{Bn\varepsilon}\right)\right).$$

It remains to put $B \geq \tau_n/2\tau_{\text{avg}}$ from Lemma 3.6.4. \square

The obtained time complexity consists of two key terms that illuminate the algorithm's behavior. The first term depends on the slowest device, which is fundamental since all devices must contribute to solving the problem. The second term, however, involves τ_{avg} rather than τ_n as in Naive Minibatch SGD (see Table 3.1)—this substitution captures the core benefit of asynchronous execution. Specifically, this advantage becomes pronounced when σ is relatively large. Intuitively, in high-noise regimes, collecting many gradients from workers is essential for convergence, and asynchronous methods can leverage faster workers more effectively. Conversely, in low-noise settings, fewer gradient evaluations suffice for good performance, making Naive Minibatch SGD already quite effective and rendering the additional complexity of asynchrony unnecessary.

This time complexity result matches the lower bound derived by Tyurin & Richtárik (2024), thus making Ringleader ASGD the first asynchronous algorithm to achieve optimality.

3.7 Experiments

To validate our theoretical results we perform a toy simulation.

We consider image classification on MNIST (LeCun et al., 2010) and on Fashion-MNIST (Xiao et al., 2017) with standard normalization for both datasets. To enable equal client sizes, we first trim the dataset so that the total number of examples is divisible by the number of clients $n = 100$. To obtain heterogeneous datasets across clients, we then partition the trimmed set using an *equal-size Dirichlet* procedure with concentration parameter $\alpha = 0.1$ (Yurochkin et al.,

2019). For each client $j \in [n]$, we draw proportions $p_j \sim \text{Dirichlet}(\alpha, \dots, \alpha)$ over the classes and compute a rounded class-allocation vector whose entries sum exactly to N/n , where N is the trimmed dataset size. This creates non-IID data where each client has a skewed distribution over classes (with $\alpha = 0.1$, clients typically observe only 1-2 classes frequently).

When assigning samples, we take the requested number from each class pool for client j . If a class pool does not have enough remaining examples to meet the requested amount, the client receives whatever is left from that class and the shortfall is *topped up* using samples from other classes that still have available examples.

Our model is a two-layer MLP $\text{Linear}(d, 128) \rightarrow \text{ReLU} \rightarrow \text{Linear}(128, 10)$ trained with mean cross-entropy. Stochastic gradients at the clients use a mini-batch of size 4, while reported gradient norms are computed on the *full* dataset.

We emulate heterogeneous compute by assigning each client i a base delay and a random jitter:

$$\tau_i = i + |\eta_i|, \quad \eta_i \sim \mathcal{N}(0, i), \quad \text{for all } i \in [n].$$

For each method we tune the stepsize γ within a fixed wall-clock budget. We sweep

$$\gamma \in \{0.001, 0.005, 0.01, 0.02, 0.05, 0.1, 0.2, 0.5, 1.0, 2.0\},$$

and then fix the best γ per method for evaluation.

We report the full-batch gradient-norm squared $\|\nabla f(x^k)\|^2$, versus wall-clock time. Each method is run 30 times over different seeds. We report the *median* with *interquartile range (IQR)*. To reduce high-frequency noise, we apply a centered moving-average smoothing to the aggregated curves (post-aggregation), while keeping the initial point unchanged.

Figure 3.7.1 shows the results. We observe that Ringleader ASGD converges faster compared to Malenia SGD and IA²SGD. Although both Ringleader ASGD and Malenia SGD have the same theoretical time complexity, in practice Ringleader ASGD performs better because Phase 2 involves n asynchronous updates instead of a single synchronous one. This design enables more optimization steps within the same wall-clock budget, which is especially advantageous when updates are sparse.

3.8 Conclusion

We have introduced Ringleader ASGD, the first asynchronous stochastic gradient method to achieve optimal time complexity under arbitrary data heterogeneity and arbitrarily heterogeneous computation times in distributed learning, without requiring similarity assumptions between workers' datasets.

Its core innovation is a two-phase structure within each round: the model is updated once per worker (for a total of n updates), while a buffering mechanism manages gradient delays and preserves the efficiency of asynchronous execution. By maintaining a gradient table and alternating between gradient collection and sequential updates, Ringleader ASGD prevents the unbounded delays common in naive asynchronous methods. Every gradient received by the server is either used in the current round or stored for future use, ensuring no computation is wasted.

Our analysis shows that Ringleader ASGD matches the optimal time complexity

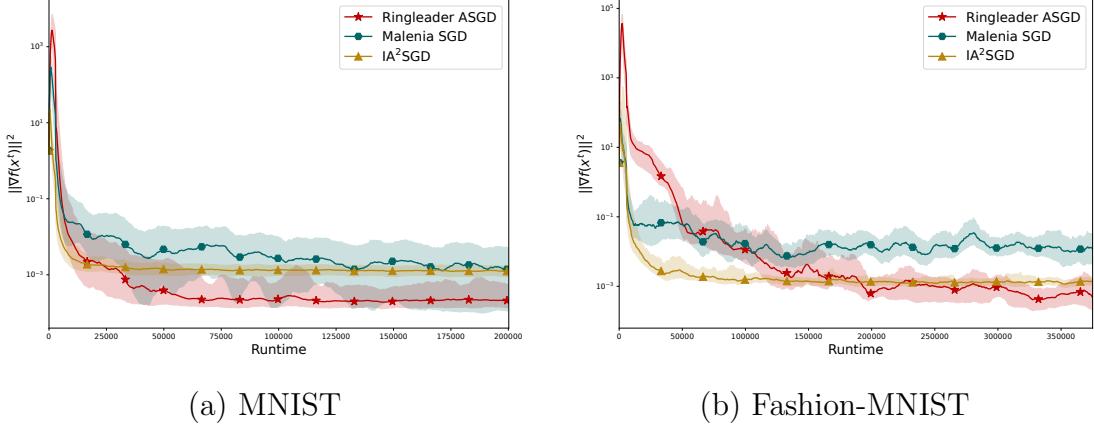


Figure 3.7.1: Convergence comparison showing median squared gradient norm $\|\nabla f(x^k)\|^2$ (solid lines) with interquartile ranges (shaded regions) plotted against wall-clock time, averaged over 30 random seeds. **Setup:** A two-layer MLP with architecture $\text{Linear}(d, 128) \rightarrow \text{ReLU} \rightarrow \text{Linear}(128, 10)$ is trained on (a) MNIST and (b) Fashion-MNIST datasets. **Client delays:** Heterogeneous delays are simulated as $\tau_i = i + |\eta_i|$, where $\eta_i \sim \mathcal{N}(0, i)$ for each client $i \in [n]$, with $n = 100$. **Results:** With optimally tuned stepsizes, Ringleader ASGD converges faster than both Malenia SGD and IA²SGD, even though Ringleader ASGD and Malenia SGD share the same theoretical time-complexity guarantees.

bounds established by [Tyurin & Richtárik \(2024\)](#). In contrast to the optimal but synchronous Malenia SGD method, Ringleader ASGD is asynchronous and requires no prior knowledge of problem parameters in the algorithm design, making it practical for real-world deployments.

Finally, with a minor modification, Ringleader ASGD also achieves optimality in the more general setting of arbitrarily varying computation times (Section B.1).

Chapter 4

ATA: Adaptive Task Allocation for Efficient Resource Management in Distributed Machine Learning

This chapter is based on the work of Maranjyan et al. (2025b):

Artavazd Maranjyan, El Mehdi Saad, Peter Richtárik, and Francesco Orabona. ATA: Adaptive task allocation for efficient resource management in distributed machine learning. In *International Conference on Machine Learning*, 2025b

4.1 Introduction

In this work, we address a very general yet fundamental and important problem arising in various contexts and fields. In particular, there are n workers/nodes/devices collaborating to run some iterative algorithm which has the following structure:

- In order to perform a single iteration of the algorithm, a certain number (B) of *tasks* needs to be performed.
- Each task can be computed by any worker, and the tasks are not temporally related. That is, they can be computed in any order, in parallel, and so on.
- Whenever a worker is asked to perform a single task, the task will take a certain amount of time, modeled as a nonnegative random variable drawn from an unknown distribution specific to that worker. The stochastic assumption makes sense because in real systems computation times are not fixed and can vary with each iteration (Dean & Barroso, 2013; Chen et al., 2016a; Dutta et al., 2018; Maranjyan et al., 2025a).
- Each worker can only work on a single task at a time. That is, a worker processes all tasks it has to perform sequentially. Different workers work in parallel.

4.1.1 Greedy Task Allocation

A natural goal in this setup is to make sure all tasks are completed as fast as possible (in expectation), which minimizes the (expected) time it takes for a single iteration of the algorithm to be performed provided that the task completion time is the dominant time factor of the iteration. Provided we are willing to waste resources, there is a simple solution to this problem, a Greedy Task Allocation (GTA) strategy, which follows this principle: *Make sure all workers are always busy working on some task, and stop once B tasks have been completed.* In GTA, we initially ask all n workers to start working on a task, and as soon as some

worker is done with a task, we ask it to start completing another task. This process is repeated until B tasks have been completed.

4.1.2 Wastefulness of GTA

While GTA minimizes the completion time, it can be immensely wasteful in terms of the total worker utilization time needed to collect all B tasks. Indeed, consider the scenario with $n = 1,010$ workers and $B = 10$ tasks. In this case, GTA will lead to at least $n - B = 1,000$ unnecessary tasks being run in each iteration! This is highly undesirable in situations where the workers are utilized across multiple other jobs besides running the iterative algorithm mentioned above.

4.1.3 Goal of this work

The goal of our work is to design new task allocation strategies, with rigorous theoretical support, that would attempt to minimize the expected completion time subject to the *constraint* that such wastefulness is completely eliminated. That is, we ensure that no more than B tasks are completed in each round.

4.1.4 A Motivating Example: Optimal Parallel SGD

A key inspiration for our work, and the prime example of the general task collection problem described above, relates to recent development in the area of parallel stochastic gradient descent (SGD) methods. Consider the problem of finding an approximate stationary point of the optimization problem

$$\underset{x \in \mathbb{R}^d}{\text{minimize}} \{f(x) := \mathbb{E}_{\xi \sim \mathcal{D}} [f_\xi(x)]\},$$

where $f_\xi : \mathbb{R}^d \rightarrow \mathbb{R}$ are smooth nonconvex functions, and f is assumed to be bounded from below. We assume that

$$\mathbb{E}_{\xi \sim \mathcal{D}} [\|f_\xi(x) - \nabla f(x)\|^2] \leq \sigma^2$$

for all $x \in \mathbb{R}^d$.

In a recent breakthrough, [Tyurin & Richtárik \(2024\)](#) developed a parallel SGD method, *optimal* in terms of a novel notion of complexity called *time complexity*, for solving the above problem with n parallel workers, assuming that it takes $\tau_i > 0$ seconds to worker i to compute a stochastic gradient of f (this corresponds to a task). Their method, **Rennala SGD**, corresponds to **Minibatch SGD** of minibatch size B (which depends on the target accuracy and σ only), with the B tasks (stochastic gradients) completed via GTA. While minimax optimal in terms of time complexity, the GTA task allocation strategy employed within Rennala SGD can be wasteful, as explained above.

Recently, [Maranjyan et al. \(2025d\)](#) proposed **Ringmaster ASGD**, a fully asynchronous SGD method, matching the optimal time complexity of Rennala SGD and achieving optimality for arbitrary compute time patterns associated with the tasks (stochastic gradients), including random, as considered in our setup. However, Ringmaster ASGD also employs a greedy task allocation strategy, leading to wastefulness.

Numerous other parallel/distributed methods involve the implementation of a task allocation strategy, including stochastic proximal point methods (task = evaluation of the stochastic prox operator), higher-order methods (task = evaluation of stochastic Hessian), and beyond. So, by addressing the general task allocation problem, we aim to tame the inherent resource wastefulness of all these methods.

4.1.5 Contributions

In this work, we formalize the task allocation problem as a *combinatorial online learning problem with partial feedback and non-linear losses*. Then, we introduce ATA, a lower-confidence bound-based algorithm designed to solve the proposed allocation problem. ATA is agnostic to workers’ computation times, and our theoretical analysis demonstrates that the total computation time achieved by our methods remains within a small multiplicative factor of the optimal computation time (i.e., the one attainable with full knowledge of the workers’ arm distributions). Additionally, we present ATA-Empirical, a variant of ATA that leverages a novel data-dependent concentration inequality and achieves better empirical results. Finally, we validate our approach through numerical simulations.

4.2 Related Work

Most of the literature on asynchronous methods focuses on demonstrating advantages over their synchronous counterparts. For the simplest method, SGD, this was only recently established by [Tyurin & Richtárik \(2024\)](#). With this result in place, the community can now shift its focus to reducing the overhead of asynchrony. Our work may be the first step in this direction.

In federated learning (FL) ([Konečný et al., 2016b](#); [McMahan et al., 2016](#); [Kairouz et al., 2021](#)), several works account for system heterogeneity. The most well-known FL method, FedAvg ([McMahan et al., 2017](#)), operates by performing multiple local steps on workers, where each step can be viewed as a task. Some works adjust the number of local steps based on worker computation times ([Li et al., 2020b](#); [Maranjyan et al., 2025c](#)), effectively adapting task assignments to worker speed. However, these methods rely on prior knowledge of these times rather than learning them adaptively, as we do.

We reformulate our problem as an online bandit problem. The literature on bandit algorithms is vast, and we refer the reader to the work by [Lattimore & Szepesvári \(2020\)](#) for an introduction to this subject. Our algorithm is based on the approach of using Lower Confidence Bounds (LCBs) on the true means of the arms. This idea, originally proposed by [Auer \(2002\)](#) for the classical Multi-Armed Bandit (MAB) setting, has since been widely adopted in the stochastic combinatorial bandits literature ([Gai et al., 2012](#); [Chen et al., 2013](#); [Combes et al., 2015](#); [Kveton et al., 2015](#)). Using LCBs instead of the empirical estimates of the means allows an optimal trade-off between exploration and exploitation.

The “greedy” approach we employ, which involves selecting the action that minimizes the loss function based on lower confidence bounds instead of the unknown means, is a standard technique in the literature ([Chen et al., 2013](#); [Lin et al., 2015](#)). However, note that our larger action space and the discontinuity

of our loss function necessitates a more tailored analysis. To the best of our knowledge, this is the first work addressing a non-continuous loss function in a stochastic combinatorial MAB-like framework. To overcome this challenge, we exploit the specific structures of our loss function and action space to control the number of rounds where suboptimal actions are chosen. Additionally, our procedure is computationally efficient.

4.3 Problem Setup

In this section, we formally describe the problem setup.

4.3.1 Task Allocation Protocol

We consider a system of n workers, each responsible for repeatedly performing the same task (e.g., computing a gradient). In each round, the allocation algorithm has a budget of B units, which must be distributed among the n workers. Allocating one unit corresponds to one execution of the task.

Let K denote the total number of rounds, which is assumed to be unknown to the learner. For worker $i \in [n] := 1, 2, \dots, n$, let $X_i^{k,u}$ be the computation time required in round $k \in [K]$ to complete its u -th task. Thus, the total computation time for worker i to perform a_i^k tasks in round k is

$$\sum_{u=1}^{a_i^k} X_i^{k,u}$$

with the convention that this sum equals 0 when $a_i^k = 0$.

At each round k , the algorithm selects an allocation vector $a^k = (a_1^k, \dots, a_n^k) \in \mathbb{Z}_+^n$ such that $\|a^k\|_1 = \sum_{i=1}^n a_i^k = B$, based only on the information available up to round $k - 1$. The feedback consists of a_i^k observed times for all the chosen workers. We will denote the action set by

$$\mathcal{A} := \{a \in \mathbb{Z}_+^n : \|a\|_1 = B\} .$$

The objective of the allocation strategy in each round k is to minimize the total computation time. Formally, we define

$$C : \mathcal{A} \rightarrow \mathbb{R}_+ ,$$

as the computation time the optimizer must wait to receive B completed tasks (e.g., gradients) under an allocation vector $a \in \mathcal{A}$, given by

$$C(a^k) := \max_{i \in \text{supp}(a^k)} \sum_{u=1}^{a_i^k} X_i^{k,u} . \quad (4.1)$$

4.3.2 Modeling Assumptions

We assume that the computation time of each worker $i \in [n]$ are i.i.d. drawn from a random variable X_i following a probability distribution ν_i . We denote by

$\mu = (\mu_1, \dots, \mu_n)$ the vector of unknown means. Hence, the random variables $X_i^{k,u}$ with $u \in \{1, \dots, a_i^k\}$ are a_i^k i.i.d. samples drawn from ν_i .

We assume that the distribution ν_i of the computation times to be sub-exponential random variables. To quantify this assumption, we recall the definition of the sub-exponential norm, also known as the Orlicz norm, for a centered real-valued random variable X :

$$\|X\|_{\psi_1} := \inf \left\{ C > 0 : \mathbb{E} \left[\exp \left(\frac{|X|}{C} \right) \right] \leq 2 \right\}. \quad (4.2)$$

Hence, formally we make the following assumption.

Assumption 4.3.1. For each $i \in [n]$, X_i is a positive random variable with

$$\|X_i - \mu_i\|_{\psi_1} \leq \alpha_i, \quad \alpha_i \geq 0.$$

In the remainder of this paper we denote $\alpha := \max_{i \in [n]} \alpha_i$.

The considered class encompasses several other well-known classes of distributions in the literature, such as support-bounded and sub-Gaussian distributions. Moreover, it includes exponential distributions, which are frequently used in the literature to model waiting or computation times in queuing theory and resource allocation in large distributed systems (Gelenbe & Mitrani, 2010; Gross et al., 2011; Hadjis et al., 2016; Mitliagkas et al., 2016; Dutta et al., 2018; Nguyen et al., 2022).

4.3.3 Objective of the Allocation Algorithm

The main objective of this work is to develop an online allocation strategy that achieves a small expected total computation time, defined as

$$\mathcal{C}_K := \sum_{k=1}^K \mathbb{E} [C(a^k)].$$

If the distributions of the arms were known in advance, the optimal allocation

$$a^* \in \arg \min_{a \in \mathcal{A}} \mathbb{E} [C(a)]$$

would be selected to minimize the expected computation time per round, and the same allocation would be used consistently across all K rounds. The corresponding optimal total computation time is then

$$\mathcal{C}_K^* := K \mathbb{E} [C(a^*)].$$

Our goal is to design a strategy that ensures the computation time \mathcal{C}_K remains within a small multiplicative factor of the optimal time \mathcal{C}_K^* , up to an additional negligible term. Specifically, we aim to satisfy

$$\mathcal{C}_K \leq c \cdot \mathcal{C}_K^* + \mathcal{E}_K, \quad (4.3)$$

where $c \geq 1$ is a constant close to 1, and \mathcal{E}_K is a negligible term compared to \mathcal{C}_K^* when $K \rightarrow \infty$. This would assure us that in the limit we are a constant

multiplicative factor away from the performance of the optimal allocation strategy that has full knowledge of the distributions of the computational times of the workers.

Finding a strategy solving the objective in (4.3) presents several technical challenges. First, the action space \mathcal{A} is discrete, and the nonlinearity of the computation time functions $C(\cdot)$ prevents reducing our objective to a convex problem. Second, the size of \mathcal{A} is combinatorial, growing on the order of $(\frac{n+B-1}{B})$, which necessitates exploiting the inherent problem structure to develop efficient strategies. Third, because the workers' computation times are stochastic, any solution must account for uncertainty. Finally, the online setting forces the learner to balance exploration and exploitation under a limited allocation budget of B units per round and partial feedback—only the computation times of workers who receive allocations are observed. This last point naturally suggests adopting a MAB approach.

In the next section, we show how to reduce this problem to a MAB problem and how to efficiently solve it.

4.4 Adaptive Task Allocation

Here, we first show how to reduce the problem in (4.3) to a *non-linear* stochastic MAB problem. Then, we propose an efficient algorithm for this formulation.

4.4.1 Reduction to Multi-Armed Bandit and Proxy Loss

The stochastic MAB problem is a fundamental framework in sequential decision-making under uncertainty. It involves a scenario where an agent must choose among a set of arms, each associated with an unknown reward distribution. The agent aims to maximize cumulative reward (or equivalently minimize the cumulative loss) over time by balancing exploration (gathering information about the reward distributions) and exploitation (leveraging the best-known arm). The challenge lies in the trade-off between exploring suboptimal arms to refine reward estimates and exploiting the arm with the highest observed reward, given the stochastic nature of the outcomes. Using the terminology from bandit literature, here we will refer to each worker as an “arm”.

However, differently from the standard MAB problem, we have a harder problem because $\mathbb{E}[C(a^k)]$ depends on the joint distribution of all the arms in the support of a^k , rather than on their expectations only. This dependency potentially renders the task of relying on estimates of $\mathbb{E}[C(a)]$ for $a \in \mathcal{A}$ computationally challenging due to the combinatorial nature of the set \mathcal{A} .

To solve this issue, our first idea is to introduce a *proxy loss* $\ell : \mathcal{A} \times \mathbb{R}_+^n \rightarrow \mathbb{R}_+$, defined as

$$\ell(a, \mu) := \max_{i \in [n]} a_i \mu_i . \quad (4.4)$$

Due to the convexity of $C(\cdot)$, the introduced proxy-loss underestimates the expected computation time. However, in Section C.4.3 we prove that this quantity also upper bounds the expected computation time up to a constant that depends on the distribution of the arms. In particular, for any $a \in \mathcal{A}$, we show that

$$\ell(a, \mu) \leq \mathbb{E}[C(a)] \leq (1 + 4\eta \ln(B))\ell(a, \mu) , \quad (4.5)$$

where η is defined as

$$\eta := \max_{i \in [n]} \frac{\alpha_i}{\mu_i}. \quad (4.6)$$

In words, η provides an upper bound on the ratio between the standard deviation and the mean of the arms. Note that in the literature, it is common to consider exponential, Erlang, or Gamma distributions, where the ratio η is typically¹ bounded by 1.

The bound above will allow us to derive guarantees on the total computation time of an allocation strategy based on its guarantees for the proxy loss $\ell(\cdot)$, up to a factor of the order $1 + 4\eta \ln(B)$. We remark that in the special case where the arms' distributions are deterministic ($\eta = 0$) or the query budget is unitary ($B = 1$), the two targets $\mathbb{E}[C(a)]$ and ℓ exactly coincide.

4.4.2 Comparison with the Combinatorial Bandits Setting

Our setting is closely related to the Combinatorial Multi-Armed Bandits (CMAB) framework (Cesa-Bianchi & Lugosi, 2012), particularly due to the combinatorial nature of the action space and the semi-bandit feedback, where the learner observes outcomes from all chosen arms. However, our formulation differs in two significant ways. First, while CMAB typically involves selecting a subset of n arms, resulting in an action space with a maximum size of 2^n , our action space \mathcal{A} has a cardinality of $\binom{n+B-1}{B}$. The ratio between these two can be extremely large, potentially growing exponentially with n . Second, although most works in this domain assume a linear loss function in the arms' means, some notable exceptions address non-linear reward functions (Chen et al., 2013; Lin et al., 2015; Chen et al., 2016b; Wang & Chen, 2018). However, these approaches generally rely on assumptions such as smoothness, Lipschitz continuity, or higher-order differentiability of the reward function. In contrast, our loss function $\ell(\cdot, \mu)$ is not continuous with respect to the arms' means. Finally, motivated by the practical requirements of our setting, we place a strong emphasis on computational efficiency that rules out most of the approaches based on CMAB.

4.4.3 Adaptive Task Allocation Algorithm

Now, we introduce our Adaptive Task Allocation algorithm (ATA). ATA does not require prior knowledge of the horizon K and only relies on an upper bound α satisfying $\alpha \geq \max_{i \in [n]} \|X_i - \mu_i\|_{\psi_1}$ for the Orlicz norms of the arm distributions. Recall that $\|X_i - \mu_i\|_{\psi_1} \leq 2\|X_i\|_{\psi_1}$, so an upper bound on $\|X_i\|_{\psi_1}$ also provides one for $\|X_i - \mu_i\|_{\psi_1}$. The core idea of the procedure is to allocate the workers based on *lower confidence bound estimates* on the arm means $(\mu_i)_{i \in [n]}$, in order to balance exploration and exploitation.

For each arm $i \in [n]$ and round $k \in [K]$, let K_i^k represent the number of samples collected from the distribution of arm i up to round k . At each round k , we compute an empirical mean, denoted by $\hat{\mu}_i^k$, using the K_i^k samples obtained so far. Based on these empirical means, we define the lower confidence bounds

¹For $\text{Gamma}(\alpha, \lambda)$, $\sigma/\mu = 1/\sqrt{\alpha}$, so the claim holds for $\alpha \geq 1$.

Algorithm 4.4.1 ATA (Adaptive Task Allocation)

- 1: **Input:** allocation budget B , $\alpha > 0$
- 2: **Initialize:** empirical means $\hat{\mu}_i^1 = 0$, usage counts $K_i^1 = 0$, and usage times $T_i^1 = 0$, for all $i \in [n]$
- 3: **for** $k = 1, \dots, K$ **do**
- 4: Compute LCBs (s_i^k) for all $i \in [n]$ using (4.7)
- 5: Find allocation:

$$a^k \in \arg \min_{a \in \mathcal{A}} \ell(a, s^k)$$

- 6: Allocate a_i^k tasks to each worker $i \in [n]$

7: **Update optimization parameters**

- 8: For all i such that $a_i^k \neq 0$, update:

$$\begin{aligned} K_i^{k+1} &= K_i^k + a_i^k \\ T_i^{k+1} &= T_i^k + \sum_{j=1}^{a_i^k} X_i^{k,j} \\ \hat{\mu}_i^{k+1} &= \frac{T_i^{k+1}}{K_i^{k+1}} \end{aligned}$$

- 9: **end for**
-

s_i^k as

$$s_i^k = (\hat{\mu}_i^k - \text{conf}(i, k))_+, \quad (4.7)$$

where $(x)_+ = \max\{x, 0\}$ and $\text{conf}(\cdot, \cdot)$ is defined as

$$\text{conf}(i, k) = \begin{cases} 2\alpha \left(\sqrt{\frac{\ln(2k^2)}{K_i^k}} + \frac{\ln(2k^2)}{K_i^k} \right), & K_i^k \geq 1, \\ +\infty, & K_i^k = 0. \end{cases}$$

The term $\text{conf}(\cdot, \cdot)$ is derived from a known concentration inequality for sub-exponential variables with an Orlicz norm bounded by α (Lemma C.5.1 in the Appendix).

Given the confidence bounds $s^k := (s_1^k, \dots, s_n^k)$, the learner selects the action $a^k \in \mathcal{A}$ at round k that minimizes the loss $\ell(\cdot, s^k)$, defined in (4.4). While nonconvex, we show in Section C.3 that this optimization problem can be solved using a recursive routine, whose computational efficiency is

$$\mathcal{O}(n \ln(\min\{B, n\}) + \min\{B, n\}^2) .$$

As last step, the feedback obtained after applying the allocation a^k is used to update the lower confidence bounds. The complete pseudocode for ATA is provided in Algorithm 4.4.1.

Remark 4.4.1. Line 7 of the algorithm acts as a placeholder for the optimization method, where the optimization parameters are updated using the quantities computed by the workers (e.g., gradients in the case of SGD). In this view, the allocation algorithm is independent of the specifics of the chosen optimization

algorithm. Refer to Section C.2 for further details.

4.4.4 Upper-Bound on the Total Computation Time

We provide guarantees for ATA in the form of an upper bound on the expected total computation time required to perform K iterations of the optimization procedure. Recall that the proxy loss $\ell(\cdot, \mu)$ and the expected computation time are related through (4.5). This relationship and Theorem 4.6.1 allow us to derive guarantees on the expected total computation time, denoted by

$$\mathcal{C}_K := \sum_{k=1}^K \mathbb{E}[C(a^k)] .$$

Recall that the optimal expected total computation time in this framework is given by

$$\mathcal{C}_K^* := K \mathbb{E}[C(a^*)] ,$$

with

$$a^* \in \arg \min_{a \in \mathcal{A}} \mathbb{E}[C(a)] .$$

We now state the result that provides an upper bound on the total expected computation time achieved by ATA.

Theorem 4.4.2 (Proof in Section C.4.3). Suppose Assumption 4.3.1 holds and let $\eta := \max_{i \in [n]} \alpha_i / \mu_i$. Then, the total expected computation time after K rounds, using the allocation prescribed by ATA with inputs (B, α) satisfies

$$\mathcal{C}_K \leq (1 + 4\eta \ln(B)) \mathcal{C}_K^* + \mathcal{O}(\ln K) .$$

Remark 4.4.3. The $\mathcal{O}(\cdot)$ term hides an instance dependent factor. We will give its full specifics in the regret upper bound of Theorem 4.6.1.

The bound in Theorem 4.4.2 shows that the total expected computation time of ATA remains within a multiplicative factor of $1 + 4\eta \ln(B)$ of the optimal computation time \mathcal{C}_K^* , with an additional remainder term that scales logarithmically with K . Since $\mathcal{C}_K = \Omega(K)$, this additive term is negligible compared to \mathcal{C}_K^* . In practical scenarios, where computation time follows common distributions such as exponential or Gamma, the factor η is typically of order 1, and $\ln(B)$ remains relatively small for the batch sizes commonly used in optimization algorithms like SGD.

The reader might wonder if the more ambitious goal of deriving bounds with a multiplicative factor of exactly 1 is achievable. However, achieving this goal would require significantly more precise estimates of the expected computation time $\mathbb{E}[C(a)]$ for all $a \in \mathcal{A}$. Since $\mathbb{E}[C(a)]$ depends on the joint distribution of all workers in the support a , obtaining such precise estimates would come at the cost of computational efficiency in the allocation strategy.

We note that it is unsurprising that η appears in the upper bound of Theorem 4.4.2, since having a heavier-tailed distribution increases the gap between $\ell(a, \mu)$ and $\mathbb{E}[C(a)]$ through the convexity of $C(\cdot)$. Instead, the factor $\ln(B)$ arises because $C(\cdot)$ is expressed as the maximum of up to B random variables.

Moreover, in the edge cases where $\eta = 0$ (deterministic case) or $B = 1$ (linear cost function), we guarantee that the expected computation time is at most an *additive* factor away from the optimal one.

4.5 Empirical Adaptive Task Allocation

The ATA procedure is based on a lower confidence bound approach that relies on concentration inequalities. These bounds play a key role in performance, as sharper concentration bounds lead to more accurate estimates and reduce exploration of suboptimal options. Since workers' computation times follow sub-exponential distributions, their concentration behavior is determined by the Orlicz norm of the corresponding variables. In ATA, the only prior knowledge available is an upper bound on the largest Orlicz norm among all arms. When the Orlicz norms of the arms' distributions vary significantly, this uniform bound may result in loose confidence intervals and inefficient exploration.

To address this issue, we introduce ATA-Empirical, which better adapts to the distribution of each arm, particularly its Orlicz norm. This adaptation is achieved through a novel data-dependent concentration inequality for sub-exponential variables. Unlike ATA, which depends on the maximum Orlicz norm, ATA-Empirical accounts for the individual Orlicz norms of all arms, denoted by $(\alpha_i)_{i \in [n]}$. This improvement is reflected in the upper bounds on regret presented in Section 4.6. In practice, this leads to improved performance at least some settings, as shown in our simulations in Section 4.7. However, this increased adaptivity comes with a trade-off since ATA-Empirical requires an upper bound on the quantity $\eta = \max_i \alpha_i / \mu_i$, rather than a bound on the largest Orlicz norm. That said, for many distributions of interest, the ratios α_i / μ_i across different arms tend to be of the same order, whereas their Orlicz norms can vary significantly. The ATA-Empirical procedure differs from ATA only in the lower confidence bounds it uses. These bounds are derived from the novel concentration inequality in Lemma 4.6.2 and are defined for arm $i \in [n]$ at round $k \in [K]$ as

$$\hat{s}_i^k = \hat{\mu}_i^k \left[1 - 2\eta \left(\sqrt{\frac{\ln(2k^2)}{K_i^k}} + \frac{\ln(2k^2)}{K_i^k} \right) \right]_+, \quad (4.8)$$

where $\eta = \max_{i \in [n]} \alpha_i / \mu_i$.

The expected total computation time \mathcal{C}_K of ATA-Empirical satisfies the same guarantee presented in Theorem 4.6.1, but we obtain an improved multiplicative factor of the additive logarithmic term. The precise expressions of these factors are provided in the next section, and they show that the guarantees of ATA-Empirical adapt to the Orlicz norms $\|X_i\|_{\psi_1}$ of each arm, while the guarantees of ATA depend on the maximum Orlicz norm $\max_i \|X_i\|_{\psi_1}$.

4.6 Theoretical Results

In this section, we sketch the derivation of Theorem 4.4.2 for ATA and ATA-Empirical, through a regret analysis on the proxy losses. We define the expected

cumulative regret of the proxy loss $\ell(\cdot, \mu)$ after K rounds

$$\mathcal{R}_K := \sum_{k=1}^K \mathbb{E} [\ell(a^k, \mu)] - K \cdot \ell(\bar{a}, \mu) , \quad (4.9)$$

where $\bar{a} \in \arg \min_{a \in \mathcal{A}} \ell(a, \mu)$ represents the optimal allocation over the workers. If multiple optimal actions exist, we consider the one returned by the optimization sub-routine used in ATA (line 5 of Algorithm C.2.1).

We derive upper bounds on the expected cumulative regret \mathcal{R}_K . Based on these bounds, we provide the guarantees on the expected total computation time required to complete K iterations of the optimization process.

4.6.1 Guarantees for ATA

For each worker $i \in [n]$, recall that \bar{a}_i denote the prescribed allocation of the optimal action \bar{a} . Define k_i as the smallest integer satisfying

$$(\bar{a}_i + k_i)\mu_i > \ell(\bar{a}, \mu) . \quad (4.10)$$

From the definition above, it follows that if the learner plays an action a^k at round k such that $a_i^k \geq \bar{a}_i + k_i$, then $\ell(a^k, \mu) \geq \ell(\bar{a}, \mu)$. Thus, k_i can be interpreted as the smallest number of additional units allocated to worker i that result in a suboptimal loss. Moreover, for every worker $i \in [n]$, we have $k_i \in \{1, 2\}$ (see Lemma C.4.1 in the Appendix).

The next result provides an upper bound on the expected regret of ATA.

Theorem 4.6.1 (Proof in Section C.4.1). Suppose that Assumption 4.3.1 holds. Then, the expected regret of ATA with inputs (B, α) satisfies

$$\begin{aligned} \mathcal{R}_K &\leq 2n \max_{i \in [n]} \{B\mu_i - \ell(\bar{a}, \mu)\} \\ &\quad + c \cdot \sum_{i=1}^n \frac{\alpha^2(\bar{a}_i + k_i)(B\mu_i - \ell(\bar{a}, \mu))}{((\bar{a}_i + k_i)\mu_i - \ell(\bar{a}, \mu))^2} \cdot \ln K , \end{aligned}$$

where $\alpha \geq \max_{i \in [n]} \|X_i - \mu_i\|_{\psi_1}$, and c is a numerical constant.

The first term in the regret upper bound is independent on the number of rounds K . The second term, however, grows logarithmically with K , which aligns with the behavior observed in stochastic bandit problems in the literature.

In the case where $B = 1$, our setting reduces to the problem of regret minimization for the standard multi-armed bandits. Observe that in this case $\ell(\bar{a}, \mu) = \min_{i \in [n]} \mu_i$, $k_i = 1$ for all $i \in [n]$. Therefore, the guarantees of Theorem 4.6.1 recover the known optimal bound

$$\mathcal{O} \left(\sum_i \frac{\ln(K)}{\Delta_i} \right)$$

of the standard MAB setting, where $\Delta_i := \mu_i - \min_j \mu_j$.

Proof sketch. In standard and combinatorial MAB problems, regret bounds are typically derived by controlling the number of rounds in which the learner selects suboptimal arms. These bounds are often of the order $\ln(K)/\Delta^2$, where Δ denotes the suboptimality gap and quantifies the exploration cost required to distinguish optimal actions from suboptimal ones.

In our setting, the problem is more complex since the learner must not only choose which arms to pull but also determine the allocation of resources across selected arms. With this in mind, we develop the following key arguments leading to the bound in Theorem 4.6.1.

We define *over-allocation* for worker i at round k as the event where $a_i^k \geq \bar{a}_i + k_i$. By definition of k_i (see (4.10)), this implies that $\ell(a^k, \mu) > \ell(\bar{a}, \mu)$. We define a *bad round* as a round where $\ell(a^k, \mu) > \ell(\bar{a}, \mu)$, and we say that a bad round is *triggered by arm i* when $a_i^k \mu_i = \ell(a^k, \mu) > \ell(\bar{a}, \mu)$. Then, the proof revolves around establishing an upper bound on the total number of bad rounds.

To derive this bound, we consider the number of samples required to verify that the mean computation time of worker i under over-allocation exceeds the optimal waiting time $\ell(\bar{a}, \mu)$. Specifically, we need to test whether the mean of the corresponding distribution, at least $(\bar{a}_i + k_i)\mu_i$, surpasses the threshold $\ell(\bar{a}, \mu)$. This is equivalent to testing whether

$$\left\{ \mu_i > \frac{\ell(\bar{a}, \mu)}{\bar{a}_i + k_i} \right\}.$$

Using the concentration inequality applied in our analysis, the number of samples required for this test is of the order:

$$\alpha_i^2 \left(\mu_i - \frac{\ell(\bar{a}, \mu)}{\bar{a}_i + k_i} \right)^{-2} = \frac{\alpha_i^2 (\bar{a}_i + k_i)^2}{((\bar{a}_i + k_i)\mu_i - \ell(\bar{a}, \mu))^2}. \quad (4.11)$$

During rounds where worker i is over-allocated, the learner collects at least $\bar{a}_i + k_i$ samples from the corresponding distribution. Therefore, the total number of rounds required to accumulate enough samples to stop over-allocating worker i can be upper-bounded by

$$\frac{\alpha_i^2 (\bar{a}_i + k_i)}{((\bar{a}_i + k_i)\mu_i - \ell(\bar{a}, \mu))^2}.$$

In the regret bound of Theorem 4.6.1, the term α^2 appears instead of α_i^2 because the learner's prior knowledge is limited to an upper bound $\alpha \geq \max_i \|X_i - \mu_i\|_{\psi_1}$ on the maximal Orlicz norm of the arm distributions. Finally, considering that the worst-case excess loss incurred when over-allocating worker i is $B\mu_i - \ell(\bar{a}, \mu)$, we obtain the stated bound.

4.6.2 Guarantees for ATA-Empirical

We present theoretical guarantees for ATA-Empirical by providing an upper bound on the expected cumulative regret (4.9). As discussed in Section 4.4, ATA-Empirical leverages lower confidence bounds derived from a novel data-dependent concentration inequality introduced below. The proof of this result is detailed in Section C.5.

Lemma 4.6.2. Let X_1, \dots, X_n be i.i.d. positive random variables with mean μ , such that $\alpha = \|X_1 - \mu\|_{\psi_1} < +\infty$. Let \hat{X}_n denote the empirical mean. For $\delta \in (0, 1)$, let

$$C_{n,\delta} := 2\sqrt{\frac{\log(2/\delta)}{n}} + 2\frac{\log(2/\delta)}{n} ,$$

where $\eta = \alpha/\mu$. Then, with probability at least $1 - \delta$, we have

$$\mu \geq \hat{X}_n (1 - \eta C_{n,\delta})_+ .$$

Moreover, if $\eta C_{n,\delta} \leq 1/4$, then, we have with probability at least $1 - \delta$, we have

$$\hat{X}_n (1 - \eta C_{n,\delta})_+ \leq \mu \leq \hat{X}_n \left(1 + \frac{4}{3}\eta C_{n,\delta}\right) .$$

Using the concentration inequality above, we construct the lower confidence bounds \hat{s}_i^k as defined in (4.8). The following theorem provides an upper bound on the regret of ATA-Empirical.

Theorem 4.6.3 (Proof in Section C.4.2). Suppose that Assumption 4.3.1 holds. Then, the expected regret of ATA-Empirical with inputs (B, η) , satisfies

$$\begin{aligned} \mathcal{R}_K &\leq 2n \max_{i \in [n]} \{B\mu_i - \ell(\bar{a}, \mu)\} \\ &\quad + c\eta^2 \cdot \sum_{i=1}^n \frac{\mu_i^2(\bar{a}_i + k_i)(B\mu_i - \ell(\bar{a}, \mu))}{((\bar{a}_i + k_i)\mu_i - \ell(\bar{a}, \mu))^2} \cdot \ln K , \end{aligned}$$

where $\eta \geq \max_{i \in [n]} \alpha_i/\mu_i$ and c is a numerical constant.

Comparing the bounds for ATA-Empirical and ATA, we observe a key differences. Unlike the bound in Theorem 4.6.1, which incurs a squared maximal Orlicz norm penalty of α^2 for all terms in the upper bound, ATA-Empirical benefits from its adaptive nature, leading to a term-specific factor of $\eta^2\mu_i^2$. In the case where the arm distributions have a ratio α_i/μ_i of the same order (such as the exponential distributions), the bound of Theorem 4.6.3 shows that ATA-Empirical adapts to the quantities α_i as we have, in the last case, $\eta\mu_i = \alpha_i$.

4.7 Experiments

In this section, we validate our algorithms by simulating a scenario with n workers, where we solve a simple problem using SGD. In each iteration, we collect $B = 23$ gradients from the workers and perform a gradient descent step.

The objective function $f : \mathbb{R}^d \rightarrow \mathbb{R}$ is a convex quadratic defined as

$$f(x) = \frac{1}{2}x^\top Ax - b^\top x ,$$

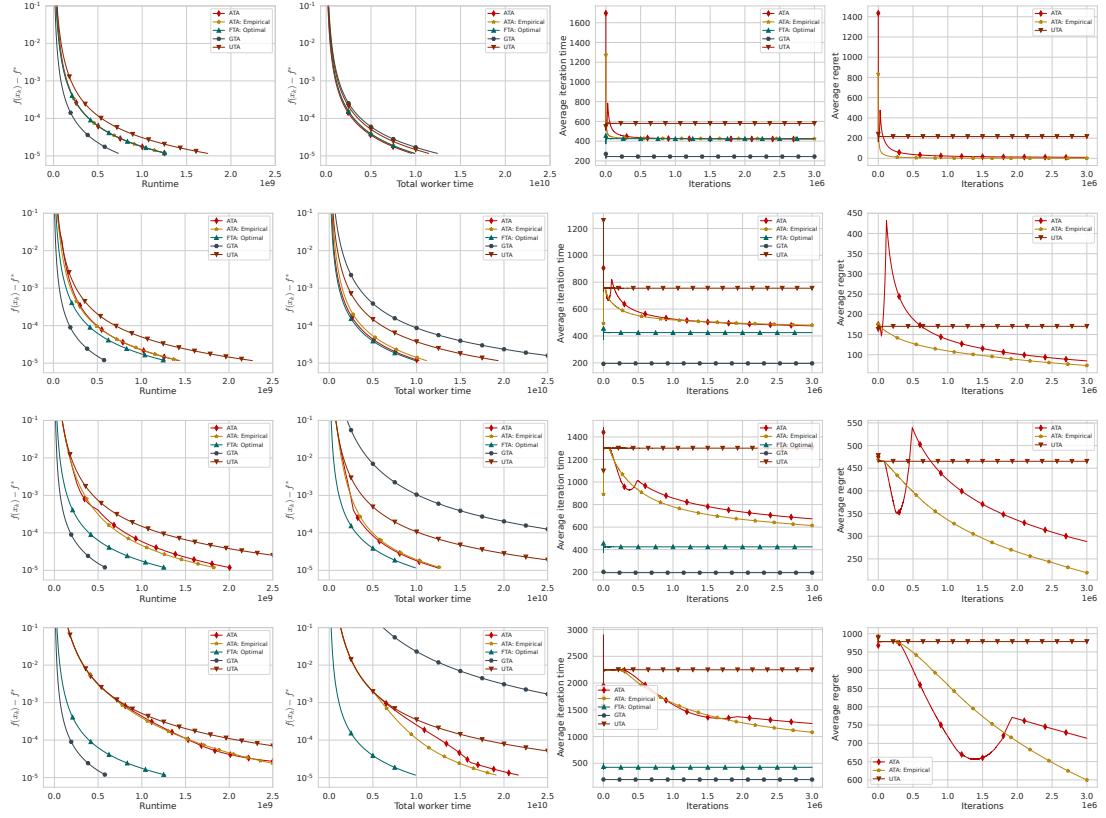


Figure 4.7.1: Each row increases the number of workers by a factor of 3, starting from 17, that is, $n = 17, 51, 153, 459$ from top to bottom. The first column shows runtime vs. suboptimality. The second column also plots suboptimality, but against total worker time, i.e., $\sum_{i=1}^n T_i^k$ in Algorithm 4.4.1. The third column presents the average iteration time, given by c_k/k over all iterations k . The last column displays the averaged cumulative regret, as defined in (4.9).

where

$$A = \frac{1}{4} \begin{bmatrix} 2 & -1 & & 0 \\ -1 & \ddots & \ddots & \\ & \ddots & \ddots & -1 \\ 0 & & -1 & 2 \end{bmatrix} \in \mathbb{R}^{d \times d}, \quad b = \frac{1}{4} \begin{bmatrix} -1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \in \mathbb{R}^d.$$

We denote f^* as the minimum value of the function f . Each of the n workers is able to calculate unbiased stochastic gradients $g(x)$ that satisfy

$$\mathbb{E} [\|g(x) - \nabla f(x)\|^2] \leq 0.01^2.$$

This is achieved by adding Gaussian noise to the gradients of f .

The computation time for worker i is modeled by the distribution

$$\nu_i = 29\sqrt{i} + \text{Exp}\left(29\sqrt{i}\right),$$

for all $i \in [n]$, where $\text{Exp}(\beta)$ denotes the exponential distribution with scale parameter β . The expected value of this distribution is $\mu_i = 2 \cdot 29\sqrt{i}$. Furthermore,

the Orlicz norm satisfies the bound $\alpha_i \leq 2\mu_i$.

We consider three benchmark algorithms. **GTA-SGD**, originally introduced as **Rennala SGD** by [Tyurin & Richtárik \(2024\)](#). Additionally, we include **OFTA** (Optimal Fixed Task Allocation), which assumes the oracle knowledge of the mean computation times and uses the optimal allocation \bar{a} in (4.9) in each iteration, and **UTA** (Uniform Task Allocation), which distributes B tasks uniformly among the n workers. If $n > B$, then in **UTA** we select B workers at random, each one tasked to calculate one stochastic gradient. Our algorithms aim to achieve a performance close to the one of **OFTA**, without any prior knowledge of the true means.

For **ATA**, we set $\alpha = \alpha_n = 4 \cdot 29\sqrt{n}$, while for **ATA-Empirical** we use $\eta = 1$. The results of our experiments are shown in Figure 4.7.1. As expected, **GTA** is the fastest in terms of runtime (first column), but it performs poorly in terms of total worker time (second column). This is because it uses all devices, most of which perform useless computations that are never used, leading to worse performance as the number of workers increases. In fact, its performance can become arbitrarily worse. On the other hand, **OFTA** performs best in terms of total worker time. Although it is slower in terms of runtime, the difference is by a constant factor that does not increase as n grows. This is because additional workers are less efficient and do not provide significant benefits for **GTA**.

Turning our attention to our algorithms, both **ATA** and **ATA-Empirical** initially behave like **UTA**, as it is expected by the need to perform an initial exploration phase with uniform allocations. However, after this phase, they begin to converge to the performance of **OFTA**.

The last two columns contain plots that confirm our theoretical derivations. The third plot validates Theorem 4.4.2, showing that **ATA** and **ATA-Empirical** converge to **OFTA** up to a constant. The final column shows the averaged cumulative regret, vanishing over time as predicted by Theorems 4.6.1 and 4.6.3.

Table 4.1: Ratios of total worker times and runtimes required to achieve $f(x) - f^* < 10^{-5}$. For total worker time, we divide the total worker time of **GTA** by the corresponding total worker times of the other algorithms listed. For runtime, we do the opposite, dividing the runtime of the other algorithms by the runtime of **GTA**, since **GTA** is the fastest. To simplify the naming, we refer to **ATA-Empirical** as **ATA-E**.

n	TOT. WORKER TIME RATIO			RUNTIME RATIO		
	ATA	ATA-E	OFTA	ATA	ATA-E	OFTA
17	1.3	1.26	1.26	1.73	1.75	1.74
51	2.91	2.69	3.03	2.43	2.45	2.17
153	7.22	7.02	9.1	3.44	3.14	2.17
459	12.45	14.1	27.3	6.36	5.51	2.17

In Table 4.1, we compare the results numerically. Both the total worker time ratio and runtime ratio increase as n grows. The total worker time ratio increases because **GTA** becomes less efficient, using more resources than necessary. The

runtime ratio grows for ATA and ATA-Empirical since a larger number of workers requires more exploration. However, for OFTA this ratio remains unchanged, as discussed earlier.

We remark that in these experiments we started all runs for ATA and ATA-Empirical without prior knowledge of the computation time distribution. However, in real systems, where these algorithms are used multiple times, prior estimates of computation times from previous runs could be available. With this information, ATA and ATA-Empirical would be much faster, as they would spend less time on exploration, approaching the performance of OFTA in a faster way. We validate this through experiments presented in Section C.1.5.

In Section C.1.1, we conducted similar experiments with a different time distribution, where the mean times vary linearly across the arms. In Section C.1.2, we examine scenarios with varying client time distributions. Additionally, in Section C.1.3, we analyze regret performance, confirming its logarithmic behavior as predicted by Theorems 4.6.1 and 4.6.3. Finally, in Section C.1.4, we trained a simple CNN on the CIFAR-100 dataset (Krizhevsky et al., 2009) using Adam (Kingma & Ba, 2014).

Chapter 5

Concluding Remarks

Asynchronous methods address the fundamental inefficiency of synchronous distributed training, where synchronization barriers cause faster workers to idle while waiting for slower ones. However, prior to this work, asynchronous methods lacked rigorous theoretical foundations and had not demonstrated any theoretical advantages over synchronous methods.

To address this gap, we began with a clean theoretical setting that isolates the core problem asynchronous methods are designed to solve: heterogeneous computation times across workers. We studied the data-parallel regime—where all workers maintain copies of the full model but process different data—using first-order stochastic methods for smooth nonconvex optimization. This setting allowed us to abstract away factors orthogonal to asynchrony (such as communication delays and memory consistency) and focus on the fundamental question: can asynchronous methods be provably better?

In this setting, existing analyses often relied on restrictive assumptions, and crucially, no asynchronous algorithm was known to achieve optimal time complexity—a benchmark attained only by synchronous methods (Tyurin & Richtárik, 2024).

This dissertation establishes that asynchronous methods can indeed achieve optimal time complexity. Through careful algorithm design and rigorous analysis, we demonstrated that staleness—the central challenge in asynchronous optimization—can be controlled through principled mechanisms such as selective gradient discarding and structured gradient buffering. Our work provides the foundational theory showing that asynchronous methods can be both theoretically optimal and practically efficient in the fundamental setting of data-parallel stochastic optimization.

5.1 Summary of Contributions

Within this theoretical framework, we developed three main contributions:

5.1.1 Ringmaster ASGD (Chapter 2)

We introduced the first asynchronous SGD algorithm that achieves optimal time complexity in the homogeneous data setting. Ringmaster ASGD (Algorithm 2.3.2) can be viewed as Asynchronous SGD (Algorithm 2.1.1) with selective gradient discarding: it rejects gradients whose delay exceeds a threshold R , filtering out stale updates that would otherwise harm convergence. With an appropriate choice of R , Ringmaster ASGD achieves optimal time complexity in both the fixed computation time (2.1) and arbitrarily varying computation time (Section 2.5) settings. Moreover, our experiments demonstrate that Ringmaster ASGD can be faster than

the optimal synchronous method **Rennala SGD** (Tyurin & Richtárik, 2024) in practice, due to more frequent model updates and the absence of synchronization overhead.

5.1.2 Ringleader ASGD (Chapter 3)

We extended optimal asynchronous methods to the heterogeneous data regime, where local objectives differ across workers—a setting typical of federated learning (Konečný et al., 2016c,a; McMahan et al., 2017).

The key algorithmic innovation combines two mechanisms. First, we maintain a *gradient table* that stores gradients per worker, and use the average of all stored gradients for model updates. This approach eliminates the need for restrictive similarity assumptions between local objectives, which rarely hold in federated learning due to diverse client data. Second, we control staleness through *structured rounds*: each round performs exactly n updates (one per worker), ensuring that both the local models and the gradients in the table remain fresh. This design keeps delays bounded without requiring explicit delay thresholds.

With these mechanisms, **Ringleader ASGD** achieves optimal time complexity in both the fixed and arbitrarily varying computation time settings. Importantly, every gradient received by the server is either used immediately or stored for future updates—no computation is wasted, unlike synchronous methods that discard work at synchronization barriers. This makes **Ringleader ASGD** the first asynchronous method proven optimal for heterogeneous data without similarity assumptions. Moreover, our numerical experiments (Section 3.7) demonstrate that **Ringleader ASGD** can outperform the theoretically optimal synchronous method **Malenia SGD** (Tyurin & Richtárik, 2024) in practice, precisely because it avoids the computational waste inherent to synchronization.

5.1.3 Adaptive Task Allocation (Chapter 4)

This chapter addresses a different but equally important concern: *computational efficiency*. While the previous optimal methods minimize wall-clock time, they achieve this by utilizing all available computational resources. In large-scale systems, such full utilization can lead to unnecessary and wasteful computation without proportional benefit. This raises a natural question: *Can we reduce computational waste without significantly increasing the total wall-clock time?*

To investigate this, we focus on the homogeneous data setting and build upon **Ringmaster ASGD** and **Rennala SGD**. We assume that worker computation times follow unknown probability distributions. If these distributions were known, one could allocate the optimal number of tasks to each worker to minimize the expected wall-clock time. We use this optimal fixed allocation as our competitor—the benchmark against which we measure performance.

Our goal is to design an algorithm that performs nearly as well as the best fixed competitor, but without prior knowledge of the worker-time distributions. We achieve this with our proposed method, **ATA**. **ATA** formulates task allocation as a multi-armed bandit problem, learning the workers’ compute-time distributions online while balancing exploration and exploitation. We prove that **ATA** matches the performance of the best fixed competitor up to a constant multiplicative

factor in expectation, a standard guarantee in online learning known as constant regret.

5.2 Future Directions

Several important directions remain for future research. We organize these into two categories: extensions specific to individual algorithms developed in this dissertation, and broader research directions that could apply to all the methods presented here or to asynchronous optimization more generally.

5.2.1 Algorithm-Specific Extensions

The following directions represent natural extensions of individual algorithms that would address their current limitations or enhance their specific capabilities.

Utilizing discarded gradients in Ringmaster ASGD. Currently, Ringmaster ASGD completely discards gradients whose delay exceeds the threshold R . An interesting question is whether these stale gradients could be incorporated with reduced weights rather than being discarded entirely, similar to delay-adaptive asynchronous SGD methods (Koloskova et al., 2022; Mishchenko et al., 2022a). Such an approach might improve wall-clock convergence time by making better use of available computation. Moreover, in data center settings, consistently slow workers may never contribute if their gradients always exceed the delay threshold, meaning their data is never incorporated into training. Developing mechanisms to include such workers—perhaps through delay-dependent weighting or adaptive thresholds—could improve both data utilization and model quality while maintaining near-optimal convergence guarantees.

Client sampling in Ringleader ASGD. Ringleader ASGD requires gradients from *all* workers before proceeding with updates, which may be impractical in federated learning where clients frequently become unavailable due to network issues, battery constraints, or user inactivity. A natural extension would allow Ringleader ASGD to work with only a subset of available clients in each round, rather than waiting for all workers. However, this modification likely requires reintroducing similarity assumptions between local objectives to maintain convergence guarantees. A complementary research direction would be to establish lower bounds for asynchronous methods under these similarity assumptions.

Closing the optimality gap in Ringleader ASGD. Our analysis shows that Ringleader ASGD attains existing lower bounds when the new smoothness constant (Assumption 3.3.2) is within a constant factor of the standard smoothness parameter. However, a gap remains: it is unclear whether this constant factor is fundamental or whether Ringleader ASGD’s complexity can be further improved. Future work could address this by either (i) relaxing the assumptions under which optimality is established or (ii) deriving tighter lower bounds tailored to asynchronous methods in the heterogeneous setting. We believe the latter is more promising—new, sharper lower bounds for asynchronous algorithms may

ultimately determine whether Ringleader ASGD is truly optimal or if additional improvements are still possible.

Adaptive task allocation for heterogeneous data. The ATA algorithm was developed for the homogeneous data setting, where it improves the computational efficiency of Ringmaster ASGD and Rennala SGD (Tyurin & Richtárik, 2024). A natural extension would be to adapt these ideas to the heterogeneous data regime, thereby enhancing the computational efficiency of Ringleader ASGD and Malenia SGD (Tyurin & Richtárik, 2024). Such an extension would need to account for the different criteria for stopping gradient collection, which in this case depend on the harmonic mean of the gradient counts.

Adaptive task allocation under adversarial delays. The existing ATA framework assumes worker computation times follow unknown but fixed distributions. In federated learning, however, worker behavior can be highly unpredictable: devices may become unavailable, experience extreme delays, or exhibit non-stationary patterns that do not follow any stable distribution. Developing allocation strategies robust to such adversarial or non-stochastic delays would significantly enhance practical applicability. This direction connects to the adversarial online learning and bandit algorithm literature (Auer et al., 2002).

5.2.2 Broader Research Directions

Beyond extensions of individual algorithms, several fundamental questions could enhance all the methods in this dissertation or advance asynchronous optimization more broadly.

Beyond SGD. Our analysis has focused exclusively on stochastic gradient descent, the most fundamental first-order optimization method. However, modern deep learning frequently relies on more advanced optimizers such as Adam (Kingma & Ba, 2014), AdamW (Loshchilov & Hutter, 2019), and preconditioned gradient methods like Shampoo (Gupta et al., 2018), as well as newer algorithms such as Muon (Jordan et al., 2024). Extending our asynchronous framework to these methods while preserving theoretical guarantees would significantly broaden the practical scope of this work.

Extensions to other forms of parallelism. This dissertation focused exclusively on data parallelism. However, modern large-scale training systems increasingly combine data parallelism with model and pipeline parallelism (Narayanan et al., 2021; Shoeybi et al., 2019). Extending asynchronous optimization methods to such hybrid settings is a natural next step. Of particular interest is developing time-complexity analyses for asynchronous pipeline parallelism, where different stages of a model are executed on distinct workers and subject to inter-stage delays. Several recent works have explored asynchronous pipeline training in this context (Yang et al., 2019; Ajanthan et al., 2025; Kong et al., 2025).

Communication costs and network delays. Throughout this work, we assumed instantaneous communication—a simplification that allowed us to isolate

the fundamental challenges of asynchrony. In federated and geographically distributed systems, however, communication often dominates computation time. To mitigate this, existing approaches employ either communication compression (Alistarh et al., 2017) or local training (McMahan et al., 2016; Mishchenko et al., 2022b; Malinovsky et al., 2022; Maranjyan et al., 2025c) to reduce the frequency or volume of transmitted updates. Combining asynchrony with these techniques—local training or communication compression—and analyzing the resulting time complexity remains an important open problem. Initial progress has been made in both directions, with recent results on asynchronous compression (Tyurin et al., 2024a) and asynchronous local training (Tyurin & Sivtsov, 2025; Fradin et al., 2025).

Lock-free shared memory algorithms. Throughout this work, we assumed atomic reads and writes with instantaneous communication. An alternative model, studied in works such as HOGWILD! (Recht et al., 2011), considers shared memory systems where multiple workers can read and write simultaneously without locks. In this setting, a worker reading the parameter vector may observe inconsistent values—a mixture of the current server state and partial updates from other workers. Recent works (Mania et al., 2017; Leblond et al., 2018; Nguyen et al., 2018) have provided improved analyses of HOGWILD!-style algorithms. It would be interesting to investigate whether the techniques developed in this dissertation can be adapted to this lock-free setting and whether optimal time complexity can be achieved despite inconsistent reads.

Large-scale empirical validation. The experiments in this dissertation were primarily small-scale demonstrations designed to validate theoretical predictions. A valuable direction for future work is large-scale empirical validation in realistic settings: training large language models or other modern architectures in actual data center or federated learning environments. Such experiments would reveal practical bottlenecks not captured by our theoretical model, and would help bridge the gap between theory and deployment.

5.3 Closing Remarks

This dissertation establishes that asynchronous methods can achieve optimal time complexity, providing the first rigorous theoretical foundation for their use in distributed optimization. While our results focus on the fundamental setting of data-parallel stochastic gradient descent applied to smooth nonconvex objectives, they mark an essential first step toward a comprehensive theory of asynchronous optimization.

Beyond the specific optimality results, this work introduces broadly applicable techniques: principled mechanisms for controlling staleness (e.g., selective gradient discarding and structured gradient buffering), a rigorous framework for analyzing convergence and time complexity under asynchrony, and strategies for balancing wall-clock time with computational cost. Together, these ideas lay the groundwork for extending asynchronous methods to more complex and realistic environments.

The significance of this work extends beyond theory. Asynchronous algorithms have historically seen limited practical adoption, hindered by weak theoretical guarantees and limited software support. By demonstrating that they can be both theoretically optimal and practically efficient, this dissertation provides strong motivation for developing robust implementations and integrating these methods into large-scale training systems. As AI systems continue to grow in scale and as training infrastructure becomes increasingly heterogeneous—spanning data centers, edge devices, and federated networks—the importance of efficient asynchronous optimization will only continue to rise.

The central message of this dissertation is one of possibility rather than limitation. Synchronous algorithms offer predictability but waste resources through forced idleness. Asynchronous algorithms eliminate this waste but introduce the challenge of managing stale information. This dissertation has shown that the trade-off is not fundamental: through careful algorithm design, staleness can be controlled while maintaining both theoretical optimality and practical efficiency. The future of large-scale machine learning lies not in forcing all workers to march in lockstep, but in enabling them to contribute asynchronously—coordinated through principled mechanisms that ensure convergence and efficiency. As demonstrated here, such coordination is not only possible—it is optimal.

REFERENCES

- Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.
- Alekh Agarwal and John C Duchi. Distributed delayed stochastic optimization. *Advances in Neural Information Processing Systems*, 24, 2011.
- International Energy Agency. Energy and AI, April 2025. URL <https://www.iea.org/reports/energy-and-ai>.
- Thalaiyasingam Ajanthan, Sameera Ramasinghe, Yan Zuo, Gil Avraham, and Alexander Long. Nesterov method for asynchronous pipeline parallel optimization. In *International Conference on Machine Learning*, 2025.
- Abdelkrim Alahyane, Céline Comte, Matthieu Jonckheere, and Éric Moulines. Optimizing asynchronous federated learning: A delicate trade-off between model-parameter staleness and update frequency. *arXiv preprint arXiv:2502.08206*, 2025.
- Dan Alistarh, Demjan Grubic, Jerry Li, Ryota Tomioka, and Milan Vojnovic. QSGD: Communication-efficient SGD via gradient quantization and encoding. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 1709–1720, 2017.
- Ganesh Ananthanarayanan, Ali Ghodsi, Scott Shenker, and Ion Stoica. Effective straggler mitigation: attack of the clones. In *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation, nsdi’13*, pp. 185–198, USA, 2013. USENIX Association.
- Yossi Arjevani, Ohad Shamir, and Nathan Srebro. A tight convergence analysis for stochastic gradient descent with delayed updates. In *Algorithmic Learning Theory*, pp. 111–132. PMLR, 2020.
- Yossi Arjevani, Yair Carmon, John C Duchi, Dylan J Foster, Nathan Srebro, and Blake Woodworth. Lower bounds for non-convex stochastic optimization. *Mathematical Programming*, pp. 1–50, 2022.

Mahmoud Assran, Arda Aytekin, Hamid Reza Feyzmahdavian, Mikael Johansson, and Michael G Rabbat. Advances in asynchronous parallel and distributed optimization. *Proceedings of the IEEE*, 108(11):2013–2031, 2020.

P Auer. Finite-time analysis of the multiarmed bandit problem, 2002.

Peter Auer, Nicolò Cesa-Bianchi, Yoav Freund, and Robert E. Schapire. The nonstochastic multiarmed bandit problem. *SIAM Journal on Computing*, 32(1):48–77, 2002. doi: 10.1137/S0097539701398375. URL <https://doi.org/10.1137/S0097539701398375>.

Arda Aytekin, Hamid Reza Feyzmahdavian, and Mikael Johansson. Analysis and implementation of an asynchronous optimization algorithm for the parameter server. *arXiv preprint arXiv:1610.05507*, 2016.

Doron Blatt, Alfred O Hero, and Hillel Gauchman. A convergent incremental gradient method with a constant step size. *SIAM Journal on Optimization*, 18(1):29–51, 2007.

Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jilai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

Léon Bottou, Frank E. Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *SIAM Review*, 60(2):223–311, 2018. doi: 10.1137/16M1080173. URL <https://doi.org/10.1137/16M1080173>.

Stéphane Boucheron, Olivier Bousquet, Gábor Lugosi, and Pascal Massart. Moment inequalities for functions of independent random variables. 2005.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 1877–1901. Curran Associates, Inc., 2020. URL <https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfc4967418bfb8ac142f64a-Paper.pdf>.

- Nicolo Cesa-Bianchi and Gábor Lugosi. Combinatorial bandits. *Journal of Computer and System Sciences*, 78(5):1404–1422, 2012.
- Jianmin Chen, Xinghao Pan, Rajat Monga, Samy Bengio, and Rafal Jozefowicz. Revisiting distributed synchronous SGD. *arXiv preprint arXiv:1604.00981*, 2016a.
- Wei Chen, Yajun Wang, and Yang Yuan. Combinatorial multi-armed bandit: General framework and applications. In *International Conference on Machine Learning*, pp. 151–159. PMLR, 2013.
- Wei Chen, Wei Hu, Fu Li, Jian Li, Yu Liu, and Pinyan Lu. Combinatorial multi-armed bandit with general reward functions. *Advances in Neural Information Processing Systems*, 29, 2016b.
- Pranav Singh Chib and Pravendra Singh. Recent advancements in end-to-end autonomous driving using deep learning: A survey. *IEEE Transactions on Intelligent Vehicles*, 9(1):103–118, 2023.
- Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113, 2023.
- Alon Cohen, Amit Daniely, Yoel Drori, Tomer Koren, and Mariano Schain. Asynchronous stochastic optimization robust to arbitrary delays. *Advances in Neural Information Processing Systems*, 34:9024–9035, 2021.
- Richard Combes, M. Sadegh Talebi, Alexandre Proutiere, and Marc Lelarge. Combinatorial bandits revisited. *Advances in Neural Information Processing Systems*, 28, 2015.
- Laurent Condat, Artavazd Maranjyan, and Peter Richtárik. LoCoDL: Communication-efficient distributed learning with local training and compression. In *Proc. of International Conference on Learning Representations (ICLR)*, 2025.
- Andrew Cotter, Ohad Shamir, Nati Srebro, and Karthik Sridharan. Better mini-batch algorithms via accelerated gradient methods. *Advances in Neural Information Processing Systems*, 24, 2011.
- Ben Cottier, Robi Rahman, Loredana Fattorini, Nestor Maslej, Tamay Besiroglu, and David Owen. The rising costs of training frontier AI models. *arXiv preprint arXiv:2405.21015*, 2024.

Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for YouTube recommendations. In *Proceedings of the 10th ACM conference on recommender systems*, pp. 191–198, 2016.

Alex De Vries. The growing energy footprint of artificial intelligence. *Joule*, 7(10):2191–2194, 2023.

Jeffrey Dean and Luiz André Barroso. The tail at scale. *Communications of the ACM*, 56(2):74–80, 2013.

Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc' aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, Quoc Le, and Andrew Ng. Large scale distributed deep networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger (eds.), *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. URL https://proceedings.neurips.cc/paper_files/paper/2012/file/6aca97005c68f1206823815f66102863-Paper.pdf.

Arthur Douillard, Qixuan Feng, Andrei A. Rusu, Rachita Chhaparia, Yani Donchev, Adhiguna Kuncoro, Marc'Aurelio Ranzato, Arthur Szlam, and Jiajun Shen. DiLoCo: Distributed low-communication training of language models, 2024. URL <https://arxiv.org/abs/2311.08105>.

Sanghamitra Dutta, Gauri Joshi, Soumyadip Ghosh, Parijat Dube, and Priya Nagpurkar. Slow and stale gradients can win the race: Error-runtime trade-offs in distributed SGD. In *International Conference on Artificial Intelligence and Statistics*, pp. 803–812. PMLR, 2018.

Cooper Elsworth, Keguo Huang, David Patterson, Ian Schneider, Robert Sedivy, Savannah Goodman, Ben Townsend, Parthasarathy Ranganathan, Jeff Dean, Amin Vahdat, et al. Measuring the environmental impact of delivering AI at Google scale. *arXiv preprint arXiv:2508.15734*, 2025.

Andre Esteva, Brett Kuprel, Roberto A Novoa, Justin Ko, Susan M Swetter, Helen M Blau, and Sebastian Thrun. Dermatologist-level classification of skin cancer with deep neural networks. *nature*, 542(7639):115–118, 2017.

Hamid Reza Feyzmahdavian and Mikael Johansson. Asynchronous iterations in optimization: New sequence results and sharper algorithmic guarantees. *Journal of Machine Learning Research*, 24(158):1–75, 2023.

Hamid Reza Feyzmahdavian, Arda Aytekin, and Mikael Johansson. An asynchronous mini-batch algorithm for regularized stochastic optimization. *IEEE Transactions on Automatic Control*, 61(12):3740–3754, 2016.

- Yann Fraboni, Richard Vidal, Laetitia Kameni, and Marco Lorenzi. A general theory for federated optimization with asynchronous and heterogeneous clients updates. *Journal of Machine Learning Research*, 24(110):1–43, 2023.
- Adrien Fradin, Peter Richtárik, and Alexander Tyurin. Local SGD and federated averaging through the lens of time complexity. *arXiv preprint arXiv:2509.23207*, 2025.
- Yi Gai, Bhaskar Krishnamachari, and Rahul Jain. Combinatorial network optimization with unknown variables: Multi-armed bandits with linear rewards and individual observations. *IEEE/ACM Transactions on Networking*, 20(5):1466–1478, 2012.
- Erol Gelenbe and Isi Mitrani. *Analysis and synthesis of computer systems*, volume 4. World Scientific, 2010.
- Saeed Ghadimi and Guanghui Lan. Stochastic first-and zeroth-order methods for nonconvex stochastic programming. *SIAM Journal on Optimization*, 23(4):2341–2368, 2013.
- Margalit R Glasgow and Mary Wootters. Asynchronous distributed optimization with stochastic delays. In *International Conference on Artificial Intelligence and Statistics*, pp. 9247–9279. PMLR, 2022.
- Robert Mansel Gower, Nicolas Loizou, Xun Qian, Alibek Sailanbayev, Egor Shulgin, and Peter Richtárik. SGD: General analysis and improved rates. In *International Conference on Machine Learning*, pp. 5200–5209. PMLR, 2019.
- Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch SGD: Training ImageNet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.
- Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.
- Martin Grigoryan and Artavazd Maranjyan. On the divergence of Fourier series in the general Haar system. *Armenian Journal of Mathematics*, 13(6):1–10, 2021a.

Martin Grigoryan, Anna Kamont, and Artavazd Maranjyan. Menshov-type theorem for divergence sets of sequences of localized operators. *Journal of Contemporary Mathematical Analysis (Armenian Academy of Sciences)*, 58(2):81–92, 2023.

Tigran M Grigoryan and Artavazd Maranjyan. On the unconditional convergence of Faber-Schauder series in L^1 . *Proceedings of the YSU A: Physical and Mathematical Sciences*, 55(1 (254)):12–19, 2021b.

Donald Gross, John F Shortle, James M Thompson, and Carl M Harris. *Fundamentals of queueing theory*, volume 627. John Wiley & Sons, 2011.

Vineet Gupta, Tomer Koren, and Yoram Singer. Shampoo: Preconditioned stochastic tensor optimization. In *International Conference on Machine Learning*, pp. 1842–1850. PMLR, 2018.

Mert Gurbuzbalaban, Asuman Ozdaglar, and Pablo A Parrilo. On the convergence rate of incremental aggregated gradient algorithms. *SIAM Journal on Optimization*, 27(2):1035–1048, 2017.

Stefan Hadjis, Ce Zhang, Ioannis Mitliagkas, Dan Iter, and Christopher Ré. Omniprivate: An optimizer for multi-device deep learning on CPUs and GPUs. *arXiv preprint arXiv:1606.04487*, 2016.

J. Edward Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. 2022.

Rustem Islamov, Mher Safaryan, and Dan Alistarh. AsGrad: A sharp unified analysis of asynchronous-SGD algorithms. In *International Conference on Artificial Intelligence and Statistics*, pp. 649–657. PMLR, 2024.

Sashank J Reddi, Ahmed Hefny, Suvrit Sra, Barnabas Poczos, and Alexander J Smola. On variance reduction in stochastic gradient descent and its asynchronous variants. *Advances in Neural Information Processing Systems*, 28, 2015.

Keller Jordan, Yuchen Jin, Vlado Boza, Jiacheng You, Franz Cesista, Laker Newhouse, and Jeremy Bernstein. Muon: An optimizer for hidden layers in neural networks, 2024. URL <https://kellerjordan.github.io/posts/muon/>.

Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, and Robert Boyle. In-datacenter performance analysis of a tensor processing unit.

In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, pp. 1–12, June 2017.

Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. Advances and open problems in federated learning. *Foundations and Trends® in Machine Learning*, 14(1–2):1–210, 2021.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Anastasiia Koloskova, Sebastian U Stich, and Martin Jaggi. Sharper convergence guarantees for asynchronous SGD for distributed and federated learning. *Advances in Neural Information Processing Systems*, 35:17202–17215, 2022.

Jakub Konečný, H Brendan McMahan, Daniel Ramage, and Peter Richtárik. Federated optimization: Distributed machine learning for on-device intelligence. *arXiv preprint arXiv:1610.02527*, 2016a.

Jakub Konečný, H. Brendan McMahan, Felix X. Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. In *NIPS Workshop on Private Multi-Party Machine Learning*, 2016b.

Jakub Konečný, H. Brendan McMahan, Felix X. Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. In *NIPS Workshop on Private Multi-Party Machine Learning*, 2016c.

Boao Kong, Xu Huang, Yuqi Xu, Yixuan Liang, Bin Wang, and Kun Yuan. Clapping: Removing per-sample storage for pipeline parallel distributed optimization with communication compression. *arXiv preprint arXiv:2509.19029*, 2025.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. Technical report, University of Toronto, Toronto, 2009.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

- Branislav Kveton, Zheng Wen, Azin Ashkan, and Csaba Szepesvari. Tight regret bounds for stochastic combinatorial semi-bandits. In *Artificial Intelligence and Statistics*, pp. 535–543. PMLR, 2015.
- T. Lattimore and C. Szepesvári. *Bandit algorithms*. Cambridge University Press, 2020.
- Remi Leblond, Fabian Pedregosa, and Simon Lacoste-Julien. Improved asynchronous parallel optimization analysis for stochastic incremental methods. *Journal of Machine Learning Research*, 19(81):1–68, 2018. URL <http://jmlr.org/papers/v19/17-650.html>.
- Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
- Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]*. Available: <http://yann.lecun.com/exdb/mnist>, 2, 2010.
- Mu Li, David G Andersen, Alexander Smola, and Kai Yu. Communication efficient distributed machine learning with the parameter server. *Advances in Neural Information Processing Systems*, 27, 2014.
- Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, et al. Pytorch distributed: Experiences on accelerating data parallel training. *arXiv preprint arXiv:2006.15704*, 2020a.
- Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks. *Proceedings of Machine Learning and Systems*, 2:429–450, 2020b.
- Xiangru Lian, Yijun Huang, Yuncheng Li, and Ji Liu. Asynchronous parallel stochastic gradient for nonconvex optimization. *Advances in Neural Information Processing Systems*, 28, 2015.
- Xiangru Lian, Wei Zhang, Ce Zhang, and Ji Liu. Asynchronous decentralized parallel stochastic gradient descent. In *International Conference on Machine Learning*, pp. 3043–3052. PMLR, 2018.
- Tian Lin, Jian Li, and Wei Chen. Stochastic online greedy learning with semi-bandit feedbacks. *Advances in Neural Information Processing Systems*, 28, 2015.

- Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. 2019.
- Yucheng Lu and Christopher De Sa. Optimal complexity in decentralized training. In *International Conference on Machine Learning*, pp. 7111–7123. PMLR, 2021.
- Grigory Malinovsky, Kai Yi, and Peter Richtárik. Variance reduced ProxSkip: Algorithm, theory and application to federated learning. *Advances in Neural Information Processing Systems*, 35:15176–15189, 2022.
- Horia Mania, Xinghao Pan, Dimitris Papailiopoulos, Benjamin Recht, Kannan Ramchandran, and Michael I Jordan. Perturbed iterate analysis for asynchronous stochastic optimization. *SIAM Journal on Optimization*, 27(4):2202–2229, 2017.
- Artavazd Maranjyan and Peter Richtárik. Ringleader ASGD: The first Asynchronous SGD with optimal time complexity under data heterogeneity. *arXiv preprint arXiv:2509.22860*, 2025.
- Artavazd Maranjyan, Abdurakhmon Sadiev, and Peter Richtárik. Differentially private random block coordinate descent. *arXiv preprint arXiv:2412.17054*, 2024.
- Artavazd Maranjyan, Omar Shaikh Omar, and Peter Richtárik. Mindflayer SGD: Efficient parallel SGD in the presence of heterogeneous and random worker compute times. In *The 41st Conference on Uncertainty in Artificial Intelligence*, 2025a.
- Artavazd Maranjyan, El Mehdi Saad, Peter Richtárik, and Francesco Orabona. ATA: Adaptive task allocation for efficient resource management in distributed machine learning. In *International Conference on Machine Learning*, 2025b.
- Artavazd Maranjyan, Mher Safaryan, and Peter Richtárik. GradSkip: Communication-accelerated local gradient methods with better computational complexity. *Transactions on Machine Learning Research*, 2025c. ISSN 2835-8856. URL <https://openreview.net/forum?id=6R3fRqFfhn>.
- Artavazd Maranjyan, Alexander Tyurin, and Peter Richtárik. Ringmaster ASGD: The first Asynchronous SGD with optimal time complexity. In *International Conference on Machine Learning*, 2025d.
- Andreas Maurer and Massimiliano Pontil. Concentration inequalities under sub-Gaussian and sub-exponential conditions. *Advances in Neural Information Processing Systems*, 34:7588–7597, 2021.

José Maurício, Inês Domingues, and Jorge Bernardino. Comparing vision transformers and convolutional neural networks for image classification: A literature review. *Applied Sciences*, 13(9):5521, 2023.

Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial Intelligence and Statistics*, pp. 1273–1282. PMLR, 2017.

H Brendan McMahan, Eider Moore, Daniel Ramage, and Blaise Agüera y Arcas. Federated learning of deep networks using model averaging. *arXiv preprint arXiv:1602.05629*, 2:2, 2016.

Konstantin Mishchenko, Francis Bach, Mathieu Even, and Blake E Woodworth. Asynchronous SGD beats minibatch SGD under arbitrary delays. *Advances in Neural Information Processing Systems*, 35:420–433, 2022a.

Konstantin Mishchenko, Grigory Malinovsky, Sebastian Stich, and Peter Richtarik. ProxSkip: Yes! Local gradient steps provably lead to communication acceleration! Finally! In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 15750–15769. PMLR, 17–23 Jul 2022b. URL <https://proceedings.mlr.press/v162/mishchenko22b.html>.

Ioannis Mitliagkas, Ce Zhang, Stefan Hadjis, and Christopher Ré. Asynchrony begets momentum, with an application to deep learning. In *2016 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pp. 997–1004. IEEE, 2016.

Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostafa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, et al. Efficient large-scale language model training on GPU clusters using Megatron-LM. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–15, 2021.

Yurii Nesterov. *Lectures on Convex Optimization*, volume 137. Springer, 2018.

John Nguyen, Kshitiz Malik, Hongyuan Zhan, Ashkan Yousefpour, Mike Rabbat, Mani Malek, and Dzmitry Huba. Federated learning with buffered asynchronous aggregation. In *International Conference on Artificial Intelligence and Statistics*, pp. 3581–3607. PMLR, 2022.

Lam Nguyen, Phuong Ha Nguyen, Marten Dijk, Peter Richtárik, Katya Scheinberg, and Martin Takáć. SGD and Hogwild! convergence without the bounded gradients assumption. In *International Conference on Machine Learning*, pp. 3750–3758. PMLR, 2018.

James O'Donnell and Casey Crownhart. We did the math on AI's energy footprint. here's the story you haven't heard. *MIT Technology Review*, May 2025. URL <https://www.technologyreview.com/2025/05/20/1116852/ai-energy-footprint/>. Accessed: 2025-10-18.

Xinghao Pan, Maximilian Lam, Stephen Tu, Dimitris Papailiopoulos, Ce Zhang, Michael I Jordan, Kannan Ramchandran, and Christopher Ré. Cyclades: Conflict-free asynchronous machine learning. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016. URL https://proceedings.neurips.cc/paper_files/paper/2016/file/28e209b61a52482a0ae1cb9f5959c792-Paper.pdf.

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

Reenu Rajpoot, Mahesh Gour, Sweta Jain, and Vijay Bhaskar Semwal. Integrated ensemble CNN and explainable AI for COVID-19 diagnosis from CT scan and X-ray images. *Scientific Reports*, 14(1):24985, 2024.

Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. HOGWILD!: A lock-free approach to parallelizing stochastic gradient descent. *Advances in Neural Information Processing Systems*, 24, 2011.

Herbert Robbins and Sutton Monro. A stochastic approximation method. *Annals of Mathematical Statistics*, 22(3):400–407, 1951.

Nicolas Roux, Mark Schmidt, and Francis Bach. A stochastic gradient method with an exponential convergence rate for finite training sets. *Advances in Neural Information Processing Systems*, 25, 2012.

Kevin Scaman, Francis Bach, Sébastien Bubeck, Yin Tat Lee, and Laurent Massoulié. Optimal algorithms for smooth and strongly convex distributed optimization in networks. In *International Conference on Machine Learning*, pp. 3027–3036. PMLR, 2017.

- Mark Schmidt, Nicolas Le Roux, and Francis Bach. Minimizing finite sums with the stochastic average gradient. *Mathematical Programming*, 162:83–112, 2017.
- Mohammad Shoeybi, Mostafa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-LM: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.
- Sebastian U Stich and Sai Praneeth Karimireddy. The error-feedback framework: SGD with delayed gradients. *Journal of Machine Learning Research*, 21(237):1–36, 2020.
- Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for modern deep learning research. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 13693–13696, 2020.
- Alysa Ziying Tan, Han Yu, Lizhen Cui, and Qiang Yang. Towards personalized federated learning. *IEEE Transactions on Neural Networks and Learning Systems*, 34(12):9587–9603, 2022.
- Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.
- John Tsitsiklis, Dimitri Bertsekas, and Michael Athans. Distributed asynchronous deterministic and stochastic gradient optimization algorithms. *IEEE Transactions on Automatic Control*, 31(9):803–812, 1986.
- Alexander Tyurin. Tight time complexities in parallel stochastic optimization with arbitrary computation dynamics. *arXiv preprint arXiv:2408.04929*, 2024.
- Alexander Tyurin and Peter Richtárik. Optimal time complexities of parallel stochastic optimization methods under a fixed computation model. *Advances in Neural Information Processing Systems*, 36, 2024.
- Alexander Tyurin and Peter Richtárik. On the optimal time complexities in decentralized stochastic asynchronous optimization. *Advances in Neural Information Processing Systems*, 37, 2024.
- Alexander Tyurin and Danil Sivtsov. Birch SGD: A tree graph framework for local and asynchronous SGD methods. *arXiv preprint arXiv:2505.09218*, 2025.

Alexander Tyurin, Kaja Gruntkowska, and Peter Richtárik. Freya PAGE: First optimal time complexity for large-scale nonconvex finite-sum optimization with heterogeneous asynchronous computations. *Advances in Neural Information Processing Systems*, 37, 2024a.

Alexander Tyurin, Marta Pozzi, Ivan Ilin, and Peter Richtárik. Shadowheart SGD: Distributed asynchronous SGD with optimal time complexity under arbitrary computation and communication heterogeneity. *Advances in Neural Information Processing Systems*, 37, 2024b.

N Denizcan Vanli, Mert Gurbuzbalaban, and Asuman Ozdaglar. Global convergence rate of proximal incremental aggregated gradient methods. *SIAM Journal on Optimization*, 28(2):1282–1300, 2018.

Zhongwei Wan, Xin Wang, Che Liu, Samiul Alam, Yu Zheng, Jiachen Liu, Zhongnan Qu, Shen Yan, Yi Zhu, Quanlu Zhang, Mosharaf Chowdhury, and Mi Zhang. Efficient large language models: A survey. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL <https://openreview.net/forum?id=bsCCJHb08A>. Survey Certification.

Qiyuan Wang, Qianqian Yang, Shibo He, Zhiguo Shi, and Jiming Chen. AsyncFedED: Asynchronous federated learning with euclidean distance based adaptive weight aggregation. *arXiv preprint arXiv:2205.13797*, 2022a.

Siwei Wang and Wei Chen. Thompson sampling for combinatorial semi-bandits. In *International Conference on Machine Learning*, pp. 5114–5122. PMLR, 2018.

Xiaolu Wang, Zijian Li, Shi Jin, and Jun Zhang. Achieving linear speedup in asynchronous federated learning with heterogeneous clients. *IEEE Transactions on Mobile Computing*, 2024.

Xiaolu Wang, Yuchang Sun, Hoi To Wai, and Jun Zhang. Incremental aggregated asynchronous SGD for arbitrarily heterogeneous data, 2025. URL <https://openreview.net/forum?id=m3x4kDbYAK>.

Zhongyu Wang, Zhaoyang Zhang, Yuqing Tian, Qianqian Yang, Hangguan Shan, Wei Wang, and Tony QS Quek. Asynchronous federated learning over wireless communication networks. *IEEE Transactions on Wireless Communications*, 21(9):6961–6978, 2022b.

Blake E Woodworth, Jialei Wang, Adam Smith, Brendan McMahan, and Nati Srebro. Graph oracle models, lower bounds, and gaps for parallel stochastic optimization. *Advances in Neural Information Processing Systems*, 31, 2018.

- Xuyang Wu, Sindri Magnusson, Hamid Reza Feyzmahdavian, and Mikael Johansson. Delay-adaptive step-sizes for asynchronous learning. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 24093–24113. PMLR, 17–23 Jul 2022. URL <https://proceedings.mlr.press/v162/wu22g.html>.
- Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.
- Cong Xie, Sanmi Koyejo, and Indranil Gupta. Asynchronous federated optimization. *arXiv preprint arXiv:1903.03934*, 2019.
- Bowen Yang, Jian Zhang, Jonathan Li, Christopher Ré, Christopher R Aberger, and Christopher De Sa. Pipemare: Asynchronous pipeline parallel dnn training. *arXiv preprint arXiv:1910.05124*, 2019.
- Mikhail Yurochkin, Mayank Agarwal, Soumya Ghosh, Kristjan Greenewald, Nghia Hoang, and Yasaman Khazaeni. Bayesian nonparametric federated learning of neural networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 7252–7261. PMLR, 09–15 Jun 2019. URL <https://proceedings.mlr.press/v97/yurochkin19a.html>.
- Feilong Zhang, Xianming Liu, Shiyi Lin, Gang Wu, Xiong Zhou, Junjun Jiang, and Xiangyang Ji. No one idles: Efficient heterogeneous federated learning with parallel edge and server computation. In *International Conference on Machine Learning*, pp. 41399–41413. PMLR, 2023.
- Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. Deep learning based recommender system: A survey and new perspectives. *ACM computing surveys (CSUR)*, 52(1):1–38, 2019.
- Shen-Yi Zhao and Wu-Jun Li. Fast asynchronous parallel stochastic gradient descent: A lock-free approach with convergence guarantee. volume 30, 2016.
- Yue Zhao, Meng Li, Liangzhen Lai, Naveen Suda, Damon Civin, and Vikas Chandra. Federated learning with non-iid data. *arXiv preprint arXiv:1806.00582*, 2018.

Shuxin Zheng, Qi Meng, Taifeng Wang, Wei Chen, Nenghai Yu, Zhi-Ming Ma, and Tie-Yan Liu. Asynchronous stochastic gradient descent with delay compensation. In Doina Precup and Yee Whye Teh (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 4120–4129. PMLR, 06–11 Aug 2017. URL <https://proceedings.mlr.press/v70/zheng17b.html>.

Kaiwen Zhou, Fanhua Shang, and James Cheng. A simple stochastic variance reduced algorithm with fast convergence rates. In *International Conference on Machine Learning*, pp. 5980–5989. PMLR, 2018.

APPENDICES

Appendix A

Appendix for Chapter 2

A.1 Proof of Lemma 2.4.2

To analyze the time complexity of Ringmaster ASGD, we first establish a bound on the time needed to perform a fixed number of updates. For completeness, we restate the lemma below before providing its proof.

Lemma 2.4.2. Let workers' computation times satisfy the *fixed computation model* (2.1). Let R be the delay threshold of Algorithm 2.3.1 or Algorithm 2.3.2. The time required to complete any R consecutive iterates updates of Algorithm 2.3.1 or Algorithm 2.3.2 is at most

$$t(R) := 2 \min_{m \in [n]} \left(\frac{1}{m} \sum_{i=1}^m \frac{1}{\tau_i} \right)^{-1} \left(1 + \frac{R}{m} \right). \quad (2.6)$$

Proof. We now focus on Algorithm 2.3.2. Let us consider a simplified notation of $t(R)$:

$$t := t(R) = 2 \min_{m \in [n]} \left(\sum_{i=1}^m \frac{1}{\tau_i} \right)^{-1} (R + m) = 2 \min_{m \in [n]} \left(\frac{1}{m} \sum_{i=1}^m \frac{1}{\tau_i} \right)^{-1} \left(1 + \frac{R}{m} \right).$$

Let us fix any iteration k , and consider the consecutive iterations from k to $k + R - 1$. By the design of Algorithm 2.3.2, any worker will be stopped at most one time, meaning that at most one stochastic gradient will be ignored from the worker. After that, the delays of the next stochastic gradients will not exceed $R - 1$ during these iterations. As soon as some worker finishes calculating a stochastic gradient, it immediately starts computing a new stochastic gradient.

Using a proof by contradiction, assume that Algorithm 2.3.2 will not be able to finish iteration $k + R - 1$ by the time t . At the same time, by the time t , all workers will calculate at least

$$\sum_{i=1}^n \max \left\{ \left\lfloor \frac{t}{\tau_i} \right\rfloor - 1, 0 \right\} \quad (\text{A.1})$$

stochastic gradients with delays less than R because $\lfloor t/\tau_i \rfloor$ is the number of stochastic gradients that worker i can calculate after t seconds. We subtract 1 to account for the fact that one stochastic gradient with a large delay can be

ignored. Let us take an index

$$j^* = \arg \min_{m \in [n]} \left(\sum_{i=1}^m \frac{1}{\tau_i} \right)^{-1} (R + m).$$

Since $\lfloor x \rfloor \geq x - 1$ for all $x \geq 0$, we get

$$\begin{aligned} \sum_{i=1}^n \max \left\{ \left\lfloor \frac{t}{\tau_i} \right\rfloor - 1, 0 \right\} &\geq \sum_{i=1}^{j^*} \max \left\{ \left\lfloor \frac{t}{\tau_i} \right\rfloor - 1, 0 \right\} \geq \sum_{i=1}^{j^*} \left(\left\lfloor \frac{t}{\tau_i} \right\rfloor - 1 \right) \\ &\geq \sum_{i=1}^{j^*} \frac{t}{\tau_i} - 2j^* \\ &= 2 \left(\sum_{i=1}^{j^*} \frac{1}{\tau_i} \right) \left(\left(\sum_{i=1}^{j^*} \frac{1}{\tau_i} \right)^{-1} (R + j^*) \right) - 2j^* \\ &= 2R + 2j^* - 2j^* \geq R. \end{aligned}$$

We can conclude that by the time (2.6), the algorithm will calculate R stochastic gradients with delays less than R and finish iteration $k + R - 1$, which contradicts the assumption.

The proof for Algorithm 2.3.1 is essentially the same. In Algorithm 2.3.1, one stochastic gradient with a large delay for each worker will be ignored, and all other stochastic will be used in the optimization process. \square

A.2 Proof of Lemma 2.5.1

To handle the dynamic setting where worker speeds vary over time, we now prove an analogue of Lemma 2.4.2 under the universal computation model.

Lemma 2.5.1. Let the workers' computation times satisfy the *universal computation model*. Let R be the delay threshold of Algorithm 2.3.1 or Algorithm 2.3.2. Assume that some iteration starts at time T^0 . Starting from this iteration, the R consecutive iterate updates of Algorithm 2.3.1 or Algorithm 2.3.2 will be performed before the time

$$T(R, T^0) := \min \left\{ T \geq 0 : \sum_{i=1}^n \left[\frac{1}{4} \int_{T^0}^T v_i(\tau) d\tau \right] \geq R \right\}.$$

Proof. Let

$$T := T(R, T^0) = \min \left\{ T \geq 0 : \sum_{i=1}^n \left[\frac{1}{4} \int_{T^0}^T v_i(\tau) d\tau \right] \geq R \right\}.$$

At the beginning, this proof follows the proof of Lemma 2.4.2 up to (A.1). Here we also use a proof by contradiction and assume that Algorithm 2.3.2 will not be able to do R consecutive iterative updates by the time T .

Instead of (A.1), all workers will calculate at least

$$N := \sum_{i=1}^n \max \left\{ \left\lfloor \int_{T^0}^T v_i(\tau) d\tau \right\rfloor - 1, 0 \right\}$$

stochastic gradients by time T because, due to (2.11), $\left\lfloor \int_{T^0}^T v_i(\tau) d\tau \right\rfloor$ is the number of stochastic gradients that worker i will calculate in the interval $[T^0, T]$. We define $V_i(T) := \int_{T^0}^T v_i(\tau) d\tau$. Thus,

$$N = \sum_{i=1}^n \max \{ \lfloor V_i(T) \rfloor - 1, 0 \} \text{ and } T = \min \left\{ T \geq 0 : \sum_{i=1}^n \left\lfloor \frac{V_i(T)}{4} \right\rfloor \geq R \right\}.$$

Let us additionally define

$$S := \left\{ i \in [n] : \frac{V_i(T)}{4} \geq 1 \right\}. \quad (\text{A.2})$$

Note that

$$\sum_{i=1}^n \left\lfloor \frac{V_i(T)}{4} \right\rfloor = \sum_{i \in S} \left\lfloor \frac{V_i(T)}{4} \right\rfloor, \quad (\text{A.3})$$

since $\lfloor V_i(T)/4 \rfloor = 0$ for all $i \notin S$. Using simple algebra, we get

$$\begin{aligned} N &= \sum_{i=1}^n \max \{ \lfloor V_i(T) \rfloor - 1, 0 \} \geq \sum_{i \in S} \max \{ \lfloor V_i(T) \rfloor - 1, 0 \} \\ &\geq \sum_{i \in S} \lfloor V_i(T) \rfloor - |S| \geq \sum_{i \in S} V_i(T) - 2|S|, \end{aligned}$$

where the last inequality due to $\lfloor x \rfloor \geq x - 1$ for all $x \in \mathbb{R}$. Next, using (A.2), we have

$$\begin{aligned} N &\geq \frac{1}{2} \sum_{i \in S} V_i(T) + \frac{1}{2} \sum_{i \in S} 4 - 2|S| = \sum_{i \in S} \frac{V_i(T)}{2} \\ &\geq \sum_{i \in S} \left\lfloor \frac{V_i(T)}{4} \right\rfloor \stackrel{(\text{A.3})}{=} \sum_{i=1}^n \left\lfloor \frac{V_i(T)}{4} \right\rfloor \geq R, \end{aligned}$$

where the last inequality due to the definition of T . As in the proof of Lemma 2.4.2, we can conclude that by the time $T = T(R, T^0)$, the algorithm will calculate R stochastic gradients with delays less than R and finish iteration $k + R - 1$, which contradicts the assumption. \square

A.3 Proof of Theorem 2.4.1

We restate below the result that bounds the number of iterations required by Ringmaster ASGD to reach the desired level of stationarity, and then provide its proof.

Theorem 2.4.1. Under Assumptions 2.1.1, 2.1.2, and 2.1.3, let the stepsize in Ringmaster ASGD (Algorithm 2.3.1 or Algorithm 2.3.2) be

$$\gamma = \min \left\{ \frac{1}{2RL}, \frac{\varepsilon}{4L\sigma^2} \right\}.$$

Then, the following holds

$$\frac{1}{K+1} \sum_{k=0}^K \mathbb{E} \left[\|\nabla f(x^k)\|^2 \right] \leq \varepsilon,$$

for

$$K \geq \frac{8RL\Delta}{\epsilon} + \frac{16\sigma^2 L\Delta}{\epsilon^2}, \quad (2.5)$$

where $R \in \{1, 2, \dots\}$ is an arbitrary delay threshold.

We adopt the proof strategy of Koloskova et al. (2022) and rely on the following two lemmas.

Lemma A.3.1 (Descent Lemma; Proof in Appendix A.3.1). Under Assumptions 2.1.1 and 2.1.3, if the stepsize in Ringmaster ASGD (Algorithm 2.3.1 or Algorithm 2.3.2) satisfies $\gamma \leq 1/(2L)$, the following inequality holds:

$$\begin{aligned} \mathbb{E}_{k+1} [f(x^{k+1})] &\leq f(x^k) - \frac{\gamma}{2} \|\nabla f(x^k)\|^2 - \frac{\gamma}{4} \|\nabla f(x^{k-\delta^k})\|^2 \\ &\quad + \frac{\gamma L^2}{2} \|x^k - x^{k-\delta^k}\|^2 + \frac{\gamma^2 L}{2} \sigma^2, \end{aligned}$$

where $\mathbb{E}_{k+1} [\cdot]$ represents the expectation conditioned on all randomness up to iteration k .

Lemma A.3.2 (Residual Estimation; Proof in Appendix A.3.2). Under Assumptions 2.1.1 and 2.1.3, the iterates of Ringmaster ASGD (Algorithm 2.3.1 or Algorithm 2.3.2) with stepsize $\gamma \leq 1/(2RL)$ satisfy the following bound:

$$\frac{1}{K+1} \sum_{k=0}^K \mathbb{E} \left[\|x^k - x^{k-\delta^k}\|^2 \right] \leq \frac{1}{2L^2(K+1)} \sum_{k=0}^K \mathbb{E} \left[\|\nabla f(x^{k-\delta^k})\|^2 \right] + \frac{\gamma}{L} \sigma^2.$$

With these results, we are now ready to prove Theorem 2.4.1.

Proof of Theorem 2.4.1. We begin by averaging over $K+1$ iterations and dividing by γ in the inequality from Lemma A.3.1:

$$\begin{aligned} \frac{1}{K+1} \sum_{k=0}^K \left(\frac{1}{2} \mathbb{E} \left[\|\nabla f(x^k)\|^2 \right] + \frac{1}{4} \mathbb{E} \left[\|\nabla f(x^{k-\delta^k})\|^2 \right] \right) &\leq \frac{\Delta}{\gamma(K+1)} + \frac{\gamma L}{2} \sigma^2 \\ &\quad + \frac{1}{K+1} \frac{L^2}{2} \sum_{k=0}^K \mathbb{E} \left[\|x^k - x^{k-\delta^k}\|^2 \right]. \end{aligned}$$

Next, applying Lemma A.3.2 to the last term, we have:

$$\begin{aligned} \frac{1}{K+1} \sum_{k=0}^K \left(\frac{1}{2} \mathbb{E} [\|\nabla f(x^k)\|^2] + \frac{1}{4} \mathbb{E} [\|\nabla f(x^{k-\delta^k})\|^2] \right) &\leq \frac{\Delta}{\gamma(K+1)} + \frac{\gamma L}{2} \sigma^2 \\ &+ \frac{1}{4(K+1)} \sum_{k=0}^K \mathbb{E} [\|\nabla f(x^{k-\delta^k})\|^2] + \frac{\gamma L}{2} \sigma^2. \end{aligned}$$

Simplifying further, we obtain:

$$\frac{1}{K+1} \sum_{k=0}^K \mathbb{E} [\|\nabla f(x^k)\|^2] \leq \frac{2\Delta}{\gamma(K+1)} + 2\gamma L \sigma^2.$$

Now, we choose the stepsize γ as

$$\gamma = \min \left\{ \frac{1}{2RL}, \frac{\varepsilon}{4L\sigma^2} \right\} \leq \frac{1}{2RL}.$$

With this choice of γ , it remains to choose

$$K \geq \frac{8\Delta RL}{\epsilon} + \frac{16\Delta L \sigma^2}{\epsilon^2}$$

to ensure

$$\frac{1}{K+1} \sum_{k=0}^K \mathbb{E} [\|\nabla f(x^k)\|^2] \leq \epsilon.$$

This completes the proof. \square

A.3.1 Proof of Lemma A.3.1

We restate below the descent lemma used in the convergence analysis of Ringmaster ASGD and then provide its proof.

Lemma A.3.1 (Descent Lemma). Under Assumptions 2.1.1 and 2.1.3, if the stepsize in Ringmaster ASGD (Algorithm 2.3.1 or Algorithm 2.3.2) satisfies $\gamma \leq 1/2L$, the following inequality holds:

$$\begin{aligned} \mathbb{E}_{k+1} [f(x^{k+1})] &\leq f(x^k) - \frac{\gamma}{2} \|\nabla f(x^k)\|^2 - \frac{\gamma}{4} \|\nabla f(x^{k-\delta^k})\|^2 \\ &+ \frac{\gamma L^2}{2} \|x^k - x^{k-\delta^k}\|^2 + \frac{\gamma^2 L}{2} \sigma^2, \end{aligned}$$

where $\mathbb{E}_{k+1} [\cdot]$ represents the expectation conditioned on all randomness up to iteration k .

Proof. Assume that we get a stochastic gradient from the worker with index i_k when calculating x^{k+1} . Since the function f is L -smooth (Assumption 2.1.1), we

have (Nesterov, 2018):

$$\begin{aligned}\mathbb{E}_{k+1}[f(x^{k+1})] &\leq f(x^k) - \gamma \underbrace{\mathbb{E}_{k+1}[\langle \nabla f(x^k), \nabla f(x^{k-\delta^k}; \xi_{i_k}^{k-\delta^k}) \rangle]}_{=:t_1} \\ &\quad + \underbrace{\frac{L}{2}\gamma^2 \mathbb{E}_{k+1}[\|\nabla f(x^{k-\delta^k}; \xi_{i_k}^{k-\delta^k})\|^2]}_{=:t_2}.\end{aligned}$$

Using Assumption 2.1.3, we estimate the second term as

$$\begin{aligned}t_1 &= \langle \nabla f(x^k), \nabla f(x^{k-\delta^k}) \rangle \\ &= \frac{1}{2} \left[\|\nabla f(x^k)\|^2 + \|\nabla f(x^{k-\delta^k})\|^2 - \|\nabla f(x^k) - \nabla f(x^{k-\delta^k})\|^2 \right].\end{aligned}$$

Using the variance decomposition equality and Assumption 2.1.3, we get

$$\begin{aligned}t_2 &= \mathbb{E}_{k+1} \left[\|\nabla f(x^{k-\delta^k}; \xi_{i_k}^{k-\delta^k}) - \nabla f(x^{k-\delta^k})\|^2 \right] + \|\nabla f(x^{k-\delta^k})\|^2 \\ &\leq \sigma^2 + \|\nabla f(x^{k-\delta^k})\|^2.\end{aligned}$$

Combining the results for t_1 and t_2 , and using L -smoothness to bound $\|\nabla f(x^k) - \nabla f(x^{k-\delta^k})\|^2$, we get

$$\begin{aligned}\mathbb{E}_{k+1}[f(x^{k+1})] &\leq f(x^k) - \frac{\gamma}{2} \|\nabla f(x^k)\|^2 - \frac{\gamma}{2}(1 - \gamma L) \|\nabla f(x^{k-\delta^k})\|^2 \\ &\quad + \frac{\gamma L^2}{2} \|x^k - x^{k-\delta^k}\|^2 + \frac{\gamma^2 L}{2} \sigma^2.\end{aligned}$$

Finally, applying the condition $\gamma \leq 1/2L$ completes the proof. \square

A.3.2 Proof of Lemma A.3.2

We restate below the lemma that bounds the expected residual between consecutive iterates of Ringmaster ASGD and then provide its proof.

Lemma A.3.2 (Residual Estimation). Under Assumptions 2.1.1 and 2.1.3, the iterates of Ringmaster ASGD (Algorithm 2.3.1 or Algorithm 2.3.2) with stepsize $\gamma \leq 1/2RL$ satisfy the following bound:

$$\frac{1}{K+1} \sum_{k=0}^K \mathbb{E} \left[\|x^k - x^{k-\delta^k}\|^2 \right] \leq \frac{1}{2L^2(K+1)} \sum_{k=0}^K \mathbb{E} \left[\|\nabla f(x^{k-\delta^k})\|^2 \right] + \frac{\gamma}{L} \sigma^2.$$

Proof. Assume that we get a stochastic gradient from the worker with index i_k when calculating x^{k+1} . We begin by expanding the difference and applying the

tower property, Assumption 2.1.3, Young's inequality, and Jensen's inequality:

$$\begin{aligned}
\mathbb{E} \left[\|x^k - x^{k-\delta^k}\|^2 \right] &= \mathbb{E} \left[\left\| \sum_{j=k-\delta^k}^{k-1} \gamma \nabla f(x^{j-\delta_j}; \xi_{i_j}^{j-\delta_j}) \right\|^2 \right] \\
&\leq 2\mathbb{E} \left[\left\| \sum_{j=k-\delta^k}^{k-1} \gamma \nabla f(x^{j-\delta_j}) \right\|^2 \right] \\
&\quad + 2\mathbb{E} \left[\left\| \sum_{j=k-\delta^k}^{k-1} \gamma \left(\nabla f(x^{j-\delta_j}; \xi_{i_j}^{j-\delta_j}) - \nabla f(x^{j-\delta_j}) \right) \right\|^2 \right] \\
&\leq 2\mathbb{E} \left[\left\| \gamma \sum_{j=k-\delta^k}^{k-1} \nabla f(x^{j-\delta_j}) \right\|^2 \right] + 2\delta^k \gamma^2 \sigma^2 \\
&\leq 2\delta^k \gamma^2 \sum_{j=k-\delta^k}^{k-1} \mathbb{E} \left[\|\nabla f(x^{j-\delta_j})\|^2 \right] + 2\delta^k \gamma^2 \sigma^2.
\end{aligned}$$

Using that $\gamma \leq 1/2RL$ and $\delta^k \leq R - 1$ (by the design), we obtain:

$$\mathbb{E} \left[\|x^k - x^{k-\delta^k}\|^2 \right] \leq \frac{1}{2L^2R} \sum_{j=k-\delta^k}^{k-1} \mathbb{E} \left[\|\nabla f(x^{j-\delta_j})\|^2 \right] + \frac{\gamma}{L} \sigma^2.$$

Next, summing over all iterations $k = 0, \dots, K$, we get:

$$\sum_{k=0}^K \mathbb{E} \left[\|x^k - x^{k-\delta^k}\|^2 \right] \leq \frac{1}{2L^2R} \sum_{k=0}^K \sum_{j=k-\delta^k}^{k-1} \mathbb{E} \left[\|\nabla f(x^{j-\delta_j})\|^2 \right] + (K+1) \frac{\gamma}{L} \sigma^2.$$

Observe that each squared norm $\|\nabla f(x^{j-\delta_j})\|^2$ in the right-hand sums appears at most R times due to the algorithms' design. Specifically, $\delta^k \leq R - 1$ for all $k \geq 0$, ensuring no more than R squared norms appear in the sums. Therefore:

$$\sum_{k=0}^K \mathbb{E} \left[\|x^k - x^{k-\delta^k}\|^2 \right] \leq \frac{1}{2L^2} \sum_{k=0}^K \mathbb{E} \left[\|\nabla f(x^{k-\delta^k})\|^2 \right] + (K+1) \frac{\gamma}{L} \sigma^2.$$

Finally, dividing the inequality by $K+1$ completes the proof. \square

A.4 Proof of Theorem 2.5.2

We restate below the theorem establishing the convergence of Ringmaster ASGD under the universal computation model and then provide its proof.

Theorem 2.5.2. Let Assumptions 2.1.1, 2.1.2, and 2.1.3 hold. Let the stepsize

in Ringmaster ASGD (Algorithm 2.3.1 or Algorithm 2.3.2) be

$$\gamma = \min \left\{ \frac{1}{2RL}, \frac{\varepsilon}{4L\sigma^2} \right\},$$

and delay threshold

$$R = \max \left\{ 1, \left\lceil \frac{\sigma^2}{\varepsilon} \right\rceil \right\}.$$

Then, under the *universal computation model*, Ringmaster ASGD finds an ε -stationary point after at most $T^{\bar{K}}$ seconds, where

$$\bar{K} := \left\lceil \frac{48L\Delta}{\epsilon} \right\rceil$$

and $T^{\bar{K}}$ is the \bar{K} -th element of the following recursively defined sequence:

$$T^k := \min \left\{ T \geq 0 : \sum_{i=1}^n \left\lceil \frac{1}{4} \int_{T^{k-1}}^T p_i(\tau) d\tau \right\rceil \geq R \right\}$$

for all $k \geq 1$ and $T^0 = 0$.

Proof. From Theorem 2.4.1, the iteration complexity of Ringmaster ASGD is

$$K = \left\lceil \frac{8RL\Delta}{\epsilon} + \frac{16\sigma^2 L\Delta}{\epsilon^2} \right\rceil. \quad (\text{A.4})$$

Without loss of generality, we assume that $L\Delta > \varepsilon/2$.¹ Thus,

$$\left\lceil \frac{8RL\Delta}{\epsilon} + \frac{16\sigma^2 L\Delta}{\epsilon^2} \right\rceil \leq \frac{16RL\Delta}{\epsilon} + \frac{32\sigma^2 L\Delta}{\epsilon^2}$$

and

$$K \leq R \times \left\lceil \frac{K}{R} \right\rceil = R \times \left\lceil \frac{16L\Delta}{\epsilon} + \frac{32\sigma^2 L\Delta}{R\epsilon^2} \right\rceil.$$

Using the choice of R , we get

$$K \leq R \times \left\lceil \frac{48L\Delta}{\epsilon} \right\rceil.$$

In total, the algorithms will require $\lceil 48L\Delta/\epsilon \rceil$ by R consecutive updates of x^k to find an ε -stationary point. Let us define $\bar{K} := \lceil 48L\Delta/\epsilon \rceil$. Using Lemma 2.5.1, we know that Ringmaster ASGD requires at most

$$T^1 := T(R, 0) = \min \left\{ T \geq 0 : \sum_{i=1}^n \left\lceil \frac{1}{4} \int_0^T v_i(\tau) d\tau \right\rceil \geq R \right\}$$

seconds to finish the first R consecutive updates of the iterates. Since the algo-

¹Otherwise, using L -smoothness, $\|\nabla f(x^0)\|^2 \leq 2L\Delta \leq \varepsilon$, and the initial point is an ε -stationary point.

rithms will finish the *first* R consecutive updates after at most T^1 seconds, they will start the iteration $R + 1$ before time T^1 . Thus, using Lemma 2.5.1 again, they will require at most

$$T^2 := T(R, T^1) = \min \left\{ T \geq 0 : \sum_{i=1}^n \left[\frac{1}{4} \int_{T^1}^T v_i(\tau) d\tau \right] \geq R \right\}$$

seconds to finish the first $2 \times R$ consecutive updates. Using the same reasoning, they will finish the first $\lceil 48L\Delta/\epsilon \rceil \times R$ consecutive updates after at most

$$T_{\bar{K}} := T(R, T_{\bar{K}-1}) = \min \left\{ T \geq 0 : \sum_{i=1}^n \left[\frac{1}{4} \int_{T_{\bar{K}-1}}^T v_i(\tau) d\tau \right] \geq R \right\}$$

seconds. \square

A.5 Derivations for the Example from Section 2.2

Let $\tau_i = \sqrt{i}$ for all $i \in [n]$, then

$$T_R = \Theta \left(\min_{m \in [n]} \left(\frac{1}{m} \sum_{i=1}^m \frac{1}{\sqrt{i}} \right)^{-1} \left(\frac{L\Delta}{\varepsilon} + \frac{\sigma^2 L\Delta}{m\varepsilon^2} \right) \right).$$

Using

$$\sum_{i=1}^m \frac{1}{\sqrt{i}} = \Theta(\sqrt{m})$$

for $m \geq 1$, we simplify the term:

$$\left(\frac{1}{m} \sum_{i=1}^m \frac{1}{\sqrt{i}} \right)^{-1} = \Theta(\sqrt{m}),$$

$$T_R = \Theta \left(\min_{m \in [n]} \sqrt{m} \left(\frac{L\Delta}{\varepsilon} + \frac{\sigma^2 L\Delta}{m\varepsilon^2} \right) \right) = \Theta \left(\min_{m \in [n]} \left(\frac{L\Delta\sqrt{m}}{\varepsilon} + \frac{\sigma^2 L\Delta}{\sqrt{m}\varepsilon^2} \right) \right).$$

The minimum is achieved when the two terms are balanced, i.e., at

$$m = \min \left\{ \left\lceil \frac{\sigma^2}{\varepsilon} \right\rceil, n \right\}.$$

Substituting this value of m , we obtain:

$$T_R = \Theta \left(\max \left[\frac{\sigma L\Delta}{\varepsilon^{3/2}}, \frac{\sigma^2 L\Delta}{\sqrt{n}\varepsilon^2} \right] \right).$$

We now consider T_A :

$$T_A = \Theta \left(\left(\frac{1}{n} \sum_{i=1}^n \frac{1}{\tau_i} \right)^{-1} \left(\frac{L\Delta}{\varepsilon} + \frac{\sigma^2 L\Delta}{n\varepsilon^2} \right) \right).$$

Using

$$\sum_{i=1}^n \frac{1}{\sqrt{i}} = \Theta(\sqrt{n})$$

for $n \geq 1$, we simplify the term:

$$\left(\frac{1}{n} \sum_{i=1}^n \frac{1}{\sqrt{i}} \right)^{-1} = \Theta(\sqrt{n}).$$

Substituting this result into T_A , we have:

$$\begin{aligned} T_A &= \Theta \left(\sqrt{n} \left(\frac{L\Delta}{\varepsilon} + \frac{\sigma^2 L\Delta}{n\varepsilon^2} \right) \right) = \Theta \left(\frac{L\Delta\sqrt{n}}{\varepsilon} + \frac{\sigma^2 L\Delta}{\sqrt{n}\varepsilon^2} \right) \\ &= \Theta \left(\max \left[\frac{L\Delta\sqrt{n}}{\varepsilon}, \frac{\sigma^2 L\Delta}{\sqrt{n}\varepsilon^2} \right] \right). \end{aligned}$$

A.6 When the Initial Point is an ε -Stationary Point

Under the assumption of L -smoothness (Assumption 2.1.1), we have:

$$f(y) \leq f(x) + \langle \nabla f(x), y - x \rangle + \frac{L}{2} \|y - x\|^2$$

for all $x, y \in \mathbb{R}^d$. Taking $y = x - \frac{1}{L} \nabla f(x)$,

$$f \left(x - \frac{1}{L} \nabla f(x) \right) \leq f(x) - \frac{1}{2L} \|\nabla f(x)\|^2.$$

Since $f \left(x - \frac{1}{L} \nabla f(x) \right) \geq f^*$ and taking $x = x^0$, we get

$$\|\nabla f(x^0)\|^2 \leq 2L\Delta.$$

Thus, if $2L\Delta \leq \varepsilon$, then $\|\nabla f(x^0)\|^2 \leq \varepsilon$.

Appendix B

Appendix for Chapter 3

B.1 Arbitrarily Changing Computation Times

In practice, the *fixed computation model* (3.2) is often not satisfied. The compute power of devices can vary over time due to temporary disconnections, hardware or network delays, fluctuations in processing capacity, or other transient effects (Maranjyan et al., 2025a).

In this section we extend our theory to the more general setting of arbitrarily varying computation times.

B.1.1 Universal Computation Model

To formalize this setting, we adopt the *universal computation model* introduced by Tyurin (2024).

For each worker $i \in [n]$, we define a *compute power* function

$$p_i : \mathbb{R}_+ \rightarrow \mathbb{R}_+,$$

assumed nonnegative and continuous almost everywhere (countably many jumps allowed). For any $T_2 \geq T_1 \geq 0$, the number of stochastic gradients *completed* by worker i on $[T_1, T_2]$ is

$$\#\text{gradients in } [T_1, T_2] = \left\lfloor \int_{T_1}^{T_2} p_i(t) dt \right\rfloor.$$

Here, $p_i(t)$ models the worker's time-varying computational ability: smaller values over an interval yield fewer completed gradients, and larger values yield more.

For instance, if worker i remains idle for the first T seconds and then becomes active, this corresponds to $p_i(t) = 0$ for $t \leq T$ and $p_i(t) > 0$ for $t > T$. More generally, $p_i(t)$ may follow periodic or irregular patterns, leading to bursts of activity, pauses, or chaotic changes in compute power. The process $p_i(t)$ may even be random, and all results hold conditional on the realized sample paths of $\{p_i\}$.

The *universal computation model* reduces to the *fixed computation model* (3.2) when $p_i(t) = 1/\tau_i$ for all $t \geq 0$ and $i \in [n]$. In this case,

$$\#\text{gradients in } [T_1, T_2] = \left\lfloor \frac{T_2 - T_1}{\tau_i} \right\rfloor,$$

meaning that worker i computes one stochastic gradient after $T_1 + \tau_i$ seconds, two gradients after $T_1 + 2\tau_i$ seconds, and so on.

B.1.2 Toward an Optimal Method

In the general setting of arbitrarily varying computation times, Algorithm 3.5.1 is not optimal. To see why, consider the following adversarial timing pattern.

Suppose there are two workers. During one gradient computation by the slower worker, the faster worker computes s gradients. Immediately afterwards, they switch roles: the previously fast worker slows down by a factor of s , while the previously slow one speeds up by the same factor. This pattern repeats each time the slower worker finishes a gradient computation.

In this setting, if we run Algorithm 3.5.1, the server waits in each Phase 1 for a single gradient from every worker. Thus, the slower worker always contributes only one gradient, and the harmonic mean of the batch sizes satisfies

$$1 \leq B^k \leq 2.$$

From Theorem 3.6.3, the iteration complexity is

$$\mathcal{O}\left(\frac{nL\Delta}{\varepsilon} \left(1 + \frac{\sigma^2}{Bn\varepsilon}\right)\right).$$

When $\sigma^2/n\varepsilon$ is much larger than B , this dependence can be highly suboptimal.

Instead, suppose the server waits until one full round of the above process completes, collecting $s+1$ gradients from each worker. Then the harmonic mean satisfies $B^k \geq s+1$, which can be arbitrarily larger than 2. Since in practice both s and $\sigma^2/n\varepsilon$ can be very large, the naive strategy of waiting for only one gradient per worker (as in Algorithm 3.5.1) cannot be optimal in the arbitrary-time setting.

B.1.3 An Optimal Method

The solution is simple and follows directly from the iteration complexity bound. From

$$\mathcal{O}\left(\frac{nL\Delta}{\varepsilon} \left(1 + \frac{\sigma^2}{Bn\varepsilon}\right)\right),$$

we see that to balance the terms it suffices to ensure

$$B \geq \frac{\sigma^2}{n\varepsilon}.$$

Accordingly, we modify the stopping condition in Phase 1 of Algorithm 3.5.1. Instead of requiring the server to receive at least one gradient from each worker, we require the stronger condition used in Malenia SGD, namely

$$\left(\frac{1}{n} \sum_{i=1}^n \frac{1}{b_i}\right)^{-1} \geq \max\left\{1, \frac{\sigma^2}{n\varepsilon}\right\}, \quad (3.3)$$

where b_i is the number of gradients received from worker i .

In the low-noise regime, where $\sigma^2/n\varepsilon \leq 1$, the condition reduces to requiring $b_i \geq 1$ for all i , so the algorithm coincides with the original Algorithm 3.5.1. In the high-noise regime, the algorithm collects more gradients in Phase 1, ensuring that B is sufficiently large for optimal convergence.

With this change, Phase 1 of our algorithm matches that of Malenia SGD. The difference lies in Phase 2: our algorithm continues to use the ongoing gradient computations from all workers to perform n updates, while Malenia SGD discards any unfinished gradients, performs a single update, and then proceeds to the next round.

The following theorem establishes the time complexity of our algorithm under the universal computation model.

Theorem B.1.1. Under Assumptions 3.3.1, 3.3.2, and 3.3.5, let the stepsize in Ringleader ASGD be

$$\gamma = \frac{1}{10nL}.$$

Then, under the *universal computation model*, Ringleader ASGD finds an ε -stationary point within at most $T^{\bar{K}}$ seconds, where

$$\bar{K} := \left\lceil \frac{160L\Delta}{\varepsilon} \right\rceil,$$

and $T^{\bar{K}}$ denotes the \bar{K} -th element of the recursively defined sequence

$$T^k = \min \left\{ T \geq 0 : \left(\frac{1}{n} \sum_{i=1}^n \left[\int_{T_{k-1}}^T p_i(t) dt \right]^{-1} \right)^{-1} \geq \max \left\{ 1, \frac{\sigma^2}{n\varepsilon} \right\} \right\},$$

for all $k \geq 1$, with initialization $T^0 = 0$.

This result matches the lower bound derived by Tyurin (2024), and therefore the proposed method is optimal.

Proof. Under the condition in (3.3), each gradient-type step of the algorithm satisfies

$$B^k = \left(\frac{1}{n} \sum_{i=1}^n \frac{1}{b_i^k} \right)^{-1} \geq \max \left\{ 1, \frac{\sigma^2}{n\varepsilon} \right\}.$$

In Theorem 3.6.3, instead of using B we can substitute any valid lower bound. Here we choose

$$B = \max \left\{ 1, \frac{\sigma^2}{n\varepsilon} \right\}.$$

With this substitution, the iteration complexity becomes

$$K = \frac{80nL\Delta}{\varepsilon}.$$

To derive the time complexity, consider the time required to perform n iterations. Each block of n updates occurs in Phase 2 following the Phase 1 gradient collection. Starting from time $T = 0$, Phase 1 ends once the accumulated number of gradients satisfies condition (3.3), which occurs at time

$$T_+^1 = \min \left\{ T \geq 0 : \left(\frac{1}{n} \sum_{i=1}^n \left[\int_0^T p_i(t) dt \right]^{-1} \right)^{-1} \geq \max \left\{ 1, \frac{\sigma^2}{n\varepsilon} \right\} \right\}.$$

After Phase 1, to complete n updates in Phase 2 we must wait for the ongoing computations to finish. This requires at most

$$T^1 = \min \left\{ T \geq 0 : \left(\frac{1}{n} \sum_{i=1}^n \left[\int_{T_+^1}^T p_i(t) dt \right]^{-1} \right)^{-1} \geq 1 \right\}.$$

Thus, the total time to complete all K iterations is bounded by

$$T^{\lceil 2K/n \rceil},$$

where the sequence $\{T^k\}_{k \geq 0}$ is defined recursively as

$$T^k = \min \left\{ T \geq 0 : \left(\frac{1}{n} \sum_{i=1}^n \left[\int_{T_{k-1}}^T p_i(t) dt \right]^{-1} \right)^{-1} \geq \max \left\{ 1, \frac{\sigma^2}{n\varepsilon} \right\} \right\}.$$

and $T^0 = 0$. \square

B.2 Auxiliary Lemmas

Here we provide proofs of lemmas omitted from the main text, along with auxiliary results that will be used later.

B.2.1 Proof of Lemma 3.3.4

We begin with a lemma relating the different smoothness constants.

Lemma 3.3.4 (Smoothness Bounds). Let L_f denote the smoothness constant of f , L_{f_i} the smoothness constant of f_i , and L the constant from Assumption 3.3.2. We have

$$L_f \leq L \leq \sqrt{\frac{1}{n} \sum_{i=1}^n L_{f_i}^2} \leq \max_{i \in [n]} L_{f_i} =: L_{\max}.$$

Moreover, if all f_i are identical, i.e., $f_i = f$ for all $i \in [n]$, then $L = L_f$.

Recall from Assumption 3.3.2 that we assumed the following generalized smoothness condition: for some constant $L > 0$ and for all $x \in \mathbb{R}^d$ and $y_1, \dots, y_n \in \mathbb{R}^d$,

$$\left\| \nabla f(x) - \frac{1}{n} \sum_{i=1}^n \nabla f_i(y_i) \right\|^2 \leq \frac{L^2}{n} \sum_{i=1}^n \|x - y_i\|^2. \quad (\text{B.1})$$

Recall that a function ϕ is called L_ϕ -smooth if

$$\|\nabla \phi(x) - \nabla \phi(y)\| \leq L_\phi \|x - y\|, \quad \forall x, y \in \mathbb{R}^d.$$

Here L_ϕ denotes the minimal such constant. We are ready to prove the lemma.

Proof. For the first inequality, take $y_1 = \dots = y_n = y$. Then (B.1) reduces to

$$\|\nabla f(x) - \nabla f(y)\|^2 \leq L^2 \|x - y\|^2,$$

so f is L -smooth. By definition of L_f as the minimal smoothness constant, this implies $L_f \leq L$.

For the second inequality, by the triangle inequality, then by the smoothness of each f_i , and finally by Cauchy–Schwarz,

$$\begin{aligned} \left\| \nabla f(x) - \frac{1}{n} \sum_{i=1}^n \nabla f_i(y_i) \right\| &\leq \frac{1}{n} \sum_{i=1}^n \|\nabla f_i(x) - \nabla f_i(y_i)\| \leq \frac{1}{n} \sum_{i=1}^n L_{f_i} \|x - y_i\| \\ &\leq \sqrt{\frac{1}{n} \sum_{i=1}^n L_{f_i}^2} \sqrt{\frac{1}{n} \sum_{i=1}^n \|x - y_i\|^2}. \end{aligned}$$

Squaring both sides shows that (B.1) holds with $L = \sqrt{\frac{1}{n} \sum_{i=1}^n L_{f_i}^2}$.

Finally, suppose all f_i are identical: $f_i \equiv f$ for all i . Then

$$\begin{aligned} \left\| \nabla f(x) - \frac{1}{n} \sum_{i=1}^n \nabla f(y_i) \right\| &\leq \frac{1}{n} \sum_{i=1}^n \|\nabla f(x) - \nabla f(y_i)\| \leq \frac{L_f}{n} \sum_{i=1}^n \|x - y_i\| \\ &\leq L_f \sqrt{\frac{1}{n} \sum_{i=1}^n \|x - y_i\|^2}, \end{aligned}$$

where the last step uses Cauchy–Schwarz. Squaring both sides yields

$$\left\| \nabla f(x) - \frac{1}{n} \sum_{i=1}^n \nabla f(y_i) \right\|^2 \leq \frac{L_f^2}{n} \sum_{i=1}^n \|x - y_i\|^2,$$

i.e., (B.1) holds with $L \leq L_f$. Combined with $L_f \leq L$, we conclude $L = L_f$. \square

B.2.2 Variance Term

The next lemma provides an upper bound on the variance of the gradient estimator used in Ringleader ASGD.

Lemma B.2.1 (Variance Bound). Under Assumption 3.3.1, the gradient estimator used in Algorithm 3.5.1 satisfies the following variance-type inequality

$$\mathbb{E} \left[\left\| \bar{g}^k - \frac{1}{n} \sum_{i=1}^n \nabla f_i \left(x^{k-\delta_i^k} \right) \right\|^2 \right] \leq \frac{\sigma^2}{B^k n}.$$

Proof. Recall that the gradient estimator is defined as

$$\bar{g}^k = \frac{1}{n} \sum_{i=1}^n \bar{g}_i^k = \frac{1}{n} \sum_{i=1}^n g_i^{k,j} = \frac{1}{n} \sum_{i=1}^n \nabla f_i \left(x^{k-\delta_i^k}; \xi_i^{k-\delta_i^k, j} \right).$$

Let \mathcal{F}^k denote the sigma-field containing all randomness up to the start of the current round, i.e., up to iteration $k - (k \bmod n)$. Conditioning on \mathcal{F}^k , the evaluation points $x^{k-\delta_i^k}$ are deterministic, and the stochastic gradients $g_i^{k,j}$ are independent across both workers i and samples j .

Using the law of total expectation and the independence of stochastic gradients, we have

$$\begin{aligned}\mathbb{E} \left[\left\| \bar{g}^k - \frac{1}{n} \sum_{i=1}^n \nabla f_i(x^{k-\delta_i^k}) \right\|^2 \right] &= \mathbb{E} \left[\mathbb{E} \left[\left\| \bar{g}^k - \frac{1}{n} \sum_{i=1}^n \nabla f_i(x^{k-\delta_i^k}) \right\|^2 \mid \mathcal{F}^k \right] \right] \\ &= \mathbb{E} \left[\frac{1}{n^2} \sum_{i=1}^n \mathbb{E} \left[\left\| \bar{g}_i^k - \nabla f_i(x^{k-\delta_i^k}) \right\|^2 \mid \mathcal{F}^k \right] \right].\end{aligned}$$

For each worker i , the conditional variance of the minibatch gradient estimator is

$$\begin{aligned}\mathbb{E} \left[\left\| \bar{g}_i^k - \nabla f_i(x^{k-\delta_i^k}) \right\|^2 \mid \mathcal{F}^k \right] &= \mathbb{E} \left[\left\| \frac{1}{b_i^k} \sum_{j=1}^{b_i^k} g_i^{k,j} - \nabla f_i(x^{k-\delta_i^k}) \right\|^2 \mid \mathcal{F}^k \right] \\ &= \frac{1}{b_i^k} \mathbb{E} \left[\left\| g_i^{k,1} - \nabla f_i(x^{k-\delta_i^k}) \right\|^2 \mid \mathcal{F}^k \right] \leq \frac{\sigma^2}{b_i^k},\end{aligned}$$

where the last inequality follows from Assumption 3.3.1.

Combining these results, we get

$$\mathbb{E} \left[\left\| \bar{g}^k - \frac{1}{n} \sum_{i=1}^n \nabla f_i(x^{k-\delta_i^k}) \right\|^2 \right] \leq \frac{1}{n^2} \sum_{i=1}^n \frac{\sigma^2}{b_i^k} = \frac{\sigma^2}{n} \frac{1}{n} \sum_{i=1}^n \frac{1}{b_i^k} = \frac{\sigma^2}{B^k n},$$

where the last equality uses the definition of the harmonic mean

$$B^k = \left(\frac{1}{n} \sum_{i=1}^n \frac{1}{b_i^k} \right)^{-1}.$$

□

B.2.3 Proof of Lemma 3.6.1

We now prove the descent lemma.

Lemma 3.6.1 (Descent Lemma). Under Assumptions 3.3.1 and 3.3.2, if the stepsize in Algorithm 3.5.1 satisfies $\gamma \leq 1/4L$, then the following inequality holds

$$\begin{aligned}\mathbb{E} [f(x^{k+1})] &\leq \mathbb{E} [f(x^k)] - \frac{\gamma}{2} \mathbb{E} [\|\nabla f(x^k)\|^2] - \frac{\gamma}{4} \mathbb{E} \left[\left\| \frac{1}{n} \sum_{i=1}^n \nabla f_i(x^{k-\delta_i^k}) \right\|^2 \right] \\ &\quad + \frac{\gamma L^2}{2n} \sum_{i=1}^n \mathbb{E} [\|x^k - x^{k-\delta_i^k}\|^2] + \frac{3\gamma^2 L \sigma^2}{2B} \\ &\quad + \gamma^2 L \sum_{\ell=k-(k \bmod n)}^{k-1} \mathbb{E} \left[\left\| \frac{1}{n} \sum_{i=1}^n \nabla f_i(x^{\ell-\delta_i^\ell}) \right\|^2 \right].\end{aligned}$$

Proof. Some proof techniques are adapted from the works of Maranjyan et al.

(2025d) and Wang et al. (2025).

From Assumption 3.3.2 and Lemma 3.3.4, we know that f is L -smooth. Therefore, the following standard inequality holds (Nesterov, 2018)

$$\mathbb{E} [f(x^{k+1})] \leq \mathbb{E} \left[f(x^k) - \gamma \langle \nabla f(x^k), \bar{g}^k \rangle + \frac{L\gamma^2}{2} \|\bar{g}^k\|^2 \right]. \quad (\text{B.2})$$

Recall that the gradient estimator is defined as

$$\bar{g}^k = \frac{1}{n} \sum_{i=1}^n \bar{g}_i^k = \frac{1}{n} \sum_{i=1}^n \frac{1}{b_i^k} \sum_{j=1}^{b_i^k} g_i^{k,j}.$$

Let \mathcal{F}^k denote the sigma-field containing all randomness up to the start of the current Phase 2, i.e., up to iteration $k - (k \bmod n)$. A key observation is that all gradients in the current gradient table were computed and received during the current round. Since these gradients were computed at points from previous iterations within the current round, we have $k - \delta_i^k \leq k - (k \bmod n)$ for all $i \in [n]$. Conditioning on \mathcal{F}^k , the points $x^{k-\delta_i^k}$ are deterministic. Therefore, we can compute the conditional expectation of the gradient estimator:

$$\mathbb{E} [\bar{g}^k \mid \mathcal{F}^k] = \frac{1}{n} \sum_{i=1}^n \frac{1}{b_i^k} \sum_{j=1}^{b_i^k} \mathbb{E} [g_i^{k,j} \mid \mathcal{F}^k] = \frac{1}{n} \sum_{i=1}^n \nabla f_i (x^{k-\delta_i^k}).$$

The last equality follows from the unbiasedness of the stochastic gradient estimator (Assumption 3.3.1).

Using this conditional expectation and the law of total expectation, we can

now simplify the inner product term in (B.2):

$$\begin{aligned}
\mathbb{E} [\langle \nabla f(x^k), \bar{g}^k \rangle] &= \mathbb{E} \left[\left\langle \nabla f(x^k), \frac{1}{n} \sum_{i=1}^n \nabla f_i(x^{k-\delta_i^k}) \right\rangle \right] \\
&\quad + \mathbb{E} \left[\left\langle \nabla f(x^k), \bar{g}^k - \frac{1}{n} \sum_{i=1}^n \nabla f_i(x^{k-\delta_i^k}) \right\rangle \right] \\
&= \mathbb{E} \left[\left\langle \nabla f(x^k), \frac{1}{n} \sum_{i=1}^n \nabla f_i(x^{k-\delta_i^k}) \right\rangle \right] \\
&\quad + \mathbb{E} \left[\left\langle \nabla f(x^k) - \nabla f(x^{k-(k \bmod n)}), \bar{g}^k - \frac{1}{n} \sum_{i=1}^n \nabla f_i(x^{k-\delta_i^k}) \right\rangle \right] \\
&\quad + \mathbb{E} \left[\left\langle \nabla f(x^{k-(k \bmod n)}), \bar{g}^k - \frac{1}{n} \sum_{i=1}^n \nabla f_i(x^{k-\delta_i^k}) \right\rangle \right] \\
&= \underbrace{\mathbb{E} \left[\left\langle \nabla f(x^k), \frac{1}{n} \sum_{i=1}^n \nabla f_i(x^{k-\delta_i^k}) \right\rangle \right]}_{T_1} \\
&\quad + \underbrace{\mathbb{E} \left[\left\langle \nabla f(x^k) - \nabla f(x^{k-(k \bmod n)}), \bar{g}^k - \frac{1}{n} \sum_{i=1}^n \nabla f_i(x^{k-\delta_i^k}) \right\rangle \right]}_{T_2}.
\end{aligned}$$

Next, using Assumption 3.3.2, we have

$$\begin{aligned}
2T_1 &= \mathbb{E} \left[2 \left\langle \nabla f(x^k), \frac{1}{n} \sum_{i=1}^n \nabla f_i(x^{k-\delta_i^k}) \right\rangle \right] \\
&= \mathbb{E} \left[\left\| \nabla f(x^k) \right\|^2 + \left\| \frac{1}{n} \sum_{i=1}^n \nabla f_i(x^{k-\delta_i^k}) \right\|^2 \right] \\
&\quad - \mathbb{E} \left[\left\| \nabla f(x^k) - \frac{1}{n} \sum_{i=1}^n \nabla f_i(x^{k-\delta_i^k}) \right\|^2 \right] \\
&\geq \mathbb{E} \left[\left\| \nabla f(x^k) \right\|^2 \right] + \mathbb{E} \left[\left\| \frac{1}{n} \sum_{i=1}^n \nabla f_i(x^{k-\delta_i^k}) \right\|^2 \right] \\
&\quad - \frac{L^2}{n} \sum_{i=1}^n \mathbb{E} \left[\left\| x^k - x^{k-\delta_i^k} \right\|^2 \right].
\end{aligned}$$

Next, we analyze T_2

$$\begin{aligned}
T_2 &= \mathbb{E} \left[\left\langle \nabla f(x^k) - \nabla f(x^{k-(k \bmod n)}), \bar{g}^k - \frac{1}{n} \sum_{i=1}^n \nabla f_i(x^{k-\delta_i^k}) \right\rangle \right] \\
&\geq -\mathbb{E} \left[\left\| \nabla f(x^k) - \nabla f(x^{k-(k \bmod n)}) \right\| \left\| \bar{g}^k - \frac{1}{n} \sum_{i=1}^n \nabla f_i(x^{k-\delta_i^k}) \right\| \right] \\
&\geq -L \mathbb{E} \left[\left\| x^k - x^{k-(k \bmod n)} \right\| \left\| \bar{g}^k - \frac{1}{n} \sum_{i=1}^n \nabla f_i(x^{k-\delta_i^k}) \right\| \right] \\
&= -L \mathbb{E} \left[\left\| \gamma \sum_{\ell=k-(k \bmod n)}^{k-1} \bar{g}^\ell \right\| \left\| \bar{g}^k - \frac{1}{n} \sum_{i=1}^n \nabla f_i(x^{k-\delta_i^k}) \right\| \right] \\
&\geq -L\gamma \sum_{\ell=k-(k \bmod n)}^{k-1} \mathbb{E} \left[\left\| \bar{g}^\ell \right\| \left\| \bar{g}^k - \frac{1}{n} \sum_{i=1}^n \nabla f_i(x^{k-\delta_i^k}) \right\| \right] \\
&\geq -L\gamma \sum_{\ell=k-(k \bmod n)}^{k-1} \frac{1}{2} \left(\mathbb{E} [\|\bar{g}^\ell\|^2] + \mathbb{E} \left[\left\| \bar{g}^k - \frac{1}{n} \sum_{i=1}^n \nabla f_i(x^{k-\delta_i^k}) \right\|^2 \right] \right) \\
&\geq -\frac{L\gamma}{2} \sum_{\ell=k-(k \bmod n)}^{k-1} \mathbb{E} [\|\bar{g}^\ell\|^2] - (k \bmod n) \frac{L\gamma\sigma^2}{2B^k n} \\
&\geq -\frac{L\gamma}{2} \sum_{\ell=k-(k \bmod n)}^{k-1} \mathbb{E} [\|\bar{g}^\ell\|^2] - \frac{L\gamma\sigma^2}{2B^k}.
\end{aligned}$$

The inequalities follow from the Cauchy-Schwarz inequality, L -smoothness of f , the triangle inequality, Young's inequality, Lemma B.2.1, and finally $(k \bmod n) \leq n-1 < n$.

It remains to bound the term $\mathbb{E} [\|\bar{g}^k\|^2]$. Using Young's inequality, we have

$$\begin{aligned}
\mathbb{E} [\|\bar{g}^k\|^2] &\leq 2\mathbb{E} \left[\left\| \frac{1}{n} \sum_{i=1}^n \nabla f_i(x^{k-\delta_i^k}) \right\|^2 \right] + 2\mathbb{E} \left[\left\| \bar{g}^k - \frac{1}{n} \sum_{i=1}^n \nabla f_i(x^{k-\delta_i^k}) \right\|^2 \right] \\
&\leq 2\mathbb{E} \left[\left\| \frac{1}{n} \sum_{i=1}^n \nabla f_i(x^{k-\delta_i^k}) \right\|^2 \right] + \frac{2\sigma^2}{B^k n},
\end{aligned}$$

where in the last step we used Lemma B.2.1.

Now, by combining all terms in (B.2), we obtain

$$\begin{aligned}
\mathbb{E} [f(x^{k+1})] &\leq \mathbb{E} [f(x^k)] - \frac{\gamma}{2} \mathbb{E} [\|\nabla f(x^k)\|^2] - \frac{\gamma}{2} \mathbb{E} \left[\left\| \frac{1}{n} \sum_{i=1}^n \nabla f_i(x^{k-\delta_i^k}) \right\|^2 \right] \\
&\quad + \frac{\gamma L^2}{2n} \sum_{i=1}^n \mathbb{E} [\|x^k - x^{k-\delta_i^k}\|^2] \\
&\quad + \frac{\gamma^2 L}{2} \sum_{\ell=k-(k \bmod n)}^{k-1} \mathbb{E} [\|\bar{g}^\ell\|^2] + \frac{\gamma^2 L \sigma^2}{2B^k} + \frac{\gamma^2 L}{2} \mathbb{E} [\|\bar{g}^k\|^2] \\
&\leq \mathbb{E} [f(x^k)] - \frac{\gamma}{2} \mathbb{E} [\|\nabla f(x^k)\|^2] - \frac{\gamma}{2} \mathbb{E} \left[\left\| \frac{1}{n} \sum_{i=1}^n \nabla f_i(x^{k-\delta_i^k}) \right\|^2 \right] \\
&\quad + \frac{\gamma L^2}{2n} \sum_{i=1}^n \mathbb{E} [\|x^k - x^{k-\delta_i^k}\|^2] \\
&\quad + \gamma^2 L \sum_{\ell=k-(k \bmod n)}^{k-1} \mathbb{E} \left[\left\| \frac{1}{n} \sum_{i=1}^n \nabla f_i(x^{\ell-\delta_i^\ell}) \right\|^2 \right] \\
&\quad + \gamma^2 L \mathbb{E} \left[\left\| \frac{1}{n} \sum_{i=1}^n \nabla f_i(x^{k-\delta_i^k}) \right\|^2 \right] \\
&\quad + \gamma^2 L \sum_{\ell=k-(k \bmod n)}^{k-1} \frac{\sigma^2}{B^\ell n} + \frac{\gamma^2 L \sigma^2}{2B^k} + \frac{\gamma^2 L \sigma^2}{B^k n}.
\end{aligned}$$

This completes the proof under the stepsize condition $\gamma \leq 1/4L$ and $B := \inf_{k \geq 0} B^k$. \square

B.2.4 Proof of Lemma 3.6.2

The following lemma provides an upper bound on the residual error due to delays.

Lemma 3.6.2 (Residual Estimation). Under Assumption 3.3.1, the iterates of Ringleader ASGD (Algorithm 3.5.1) with stepsize $\gamma \leq 1/4nL$ satisfy the following bound

$$\frac{1}{K} \sum_{k=0}^{K-1} \frac{1}{n} \sum_{i=1}^n \mathbb{E} [\|x^k - x^{k-\delta_i^k}\|^2] \leq \frac{2\gamma n}{LK} \sum_{k=0}^{K-1} \mathbb{E} \left[\left\| \frac{1}{n} \sum_{j=1}^n \nabla f_j(x^{k-\delta_j^k}) \right\|^2 \right] + \frac{2\gamma \sigma^2}{LB}.$$

Proof. By Young's inequality, we have

$$\begin{aligned}
\mathbb{E} \left[\|x^k - x^{k-\delta_i^k}\|^2 \right] &= \mathbb{E} \left[\left\| \gamma \sum_{\ell=k-\delta_i^k}^{k-1} \bar{g}^\ell \right\|^2 \right] \\
&\leq 2\gamma^2 \mathbb{E} \left[\left\| \sum_{\ell=k-\delta_i^k}^{k-1} \frac{1}{n} \sum_{j=1}^n \nabla f_j(x^{\ell-\delta_j^\ell}) \right\|^2 \right] \\
&\quad + 2\gamma^2 \mathbb{E} \left[\left\| \sum_{\ell=k-\delta_i^k}^{k-1} \left(\bar{g}^\ell - \frac{1}{n} \sum_{j=1}^n \nabla f_j(x^{\ell-\delta_j^\ell}) \right) \right\|^2 \right] \\
&\leq 2\gamma^2 \underbrace{\delta_i^k \sum_{\ell=k-\delta_i^k}^{k-1} \mathbb{E} \left[\left\| \frac{1}{n} \sum_{j=1}^n \nabla f_j(x^{\ell-\delta_j^\ell}) \right\|^2 \right]}_{T_{ik}} + 2\gamma^2 (\delta_i^k)^2 \frac{\sigma^2}{Bn}.
\end{aligned}$$

In the last inequality, we used Jensen's inequality and Lemma B.2.1.

Next, we estimate the sum of T_{ik}

$$\begin{aligned}
\sum_{k=0}^{K-1} \frac{1}{n} \sum_{i=1}^n T_{ik} &= \sum_{k=0}^{K-1} \frac{1}{n} \sum_{i=1}^n \delta_i^k \sum_{\ell=k-\delta_i^k}^{k-1} \mathbb{E} \left[\left\| \frac{1}{n} \sum_{j=1}^n \nabla f_j(x^{\ell-\delta_j^\ell}) \right\|^2 \right] \\
&= \frac{1}{n} \sum_{i=1}^n \sum_{k=0}^{K-1} \delta_i^k \sum_{\ell=k-\delta_i^k}^{k-1} \mathbb{E} \left[\left\| \frac{1}{n} \sum_{j=1}^n \nabla f_j(x^{\ell-\delta_j^\ell}) \right\|^2 \right] \\
&\leq 2 \sum_{i=1}^n \sum_{k=0}^{K-1} \sum_{\ell=k-\delta_i^k}^{k-1} \mathbb{E} \left[\left\| \frac{1}{n} \sum_{j=1}^n \nabla f_j(x^{\ell-\delta_j^\ell}) \right\|^2 \right] \\
&\leq 2 \sum_{i=1}^n \delta_i^{\max} \sum_{k=0}^{K-1} \mathbb{E} \left[\left\| \frac{1}{n} \sum_{j=1}^n \nabla f_j(x^{k-\delta_j^k}) \right\|^2 \right] \\
&\leq 4n^2 \sum_{k=0}^{K-1} \mathbb{E} \left[\left\| \frac{1}{n} \sum_{j=1}^n \nabla f_j(x^{k-\delta_j^k}) \right\|^2 \right].
\end{aligned}$$

In the first and last inequality, we used the bound $\delta_i^{\max} \leq 2n$ from Lemma 3.5.1. Finally, applying the stepsize condition $\gamma \leq 1/(4nL)$ yields the result. \square

B.3 Time Complexity of IA²SGD

The iteration complexity of IA²SGD (Wang et al., 2025) is

$$K = \mathcal{O} \left(\frac{\delta^{\max} L \Delta}{\varepsilon} \left(1 + \frac{\sigma^2}{n \varepsilon} \right) \right).$$

We now analyze the corresponding wall-clock time under the *fixed computation model* (3.2). Since the algorithm performs an update whenever a single worker

finishes a computation, we seek the minimal time T such that

$$\sum_{i=1}^n \left\lfloor \frac{T}{\tau_i} \right\rfloor \geq K .$$

Observe that

$$\sum_{i=1}^n \frac{T}{\tau_i} \geq \sum_{i=1}^n \left\lfloor \frac{T}{\tau_i} \right\rfloor .$$

Hence, if we define T' by

$$\sum_{i=1}^n \frac{T'}{\tau_i} = K ,$$

then

$$T' = \left(\sum_{i=1}^n \frac{1}{\tau_i} \right)^{-1} K .$$

It follows that the minimal time T is necessarily larger than T' .

It remains to bound δ^{\max} . At initialization, all workers start computing their first gradients simultaneously. By the time the slowest worker completes its first gradient (at time τ_n), the other workers may each have completed multiple gradients. In particular,

$$\delta^{\max} \geq \sum_{i=1}^n \left\lfloor \frac{\tau_n}{\tau_i} \right\rfloor .$$

Combining this with the iteration complexity bound, we obtain that the total runtime satisfies

$$T \geq c \times \frac{\tau_n L \Delta}{\varepsilon} \left(1 + \frac{\sigma^2}{n \varepsilon} \right) ,$$

for some universal constant $c > 0$.

Note that the expression above should not be viewed as an exact upper bound on the runtime. It is better understood as a simplified estimate of T , which is sufficient for our purposes and provides a cleaner basis for comparison.

B.4 Improved Malenia SGD

Malenia SGD has the following iteration complexity (Tyurin & Richtárik, 2024)

$$K \geq \frac{12\Delta L_f}{\varepsilon} + \frac{12\Delta L_f \sigma^2}{\varepsilon^2 n S} ,$$

where S is a lower bound on the harmonic mean of the batch sizes, i.e.,

$$\left(\frac{1}{n} \sum_{i=1}^n \frac{1}{b_i^k} \right)^{-1} \geq S ,$$

for all iterations k . In the original Malenia SGD analysis (Tyurin & Richtárik, 2024), this bound follows from the condition in (3.3), which fixes the same value of S across all iterations.

In the fixed-time regime (3.2), however, this condition is no longer necessary.

By adopting the same strategy as Ringleader ASGD (Algorithm 3.5.1)—namely, waiting for at least one gradient from each worker—we effectively replace S with B in the rate. This yields the following time complexity

$$\tau_n K = \frac{12\tau_n \Delta L_f}{\varepsilon} \left(1 + \frac{\sigma^2}{\varepsilon n B}\right).$$

Substituting the expression for B from Lemma 3.6.4 and proceeding as in the proof of Theorem 3.6.5, we obtain the same overall time complexity as before—this time *without* requiring condition (3.3), which depends on knowing σ and fixing ε in advance.

Finally, note that this improvement is only valid in the fixed-time regime. In the setting with arbitrarily varying computation times, the same optimization cannot be applied, for the same reasons discussed for Ringleader ASGD in Section B.1.

Appendix C

Appendix for Chapter 4

C.1 Additional Experiments

The objective function is a convex quadratic function $f : \mathbb{R}^d \rightarrow \mathbb{R}$ defined as

$$f(x) = \frac{1}{2}x^\top Ax - b^\top x ,$$

where

$$A = \frac{1}{4} \begin{bmatrix} 2 & -1 & & 0 \\ -1 & \ddots & \ddots & \\ & \ddots & \ddots & -1 \\ 0 & & -1 & 2 \end{bmatrix} \in \mathbb{R}^{d \times d}, \quad \text{and} \quad b = \frac{1}{4} \begin{bmatrix} -1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \in \mathbb{R}^d .$$

We denote f^* as the minimum value of the function f . Each of the n workers is able to calculate unbiased stochastic gradients $g(x)$ that satisfy

$$\mathbb{E} [\|g(x) - \nabla f(x)\|^2] \leq 0.01^2 .$$

This is achieved by adding Gaussian noise to the gradients of f .

The experiments were implemented in Python. The distributed environment was emulated on machines with Intel(R) Xeon(R) Gold 6248 CPU @ 2.50GHz.

C.1.1 Linear Noise

In this section we model the computation time for worker i by the distribution

$$\nu_i = 29i + \text{Exp}(29i), \quad \text{for all } i \in [n] .$$

The expected value of this distribution is $\mu_i = 2 \cdot 29i$. Furthermore, the Orlicz norm satisfies the bound $\alpha_i \leq 2\mu_i$.

We again set $B = 23$ and run simulations similar to those in Section 4.7. The results are shown in Figure C.1.1.

The important difference to the previous Figure 4.7.1 is that here ATA-Empirical performs better than ATA. This is because the Orlicz norm $\alpha = 4 \cdot 29n$ is much larger.

Similarly, we provide a numerical comparison in Table C.1.

C.1.2 Heterogeneous Time Distributions

So far, we have only considered cases where clients follow the same distributions but with different means. In this section, we extend our experiments to cases

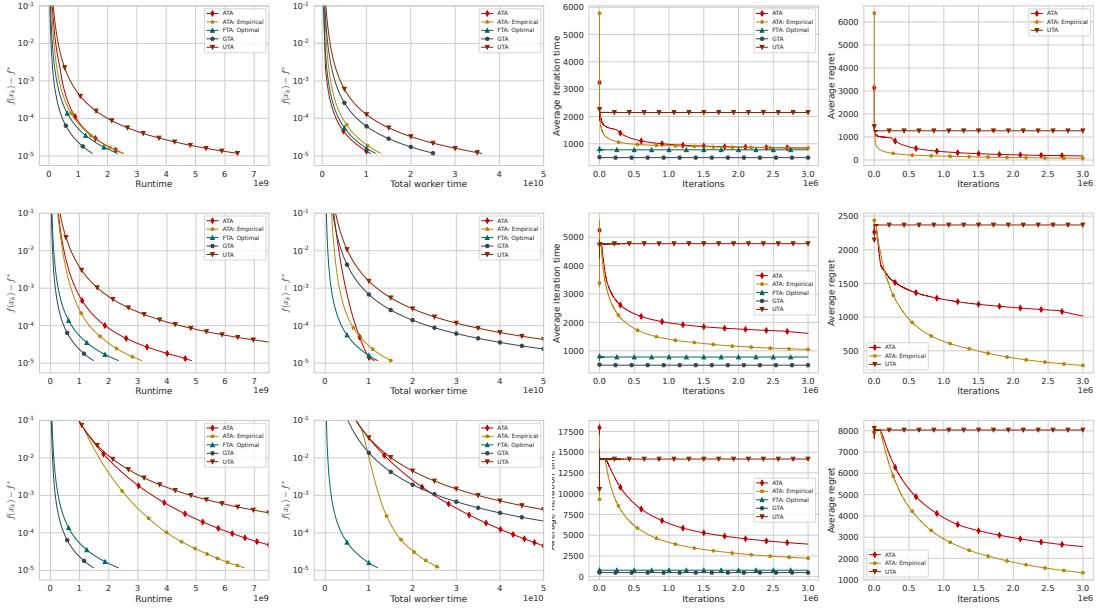


Figure C.1.1: Each row increases the number of workers by a factor of 3, starting from 17, that is, $n = 17, 51, 153, 459$ from top to bottom. The first column shows runtime vs. suboptimality. The second column also plots suboptimality, but against total worker time, i.e., $\sum_{i=1}^n T_i^k$ in Algorithm 4.4.1. The third column presents the average iteration time, given by C_k/k over all iterations k . The last column displays the averaged cumulative regret, as defined in (4.9).

where the distributions themselves differ. We consider five distributions: Exponential, Uniform, Half-Normal, lognormal, and Gamma. We group five workers with these five distributions so that each group has the same mean, then vary the mean across different groups. More concretely, we use:

- $\text{Exp}(c(5g + 1))$,
- $\text{Uniform}\left(\frac{c(5g+1)}{2}, 3\frac{c(5g+1)}{2}\right)$,
- $|\mathcal{N}(0, c(5g + 1)\sqrt{\frac{\pi}{2}})|$,
- $\text{Lognormal}\left(\log(c(5g + 1))/2, \sqrt{\log(c(5g + 1))}\right)$,
- $\text{Gamma}\left((c(5g + 1))^2, \frac{1}{c(5g+1)}\right)$ with shape and scale parameters.

Next, we add a constant $c(5g + 1)$ to all the distributions, where $c = 29$, and g represents the group number, starting from 0. The clients are divided into $n/5$ groups.

The results of the experiments are shown in Figure C.1.2. The plots demonstrate that the algorithms are robust across different distributions.

C.1.3 Regret

In this section, we verify Theorem 4.6.1 and 4.6.3 on regret through simulations. We set $n = 20$ and $B = 5$, with the computation time for worker i following the

Table C.1: Ratios of total worker times and runtimes required to achieve $f(x) - f^* < 10^{-5}$. For total worker time, we divide the total worker time of **GTA** by the corresponding total worker times of the other algorithms listed. For runtime, we do the opposite, dividing the runtime of the other algorithms by the runtime of **GTA**, since **GTA** is the fastest.

n	TOTAL WORKER TIME RATIO			RUNTIME RATIO		
	ATA	ATA-Empirical	OFTA	ATA	ATA-Empirical	OFTA
17	2.32	1.91	2.1	1.71	1.71	1.58
51	6.71	5.02	6.29	3.27	2.12	1.58
153	3.41	8.68	18.87	7.96	4.5	1.58

distribution

$$\nu_i = \text{Exp}(2i), \quad \text{for all } i \in [n].$$

We ran the simulation five times, and the plots include standard deviation bars, although they are not visible. The results are presented in Figure C.1.3.

As expected, the regret grows logarithmically.

C.1.4 Real Dataset

In this section, we present an experiment where we train a convolutional neural network (CNN) on the CIFAR-100 dataset (Krizhevsky et al., 2009). The network consists of three convolutional layers and two fully connected layers, with a total of 160k parameters.

We use the Adam optimizer (Kingma & Ba, 2014) with a constant step size of $8 \cdot 10^{-5}$. The computation time of the workers follows the same setup as in Figure C.1.1. The results are shown in Figure C.1.4.

C.1.5 Impact of Prior Knowledge on Time Distributions

In real-world systems where multiple machine learning models are trained, estimates of computation times from previous runs may be available. With this prior knowledge, **ATA** and **ATA-Empirical** can be much faster, as they spend less time on exploration and quickly approach the performance of **OFTA**.

To illustrate this, we vary the number of prior runs, P . Since our algorithms operate independently of the underlying optimization process, we first focus solely on the bandit component, updating the confidence scores of machines over several iterations. We then apply the loss curves to different segments of the bandit phase and compare the results as P increases. A larger P yields more accurate estimates.

The optimization setup remains the same as in Figure C.1.4, with $B = 23$ and $n = 51$. The results are presented in Figure C.1.5.

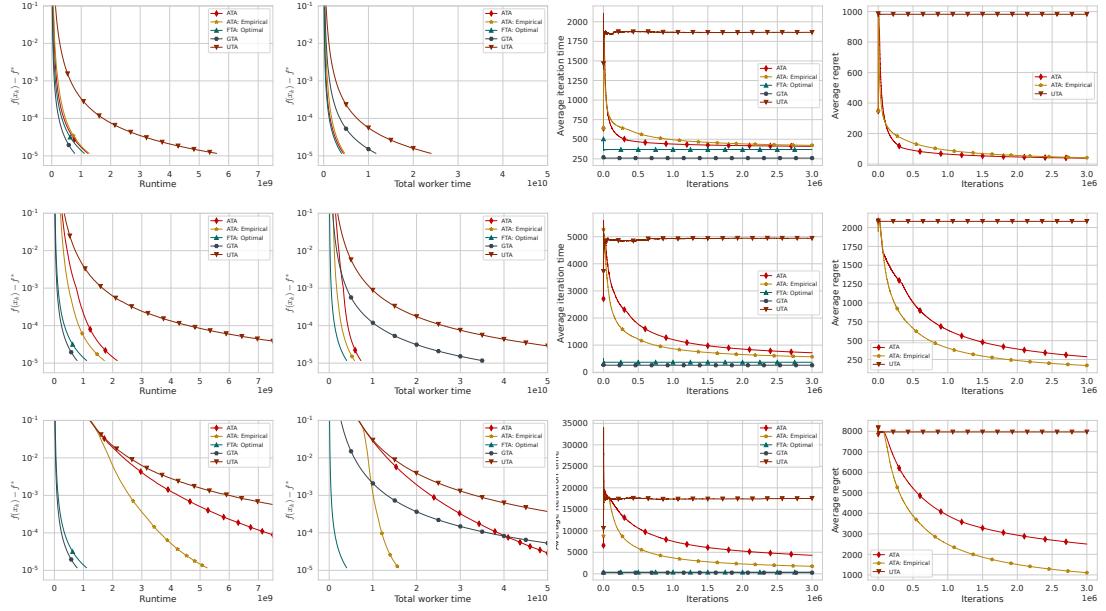


Figure C.1.2: Each row corresponds to an increasing number of workers, with $n = 15, 45, 150$ from top to bottom. We consider five distributions—Exponential, Uniform, Half-Normal, Lognormal, and Gamma—grouping them to have the same mean and then varying the mean across different groups. The results demonstrate that the algorithms remain robust across different distributions. The columns represent the same as in Figure C.1.1.

C.2 Concrete Optimization Methods

In this section, we provide concrete examples of optimization algorithms using the ATA and GTA allocation strategies.

For optimization problems, we focus on SGD and Asynchronous SGD. Other methods, such as stochastic proximal point methods and higher-order methods, can be developed in a similar fashion.

C.2.1 Stochastic Gradient Descent

Consider the problem of finding an approximate stationary point of the optimization problem

$$\underset{x \in \mathbb{R}^d}{\text{minimize}} \quad \{f(x) := \mathbb{E}_{\xi \sim \mathcal{D}} [f(x; \xi)]\}. \quad (\text{C.1})$$

We assume that each worker is able to compute stochastic gradient $f(x; \xi)$ satisfying $\mathbb{E}_{\xi \sim \mathcal{D}} [\|f(x; \xi) - \nabla f(x)\|^2] \leq \sigma^2$ for all $x \in \mathbb{R}^d$.

In this case, SGD with allocation budget B becomes Minibatch SGD with batch size B . The next step is determining how the batch is collected. For ATA, we refer to this method as SGD-ATA, as described in Algorithm C.2.1.

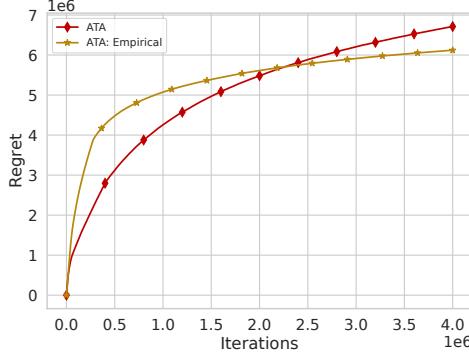


Figure C.1.3: Regret growth over iterations.

Algorithm C.2.1 SGD-ATA

- 1: **Optimization inputs:** initial point $x^1 \in \mathbb{R}^d$, stepsize $\gamma > 0$
- 2: **Allocation inputs:** allocation budget B
- 3: **Initialize:** empirical means $\hat{\mu}_i^1 = 0$, usage counts $K_i^1 = 0$, and usage times $T_i^1 = 0$, for all $i \in [n]$
- 4: **for** $k = 1, \dots, K$ **do**
- 5: Compute LCBs (s_i^k) for all $i \in [n]$
- 6: Find allocation:

$$a^k \in \arg \min_{a \in \mathcal{A}} \ell(a, s^k)$$

- 7: Allocate a_i^k tasks to each worker $i \in [n]$
- 8: Update x :

$$x^{k+1} = x^k - \frac{\gamma}{B} \sum_{i=1}^n \sum_{j=1}^{a_i^k} \nabla f(x^k; \xi_i^j)$$

- 9: For all i such that $a_i^k \neq 0$, update:

$$\begin{aligned} K_i^{k+1} &= K_i^k + a_i^k \\ T_i^{k+1} &= T_i^k + \sum_{j=1}^{a_i^k} X_i^{k,j} \\ \hat{\mu}_i^{k+1} &= \frac{T_i^{k+1}}{K_i^{k+1}} \end{aligned}$$

- 10: **end for**
-

In this case, each task consists in calculating the gradient using the device's local data, which is assumed to have the same distribution as the data on all other devices. Because of this, it does not matter which device performs the task. The method then averages these gradients to obtain an unbiased gradient estimator and performs a gradient descent step.

Now, let us give the version of Minibatch SGD using greedy allocation Algorithm C.2.2.

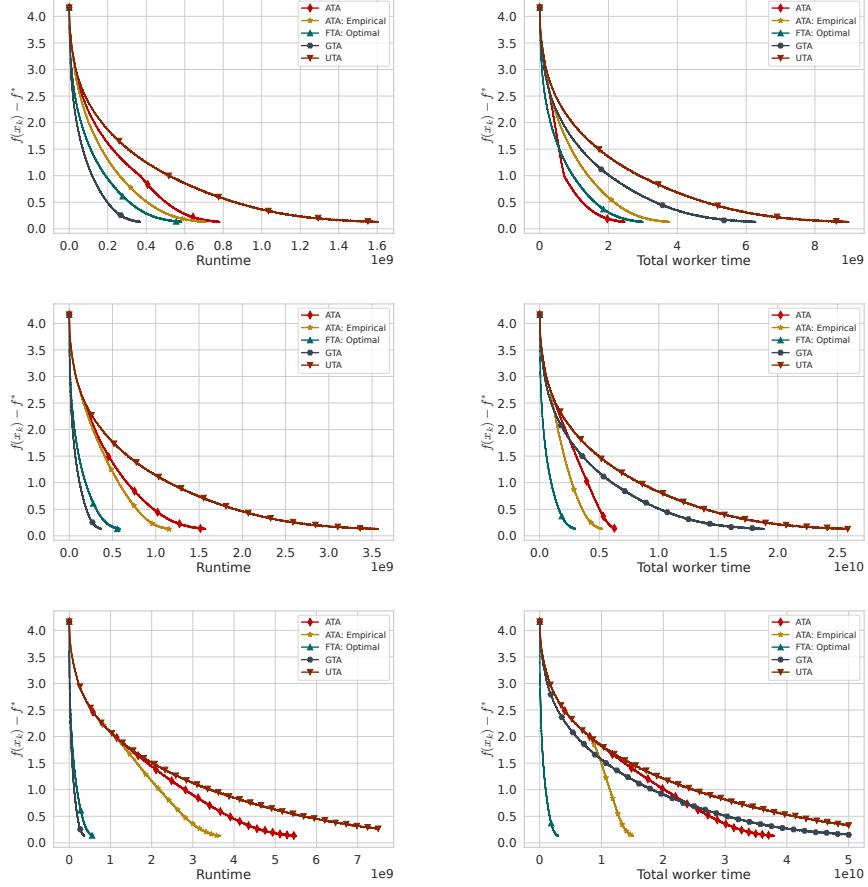


Figure C.1.4: We use the CIFAR-100 dataset (Krizhevsky et al., 2009). The model is a CNN with three convolutional layers and two fully connected layers, totaling 160k parameters. The Adam optimizer (Kingma & Ba, 2014) is used with a constant step size of $8 \cdot 10^{-5}$. The computation time of the workers follows the same setup as in Figure C.1.1, where the mean time increases linearly. The batch size remains the same at $B = 23$. Each row corresponds to a different number of workers, with $n = 17, 51, 153$ from top to bottom.

Algorithm C.2.2 SGD-GTA

- 1: **Input:** initial point $1 \in \mathbb{R}^d$, stepsize $\gamma > 0$, allocation budget B
- 2: **for** $k = 1, \dots, K$ **do**
- 3: $b = 0$
- 4: Query single gradient from each worker $i \in [n]$
- 5: **while** $b < B$ **do**
- 6: Gradient $\nabla f(x^k; \xi^{k,b})$ arrives from worker $i^{k,b}$
- 7: $g^k = g^k + \nabla f(x^k; \xi^{k,b})$
- 8: $b = b + 1$
- 9: Query gradient at x^k from worker $i^{k,b}$
- 10: **end while**
- 11: Update the model:

$$x^{k+1} = x^k - \gamma \frac{g^k}{B}$$
- 12: **end for**

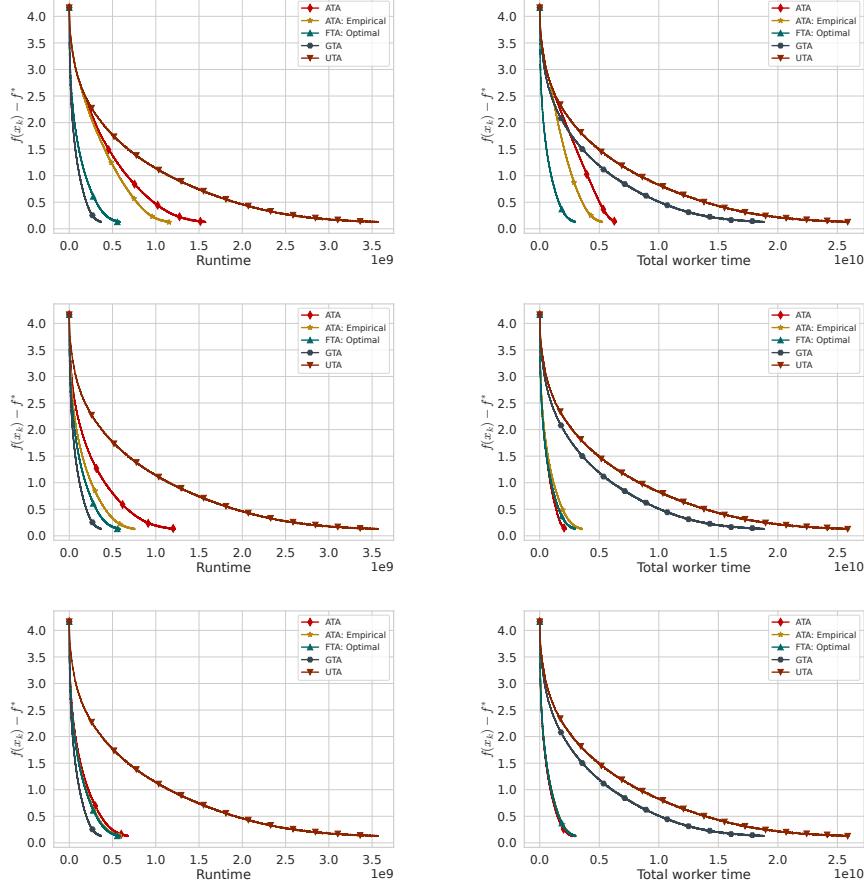


Figure C.1.5: We use the same optimization setup as in Figure C.1.4, with $B = 23$ and $n = 51$. The number of prior iterations, P , varies across rows, starting from the top with $P = 0, 5 \cdot 10^5, 5 \cdot 10^6$. As the number of prior iterations increases, we observe that the training of ATA and ATA-Empirical accelerates, bringing their performance closer to the optimal performance of OFTA.

Algorithm C.2.2 corresponds exactly to the Rennala SGD method proposed by Tyurin & Richtárik (2024), which has optimal time complexity when the objective function is non-convex and smooth.

If the computation times are deterministic, then GTA makes the same allocation in each iteration. In that case, SGD-ATA will converge to this fixed allocation. If the times are random, the allocation found by GTA may vary in each iteration. In this case, SGD-ATA will approach the best allocation for the expected times.

C.2.2 Asynchronous SGD

Here, we focus on the homogeneous problem given in Equation (C.1). The greedy variant, Ringmaster ASGD, was proposed by Maranjyan et al. (2025d) and, like Rennala SGD, achieves the best runtime.

We now present its version with ATA, given in Algorithm C.2.3.

Algorithm C.2.3 ASGD-ATA

- 1: **Optimization inputs:** initial point $x^1 \in \mathbb{R}^d$, stepsize $\gamma > 0$
- 2: **Allocation inputs:** allocation budget B
- 3: **Initialize:** empirical means $\hat{\mu}_i^1 = 0$, usage counts $K_i^1 = 0$, and usage times $T_i^1 = 0$, for all $i \in [n]$
- 4: **for** $k = 1, \dots, K$ **do**
- 5: Compute LCBs (s_i^k) for all $i \in [n]$
- 6: Find allocation: $a^k = \text{RAS}(s^k; B)$
- 7: Update x^k using Algorithm C.2.4 with allocation a^k
- 8: For all i such that $a_i^k \neq 0$, update:

$$\begin{aligned} K_i^{k+1} &= K_i^k + a_i^k \\ T_i^{k+1} &= T_i^k + \sum_{j=1}^{a_i^k} X_i^{k,j} \\ \hat{\mu}_i^{k+1} &= \frac{T_i^{k+1}}{K_i^{k+1}} \end{aligned}$$

- 9: **end for**
-

Algorithm C.2.4 ASGD updates

- 1: **Input:** Initial point $x^1 \in \mathbb{R}^d$, stepsize $\gamma > 0$, allocation vector a with $\|a\|_1 = B$
- 2: Workers with $a_i > 0$ start computing stochastic gradients at x^1
- 3: **for** $s = 1, 2, \dots, B$ **do**
- 4: Receive gradient $\nabla f(x^{s+\delta^s}; \xi_{i^s}^{s+\delta^s})$ from worker i^s
- 5: Update: $x^{s+1} = x^s - \gamma \nabla f(x^{s+\delta^s}; \xi_{i^s}^{s+\delta^s})$
- 6: **if** $a_{i^s} > 0$ **then**
- 7: Worker i^s begins computing $\nabla f(x^{s+1}; \xi_{i^s}^{s+1})$
- 8: Decrease remaining allocation for worker i^s by one: $a_{i^s} = a_{i^s} - 1$
- 9: **end if**
- 10: **end for**
- 11: **return:** x^{B+1}

The sequence $\{\delta^s\}$ represents delays, where $\delta^s \geq 0$ is the difference between the iteration when worker i^s started computing the gradient and iteration s , when it was applied.

Here, the task remains gradient computation, but each worker's subsequent tasks use different points for computing the gradient. These points depend on the actual computation times and the asynchronous nature of the method, hence the name Asynchronous SGD.

C.3 Recursive Allocation Selection Algorithm

In this section, we introduce an efficient method for finding the best allocation. Given LCBs s^k and allocation budget B , each iteration of ATA (Algorithm 4.4.1)

determines the allocation by solving

$$a^k \in \arg \min_{a \in \mathcal{A}} \ell(a, s^k) ,$$

where

$$\ell(a, \mu) := \max_{i \in [n]} a_i \mu_i = \|a \odot \mu\|_\infty ,$$

with \odot denoting the element-wise product. When clear from context, we write $\ell(a)$ instead of $\ell(a, \mu)$.

In the early iterations, when some s_i values are 0, ATA allocates uniformly across these arms until all s_i values become positive. After that, the allocation is determined using the recursive routine in Algorithm C.3.1.

Algorithm C.3.1 Recursive Allocation Selection (RAS)

- 1: **Input:** Scores s_1, \dots, s_n , allocation budget B
- 2: Assume without loss of generality that $s_1 \leq s_2 \leq \dots \leq s_n$ (i.e., sort the scores)
- 3: **if** $B = 1$ **then**
- 4: **return:** $(1, 0, \dots, 0)$
- 5: **end if**
- 6: Find the previous best allocation:

$$a = (a_1, \dots, a_n) = \text{RAS}(s_1, \dots, s_n; B - 1)$$

- 7: Determine the first zero allocation:

$$r = \begin{cases} \min\{i \mid a_i = 0\}, & \text{if } a_n = 0, \\ n, & \text{otherwise} \end{cases} \quad (\text{C.2})$$

- 8: Find the best next query allocation set:

$$M = \arg \min_{i \in [r]} \|(a + e_i) \odot s\|_\infty ,$$

where e_i is the unit vector in direction i .

- 9: Select $j \in M$ such that the cardinality of

$$\arg \max_{i \in [r]} (a_i + e_{j,i}) s_i$$

is minimized

- 10: **return:** $a + e_j$
-

Remark C.3.1. *The iteration complexity of RAS is*

$$\mathcal{O}(n \ln(\min\{B, n\}) + \min\{B, n\}^2) .$$

In fact, the first $n \ln(\min\{B, n\})$ term arises from identifying the smallest B scores. For the second term, note that in (C.2), we have $r \leq \min\{B, n\}$.

C.3.1 Optimality

We now prove that RAS finds the optimal allocation, as stated in the following lemma.

Lemma C.3.2. For positive scores $0 < s_1 \leq s_2 \leq \dots \leq s_n$, RAS (Algorithm C.3.1) finds an optimal allocation $h \in \mathcal{A}$, satisfying

$$h \in \arg \min_{a \in \mathcal{A}} \|a \odot s\|_\infty .$$

Proof. We prove the claim by induction on the allocation budget B .

Base Case ($B = 1$): When $B = 1$, RAS (Algorithm C.3.1) allocates the task to worker with the smallest score (line 9). Thus, the base case holds.

Inductive Step: Assume that RAS finds an optimal allocation for budget $B - 1$, denoted by

$$\bar{h} = \text{RAS}(s_1, \dots, s_n; B - 1) .$$

We need to prove that the solution returned for budget B , denoted by $h = \bar{h} + e_r$, is also optimal.

Assume, for contradiction, that there exists $a \in \mathcal{A}$ such that $a \neq h$ and $\ell(a) < \ell(h)$. Write $a = \bar{a} + e_q$ for some $q \in [n]$. Observe that $\|\bar{a}\|_1 = B - 1$ because $a \in \mathcal{A}$.

We consider two cases based on the value of $\ell(\bar{h} + e_r)$:

- $\ell(\bar{h} + e_r) = h_k s_k$ for some $k \neq r$. In this case, adding one unit to index r does not change the maximum value, i.e., $\ell(\bar{h}) = \ell(\bar{h} + e_r)$. By the inductive hypothesis, \bar{h} minimizes $\ell(\mathbf{x})$ for budget $B - 1$. Therefore, we have

$$\ell(a) \geq \ell(\bar{a}) \geq \ell(\bar{h}) = \ell(\bar{h} + e_r) = \ell(h) ,$$

which contradicts the assumption that $\ell(a) < \ell(h)$.

- $\ell(\bar{h} + e_r) = (\bar{h}_r + 1) s_r$. By the algorithm's logic, $(\bar{h}_i + 1) s_i \leq (\bar{h}_r + 1) s_r$ for all $i \neq r$. Since $\ell(\bar{h} + e_r) \leq \ell(\bar{h} + e_q)$ and we assumed $\ell(\bar{a} + e_q) = \ell(a) < \ell(h) = \ell(\bar{h} + e_r)$, then $\bar{a} \neq \bar{h}$ otherwise $\ell(\bar{a} + e_q) < \ell(\bar{a} + e_r)$. Given that $\|\bar{h}\|_1 = \|\bar{a}\|_1$, this implies that there exists some $u \in [n]$ such that $0 \leq \bar{a}_u \leq \bar{h}_u - 1$ and another index $v \in [n]$ where $\bar{a}_v \geq \bar{h}_v + 1$.

In addition, note that r is chosen such that $\ell(\bar{h} + e_r)$ is minimum. Using the fact that $\ell(\bar{h} + e_r) = (\bar{h}_r + 1) s_r$, we have that for any index q , we also necessarily have $\ell(\bar{h} + e_q) = (\bar{h}_q + 1) s_q$. Using this, we deduce

$$\ell(h) = \ell(\bar{h} + e_r) \leq \ell(\bar{h} + e_v) = (\bar{h}_v + 1) s_v \leq \max_i \bar{a}_i s_i = \ell(\bar{a}) \leq \ell(a) ,$$

where in the second inequality we used the fact that $\bar{a}_v \geq \bar{h}_v + 1$ and in the last inequality we used the fact that the loss is not decreasing for we add one element to the vector. This chain of inequalities again contradicts the assumption that $\ell(a) < \ell(h)$.

Since both cases lead to contradictions, we conclude that no $a \in \mathcal{A}$ exists with $\ell(a) < \ell(h)$. Thus, RAS produces an optimal allocation for budget B . \square

C.3.2 Minimal Cardinality

Among all possible allocations RAS choose one that always minimizes the cardinality of the set:

$$\arg \max_{i \in [n]} a_i s_i .$$

The reason for this choice is just technical as it allows the Lemma C.4.1 to be true.

Lemma C.3.3. The output of RAS ensures the smallest cardinality of the set:

$$\arg \max_{i \in [n]} a_i s_i$$

among all the optimal allocations a .

Proof. This proof uses similar reasoning as the one before.

Let $h = \text{RAS}(s; B)$, and denote the cardinality of the set $\arg \max_{i \in [n]} a_i s_i$ for allocation a by

$$C_B(a) = \left| \arg \max_{i \in [n]} a_i s_i \right| \geq 1 .$$

We prove the claim by induction on B .

Base Case ($B = 1$): For $B = 1$, there is a single coordinate allocation, thus $C_1(h) = 1$, which is the smallest possible cardinality.

Inductive Step: Assume that RAS finds an optimal allocation for budget $B - 1$ with the smallest cardinality, denote its output by

$$\bar{h} = \text{RAS}(s_1, \dots, s_n; B - 1) .$$

We need to prove that $h = \bar{h} + e_r$ minimizes $C_B(a)$ among all optimal allocations for budget B .

Assume, for contradiction, that there exists $a \in \mathcal{A}$ such that $a \neq h$, $\ell(a) = \ell(h)$, and $C_B(a) < C_B(h)$. Write $a = \bar{a} + e_q$ for some $q \in [n]$. We consider three cases:

- $C_B(h) = 1$. Since the minimum cardinality is exactly 1, we must have $C_B(a) \geq 1 = C_B(h)$, that contradicts our assumption.
- $C_B(h) = C_{B-1}(\bar{h}) > 1$. This occurs when $\ell(h) = \ell(\bar{h}) \neq (\bar{h}_r + 1) s_r$. By the optimality of h , we have $\ell(\bar{h}) \leq \ell(\bar{a}) \leq \ell(a) = \ell(h) = \ell(\bar{h})$, which implies $\ell(\bar{a}) = \ell(a)$. Therefore, $C_{B-1}(\bar{a}) \leq C_B(a)$. Since the induction hypothesis holds for $B - 1$, we have $C_{B-1}(\bar{h}) \leq C_{B-1}(\bar{a})$. Thus,

$$C_B(h) = C_{B-1}(\bar{h}) \leq C_{B-1}(\bar{a}) \leq C_B(a) ,$$

which leads to a contradiction.

- $C_B(h) = C_{B-1}(\bar{h}) + 1$. This occurs when $\ell(h) = \ell(\bar{h}) = (\bar{h}_r + 1) s_r$. Proceeding as in the previous case, we have $\ell(\bar{a}) = \ell(a)$, and hence $C_{B-1}(\bar{a}) \leq C_B(a)$. Since the induction hypothesis holds for $B - 1$, we know $C_{B-1}(\bar{h}) \leq C_{B-1}(\bar{a})$.

We now have additional cases:

- If $C_{B-1}(\bar{a}) = C_{B-1}(\bar{h}) + 1$, then

$$C_B(h) = C_{B-1}(\bar{h}) + 1 = C_{B-1}(\bar{a}) \leq C_B(a) ,$$

which leads to a contradiction.

- Now assume $C_{B-1}(\bar{a}) = C_{B-1}(\bar{h})$. We will show that in this case, $C_B(a) = C_{B-1}(\bar{a}) + 1$. By contradiction, suppose $C_B(a) = C_{B-1}(\bar{a})$, which implies $(\bar{a}_q + 1)s_q < \ell(a)$. Let k be an index such that $\bar{a}_k s_k = \ell(a)$. Construct a new allocation $a' = \bar{a} + e_q - e_k$. Then,

$$C_{B-1}(a') = C_{B-1}(\bar{a}) - 1 < C_{B-1}(\bar{h}) ,$$

which contradicts the induction hypothesis. Thus,

$$C_B(a) = C_{B-1}(\bar{a}) + 1 .$$

Using this, we have

$$C_B(h) = C_{B-1}(\bar{h}) + 1 = C_{B-1}(\bar{a}) + 1 = C_B(a) ,$$

which again contradicts $C_B(a) < C_B(h)$.

This concludes the proof. \square

C.4 Proofs of Theorem 4.6.1, Theorem 4.6.3, and Theorem 4.4.2

We start by recalling the notation. For $i \in [n]$ and $k \in [K]$, $(X_{i,k}^{(u)})_{u \in [B]}$ denote B independent samples at round k from distribution ν_i . When using an allocation vector $a^k \in \mathcal{A}$, the total computation time of worker i at round k is $\sum_{u=1}^{a_i^k} X_{i,k}^{(u)}$, when $a_i^k > 0$. $\mu = (\mu_1, \dots, \mu_K)$ is the vector of means. For each $k \in [K]$, when using the allocation vector a^k , we recall the definition of the proxy loss $\ell : \mathcal{A} \times \mathbb{R}_+^n \rightarrow \mathbb{R}_+$ by

$$\ell(a^k, \lambda) = \max_{i \in [n]} a_i^k \lambda_i ,$$

where $\lambda = (\lambda_1, \dots, \lambda_n)$ is a vector of non-negative components. When $\lambda = \mu$, we drop the dependence on the second input of ℓ . For each λ , let $\bar{a}_\lambda \in \mathcal{A}$, be the action minimizing this loss

$$\bar{a}_\lambda \in \arg \min_{a \in \mathcal{A}} \ell(a, \lambda) .$$

We drop the dependency on μ from \bar{a}_μ to ease notation. The actual (random) computation time at round k is denoted by $C : \mathcal{A} \rightarrow \mathbb{R}_+$:

$$C(a^k) := \max_{i \in [n]} \sum_{u=1}^{a_i^k} X_i^{k,u} . \tag{4.1}$$

Let a^* be the action minimizing the expected time

$$a^* \in \arg \min_{a \in \mathcal{A}} \mathbb{E}[C(a)] .$$

The expected regret after K rounds is defined as follows

$$\mathcal{R}_K := \sum_{t=1}^K \mathbb{E} [\ell(a^k) - \ell(\bar{a})] .$$

For the remainder of this analysis we consider $\bar{a} \in \arg \min_{a \in \mathcal{A}} \ell(a)$ found using the RAS procedure. For each $i \in [n]$, recall that k_i is the smallest integer such that

$$(\bar{a}_i + k_i)\mu_i > \ell(\bar{a}) . \quad (\text{C.3})$$

Below we present a technical lemma used in the proofs of Theorems 4.6.1 and 4.6.3.

Lemma C.4.1. Let $x = (x_1, \dots, x_n) \in \mathbb{R}_{\geq 0}^n$. Let a be the output of $\text{RAS}(x; B)$. For each $i, j \in [n]$, we have

$$a_j x_j \leq (a_i + 1) x_i .$$

Proof. Fix $x \in \mathbb{R}_+^n$, and let $a = \text{RAS}(x; B)$. The result is straightforward when $\min_{i \in [n]} x_i = 0$.

Suppose that $x_i > 0$ for all $i \in [n]$. Let $s \geq 1$ denote the cardinality

$$s := \left| \arg \max_{i \in [n]} a_i x_i \right| .$$

Fix $i, j \in [n]$, let $k \in \arg \max_{i \in [n]} a_i x_i$. We need to show that

$$a_k x^k \leq (a_i + 1) x_i .$$

We use a proof by contradiction. Suppose that we have $a_k x^k > (a_i + 1) x_i$ consider the allocation vector $a' \in \mathcal{A}$ given by $a'_k = a_k - 1$, $a'_i = a_i + 1$ and $a'_u = \bar{a}_u$ when $u \notin \{i, k\}$. Let $R := \max_{u \neq i, k} \{a_u x_u\}$. We have

$$\ell(a', x) = \max_{u \in [n]} a'_u x_u = \max \{(a_i + 1)x_i, (a_k - 1)x^k, R\} .$$

We consider two cases:

- Suppose that $s = 1$ (i.e., the only element in $[n]$ such that $a_u x_u = \ell(a, x)$ is k), then we have necessarily $R < a_k x^k$. Moreover, by the contradiction hypothesis, $(a_i + 1)x_i < a_k x^k$. Therefore,

$$\ell(a', x) = \max \{(a_i + 1)x_i, (a_k - 1)x^k, R\} < a_k x^k = \ell(a, x) ,$$

which contradicts the definition of a .

- Suppose that $s \geq 2$, since by hypothesis $a_k x^k > (a_i + 1)x_i$, we clearly have $a_i x_i < \ell(a, x)$ therefore among the set $[n] \setminus \{k, i\}$ there are exactly $s - 1$

elements such that $a_u x_u = \ell(a, x)$. In particular, this gives

$$\ell(a', x) = \max_{u \in [n]} \{(a_i + 1)x_i, (a_k - 1)x^k, R\} = R = \ell(a, x).$$

Therefore, $a' \in \arg \min_{a \in \mathcal{A}} \ell(a, x)$ and the number of elements such that $a'_i x_i = \ell(a', x) = \ell(a, x)$ is at most $s - 1$, which contradicts the fact that s is minimal given the RAS choice and Lemma C.3.3.

As a conclusion we have $a_k x^k \leq (a_i + 1)x_i$. □

Remark C.4.2. Recall that Lemma C.4.1 guarantees that k_i defined in (C.3) satisfy: $k_i \in \{1, 2\}$ for each $i \in [n]$.

C.4.1 Proof of Theorem 4.6.1

Below we restate the theorem.

Theorem 4.6.1. Suppose that Assumption 4.3.1 holds. Let $\bar{a} \in \arg \min_{a \in \mathcal{A}} \ell(a)$, in case of multiple optimal actions, we consider the one output by RAS when fed with μ . Then, the expected regret of ATA with inputs (B, α) satisfies

$$\mathcal{R}_K \leq 2n \max_{i \in [n]} \{B\mu_i - \ell(\bar{a})\} + c \cdot \sum_{i=1}^n \frac{\alpha^2 (\bar{a}_i + k_i)(B\mu_i - \ell(\bar{a}))}{((\bar{a}_i + k_i)\mu_i - \ell(\bar{a}))^2} \cdot \ln K,$$

where $\alpha = \max_{i \in [n]} \|X_i - \mu_i\|_{\psi_1}$, and c is a constant.

Proof. Let K_i^k be the number of rounds where arm i was queried prior to round k (we take $K_i^1 = 0$). Recall that we chose the following confidence bound: if $K_i^k \geq 1$, then

$$\text{conf}(i, k) = 2\alpha \sqrt{\frac{\ln(2k^2)}{K_i^k}} + 2\alpha \frac{\ln(2k^2)}{K_i^k},$$

and $\text{conf}(i, k) = \infty$ otherwise. Recall that $\hat{\mu}_i^k$ denotes the empirical mean of samples from ν_i observed prior to k if $K_i^k \geq 0$ and $\hat{\mu}_i^k = 0$ if $K_i^k = 0$. Let s_i^k denote the lower confidence bound used in the algorithm:

$$s_i^k = (\hat{\mu}_i^k - \text{conf}(i, k))_+.$$

We introduce the events \mathcal{E}_i^k for $i \in [n]$ and $k \in [K]$ defined by

$$\mathcal{E}_i^k := \{|\hat{\mu}_i^k - \mu_i| > \text{conf}(i, k)\}.$$

Let

$$\mathcal{E}_k = \bigcup_{i \in [n]} \mathcal{E}_i^k.$$

Let us prove that for each $k \in [K]$ and $i \in [n]$: $\mathbb{P}(\mathcal{E}_i^k) \leq 1/k^2$, which gives using a union bound $\mathbb{P}(\mathcal{E}_k) \leq n/k^2$. Let $i \in [n]$, using Lemma C.5.1 and taking $\delta = 1/k^2$, we have

$$\mathbb{P}(\mathcal{E}_i^k) = \mathbb{P}\{|\hat{\mu}_i^k - \mu_i| > \text{conf}(i, k)\} \leq \frac{1}{k^2}.$$

We call a “bad round”, a round k where we have $\ell(a^k) > \ell(\bar{a})$. Let us upper bound the number of bad rounds.

Observe that in a bad round there is necessarily an arm $i \in [K]$ such that $a_i^k \mu_i > \ell(\bar{a})$. For each $i \in [n]$, let $N_i(k)$ denote the number of rounds $q \in \{1, \dots, k\}$ where $a_i^q \mu_i > \ell(\bar{a})$ and $i \in \arg \max_{j \in [n]} a_j^q \mu_j$ (this corresponds to a bad round triggered by worker i)

$$N_i(k) := |\{q \in \{1, \dots, k\} : a_i^q \mu_i > \ell(\bar{a}) \text{ and } a_i^q \mu_i = \ell(a_q)\}| .$$

We show that in the case of $\ell(a^k) > \ell(\bar{a})$, the following event will hold: there exists $i \in [n]$ such that

$$E_{i,k} := \mathcal{E}_k \text{ or } \left\{ N_i(k-1) \leq \frac{24\alpha^2(\bar{a}_i + k_i) \ln(2K^2)}{((\bar{a}_i + k_i)\mu_i - \ell(\bar{a}))^2} \right\} .$$

To prove this we use a contradiction argument. Suppose that for each $i \in [n]$, $\neg E_{i,k}$ holds and that $\ell(a^k) > \ell(\bar{a})$. This means that k is a bad round, let i be an arm that triggered this bad round (i.e., $i \in \arg \max_{j \in [n]} a_j^k \mu_j$). Event $\neg E_{i,k}$ gives in particular

$$N_i(k-1) > \frac{24\alpha^2(\bar{a}_i + k_i) \ln(2K^2)}{((\bar{a}_i + k_i)\mu_i - \ell(\bar{a}))^2} . \quad (\text{C.4})$$

Observe that in each round where $N_i(\cdot)$ is incremented, the number of samples received from the distribution ν_i increases by at least $\bar{a}_i + k_i$. Therefore, we have (C.4) implies

$$K_i^k > \frac{24\alpha^2(\bar{a}_i + k_i)^2 \ln(2K^2)}{((\bar{a}_i + k_i)\mu_i - \ell(\bar{a}))^2} = \frac{24\alpha^2 \ln(2K^2)}{\left(\mu_i - \frac{\ell(\bar{a})}{\bar{a}_i + k_i}\right)^2} .$$

Then we have, using the expressions of $\text{conf}(\cdot)$ and the bound above

$$\begin{aligned} 2\text{conf}(i, k) &= 4\alpha \sqrt{\frac{\ln(2k^2)}{K_i^k}} + 4\alpha \frac{\ln(2k^2)}{K_i^k} \\ &\leq \mu_i - \frac{\ell(\bar{a})}{\bar{a}_i + k_i} . \end{aligned} \quad (\text{C.5})$$

The contradiction hypothesis gives that $a_i^k \mu_i > \ell(\bar{a})$, then we have, using the definition of k_i that $a_i^k \geq \bar{a}_i + k_i$. Therefore, (C.5) gives

$$2\text{conf}(i, k) < \mu_i - \frac{\ell(\bar{a})}{a_i^k} . \quad (\text{C.6})$$

Observe that in each round $\|a^k\|_1 = B$, therefore if we have $a_i^k \geq \bar{a}_i + k_i > \bar{a}_i$ for some i , we necessarily have that there exists $j \in [n] \setminus \{i\}$ such that $a_j^k \leq \bar{a}_j - 1$. Using the fact that $\ell(\bar{a}) \geq \bar{a}_j \mu_j$ with (C.6), we get

$$a_i^k (\mu_i - 2\text{conf}(i, k)) > \bar{a}_j \mu_j . \quad (\text{C.7})$$

Since both $\neg\mathcal{E}_i^k$ and $\neg\mathcal{E}_{j,k}$ hold (because $\neg E_{i,k}$ implies $\neg\mathcal{E}_k$), we have that

$$\mu_i - 2\text{conf}(i, k) \leq \hat{\mu}_{i,k} - \text{conf}(i, k) \leq s_i^k , \quad (\text{C.8})$$

and $\mu_j \geq \hat{\mu}_{j,k} - \text{conf}(j, k)$. Recall that $\mu_j \geq 0$, therefore

$$\mu_j \geq (\hat{\mu}_{j,k} - \text{conf}(j, k))_+ = s_{j,k} . \quad (\text{C.9})$$

Using the bounds (C.8) and (C.9) in (C.7), we have

$$a_i^k s_i^k > \bar{a}_j s_{j,k} \geq (a_j^k + 1) s_{j,k} ,$$

where we used the definition of j in the second inequality. This contradicts the statement of Lemma C.4.1, which concludes the contradiction argument. Therefore, the event that k is a bad round implies that $E_{i,k}$ holds for at least one $i \in [n]$. We say that a bad round was triggered by arm i , a round where $N_i(\cdot)$ was incremented. Observe that if $k \in [K]$ is not a bad round then $\mathbb{E} [\ell(a^k)] - \ell(\bar{a}) = 0$, otherwise if k is a bad round triggered by $i \in [n]$ then

$$\mathbb{E} [\ell(a^k)] - \ell(\bar{a}) \leq B\mu_i - \ell(\bar{a}) .$$

To ease notation we introduce for $i \in [n]$

$$H_i := \frac{24\alpha^2(\bar{a}_i + k_i) \ln(2K^2)}{((\bar{a}_i + k_i)\mu_i - \ell(\bar{a}))^2} .$$

We denote by $\mathbb{1}(\cdot)$ the indicator function, which takes the value 1 if the condition inside the braces is true and 0 otherwise.

The expected regret satisfies

$$\begin{aligned}
\mathcal{R}_K &= \sum_{i=1}^K \mathbb{E} [\ell(a^k) - \ell(\bar{a})] \\
&\leq \sum_{i=1}^n (B\mu_i - \ell(\bar{a})) \mathbb{E}[N_i(K)] \\
&= \sum_{i=1}^n \sum_{k=1}^K (B\mu_i - \ell(\bar{a})) \mathbb{E} [\mathbb{1}(k \text{ is a bad round triggered by } i)] \\
&\leq \max_{i \in [n]} \{(B\mu_i - \ell(\bar{a}))\} \cdot \sum_{t=1}^K \mathbb{P}(\mathcal{E}_k) \\
&\quad + \sum_{i=1}^n (B\mu_i - \ell(\bar{a})) \sum_{k=1}^K \mathbb{E} [\mathbb{1}(k \text{ is a bad round triggered by } i) \mid \neg \mathcal{E}_k] \\
&\leq \max_{i \in [n]} \{(B\mu_i - \ell(\bar{a}))\} \cdot \sum_{t=1}^K \mathbb{P}(\mathcal{E}_k) \\
&\quad + \sum_{i=1}^n (B\mu_i - \ell(\bar{a})) \sum_{k=1}^K \mathbb{E} [\mathbb{1}(N_i(k) = 1 + N_i(k-1) \text{ and } N_i \leq H_i) \mid \neg \mathcal{E}_k] \\
&\leq \max_{i \in [n]} \{(B\mu_i - \ell(\bar{a}))\} \cdot \sum_{k=1}^K \mathbb{P}(\mathcal{E}_k) + \sum_{i=1}^n (B\mu_i - \ell(\bar{a})) H_i \\
&\leq 2n \max_{i \in [n]} \{(B\mu_i - \ell(\bar{a}))\} + \sum_{i=1}^n \frac{24\alpha^2(\bar{a}_i + k_i)(B\mu_i - \ell(\bar{a})) \ln(2K^2)}{((\bar{a}_i + k_i)\mu_i - \ell(\bar{a}))^2}.
\end{aligned}$$

□

C.4.2 Proof of Theorem 4.6.3

We restate below the theorem that provides the regret bound for ATA-Empirical and then present its proof.

Theorem 4.6.3. Suppose that Assumption 4.3.1 holds. Let $\bar{a} \in \arg \min_{a \in \mathcal{A}} \ell(a)$, in case of multiple optimal actions, we consider the one output by RAS when fed with μ . Then, the expected regret of ATA-Empirical with the empirical confidence bounds using the inputs (B, η) satisfies

$$\mathcal{R}_K \leq 2n \max_{i \in [n]} \{B\mu_i - \ell(\bar{a})\} + c \cdot \sum_{i=1}^n \frac{\eta^2 \mu_i^2 (\bar{a}_i + k_i)(B\mu_i - \ell(\bar{a}))}{((\bar{a}_i + k_i)\mu_i - \ell(\bar{a}))^2} \cdot \ln K,$$

where $\eta = \max_{i \in [n]} \alpha_i / \mu_i$, and c is a constant.

Proof. We build on the techniques used in the proof of Theorem 4.6.1. Recall the expression of η :

$$\eta = \max_{i \in [n]} \frac{\alpha_i}{\mu_i}.$$

Define the quantities $C_{i,k}$ by

$$C_{i,k} = 2\sqrt{\frac{\ln(2k^2)}{K_i^k}} + 2\frac{\ln(2k^2)}{K_i^k} .$$

Recall that the lower confidence bounds used here are defined as

$$\hat{s}_i^k = \hat{\mu}_i^k (1 - \eta C_{i,k})_+ .$$

We additionally define the following quantities

$$\hat{u}_{i,k} := \hat{\mu}_i^k \left(1 + \frac{4}{3}\eta C_{i,k} \right) .$$

We introduce the events \mathcal{E}_i^k for $i \in [n]$ and $k \in [K]$ defined by

$$\mathcal{E}_i^k := \left\{ \mu_i < \hat{s}_i^k \right\} \quad \text{or} \quad \left\{ \eta C_{i,k} \leq \frac{1}{4} \quad \text{and} \quad \mu_i > \hat{u}_{i,k} \right\} .$$

Let

$$\mathcal{E}_k = \bigcup_{i \in [n]} \mathcal{E}_i^k .$$

We have, using Lemma C.5.3, for each $k \in [K]$ and $i \in [n]$: $\mathbb{P}(\mathcal{E}_i^k) \leq 1/k^2$, which gives using a union bound $\mathbb{P}(\mathcal{E}_k) \leq n/k^2$.

Following similar steps as in the proof of Theorem 4.6.1, we call a “bad round”, a round k where we have $\ell(a^k) > \ell(\bar{a})$. Let us upper bound the number of bad rounds.

Observe that in a bad round there is necessarily an arm $i \in [K]$ such that $a_i^k \mu_i > \ell(\bar{a})$. For each $i \in [n]$, let $N_i(k)$ denote the number of rounds $q \in \{1, \dots, k\}$ where $a_i^q \mu_i > \ell(\bar{a})$ and $i \in \arg \max_{j \in [n]} \{a_j^q \mu_j\}$ (this corresponds to a bad round triggered by worker q):

$$N_i(k) := |\{q \in \{1, \dots, k\} : a_i^q \mu_i > \ell(\bar{a}) \text{ and } a_i^q \mu_i = \ell(a_q)\}| .$$

We show that in the case of $\ell(a^k) > \ell(\bar{a})$, the following event will hold: there exists $i \in [n]$ such that

$$E_{i,k} := \mathcal{E}_k \text{ or } \left\{ N_i(k-1) \leq \frac{185\eta^2 \mu_i^2 (\bar{a}_i + k_i) \ln(2K^2)}{((\bar{a}_i + k_i)\mu_i - \ell(\bar{a}))^2} \right\} .$$

To prove this, suppose for a contradiction argument that we have for some $i \in [n]$: $\neg E_{i,k}$ and that k is a bad round triggered by arm i (i.e., $\ell(a^k) > \ell(\bar{a})$ and $i \in \arg \max_{j \in [n]} a_j^k \mu_j$).

This gives in particular

$$N_i(k-1) > \frac{185\eta^2 \mu_i^2 (\bar{a}_i + k_i) \ln(2K^2)}{((\bar{a}_i + k_i)\mu_i - \ell(\bar{a}))^2} . \quad (\text{C.10})$$

Observe that in each round where $N_i(\cdot)$ is incremented, the number of samples received from the distribution ν_i increases by at least $\bar{a}_i + k_i$. Therefore, we have

(C.10) implies

$$K_i^k > \frac{185\eta^2\mu_i^2(\bar{a}_i + k_i)^2 \ln(2K^2)}{((\bar{a}_i + k_i)\mu_i - \ell(\bar{a}))^2} = \frac{185\eta^2\mu_i^2 \ln(2K^2)}{\left(\mu_i - \frac{\ell(\bar{a})}{\bar{a}_i + k_i}\right)^2}.$$

Therefore, we have, using the expression of $C_{i,k}$ and the bound above

$$\begin{aligned} C_{i,k} &= 2\sqrt{\frac{\ln(2k^2)}{K_i^k}} + 2\frac{\ln(2k^2)}{K_i^k} \\ &\leq \frac{3}{19\eta\mu_i} \left(\mu_i - \frac{\ell(\bar{a})}{\bar{a}_i + k_i}\right). \end{aligned} \quad (\text{C.11})$$

The last bound implies in particular that $\eta C_{i,k} \leq 3/19$, hence $(1 - \eta C_{i,k})_+ = 1 - \eta C_{i,k}$.

We have

$$\begin{aligned} \hat{\mu}_i^k - \hat{s}_i^k &= \hat{\mu}_i^k (1 - (1 - \eta C_{i,k})_+) \\ &\leq \eta C_{i,k} \hat{\mu}_i^k \\ &\leq \eta C_{i,k} \frac{\mu_i}{1 - \eta C_{i,k}} \\ &\leq \frac{1}{5} \left(\mu_i - \frac{\ell(\bar{a})}{\bar{a}_i + k_i}\right), \end{aligned} \quad (\text{C.12})$$

where we used the event $\neg \mathcal{E}_i^k$ in the penultimate inequality (in particular $\hat{s}_i^k = \hat{\mu}_i^k (1 - \eta C_{i,k})_+ \leq \mu_i$), and the bound (C.11) in the last inequality.

Given the hypothesis that $\ell(a^k) > \ell(\bar{a})$ and, $\ell(a^k) = a_i^k \mu_i$, we necessarily have $a_i^k \geq \bar{a}_i + k_i$. Therefore, bound (C.12)

$$5\hat{\mu}_i^k - 5\hat{s}_i^k < \mu_i - \frac{\ell(\bar{a})}{a_i^k}.$$

Observe that in each round $\|a^k\|_1 = B$, therefore if we have $a_i^k \geq \bar{a}_i + k_i > \bar{a}_i$ for some i , we necessarily have that there exists $j \in [n] \setminus \{i\}$ such that $a_j^k \leq \bar{a}_j - 1$. Therefore, using the fact that $\ell(\bar{a}) \geq \bar{a}_j \mu_j$, we obtain

$$a_i^k (\mu_i + 5\hat{s}_i^k - 5\hat{\mu}_i^k) > \ell(\bar{a}) \geq \bar{a}_j \mu_j. \quad (\text{C.13})$$

Since both $\neg \mathcal{E}_i^k$ and $\neg \mathcal{E}_{j,k}$ hold (because $\neg E_{i,k}$ implies $\neg \mathcal{E}_k$), we have that

$$\begin{aligned} \mu_i + 5\hat{s}_i^k - 5\hat{\mu}_i^k &= \hat{s}_i^k + \mu_i - \hat{\mu}_i^k + 4(\hat{s}_i^k - \hat{\mu}_i^k) \\ &= \hat{s}_i^k + \mu_i - \hat{\mu}_i^k + 4\hat{\mu}_i^k ((1 - \eta C_{i,k})_+ - 1) \\ &\leq \hat{s}_i^k + \mu_i - \hat{\mu}_i^k - 4\hat{\mu}_i^k \eta C_{i,k} \\ &\leq \hat{s}_i^k + \mu_i - \hat{u}_{i,k}, \end{aligned}$$

To conclude, observe that given $\eta C_{i,k} \leq 3/19$, event $\neg \mathcal{E}_i^k$ implies that $\mu_i \leq \hat{u}_{i,k}$, therefore

$$\mu_i + 5\hat{s}_i^k - 5\hat{\mu}_i^k \leq \hat{s}_i^k.$$

Since $\neg\mathcal{E}_{j,k}$ holds, we also have

$$\mu_j \geq \hat{s}_{j,k} .$$

Using the two last bounds in (C.13), we have

$$a_i^k \hat{s}_i^k > \bar{a}_j \hat{s}_{j,k} \geq (a_j^k + 1) \hat{s}_{j,k} ,$$

where we used the definition of j , as an arm satisfying $\bar{a}_j \geq 1 + a_j^k$, in the second inequality. This contradicts the statement of Lemma C.4.1, which concludes the contradiction argument. Therefore, the event that k is a bad round implies that $E_{i,k}$ holds for at least one $i \in [n]$. We say that a bad round was triggered by arm i , a round where $N_i(\cdot)$ was incremented. Observe that if $k \in [K]$ is not a bad round then $\mathbb{E}[\ell(a^k)] - \ell(\bar{a}) = 0$, otherwise if k is a bad round triggered by $i \in [n]$ then $\mathbb{E}[\ell(a^k)] - \ell(\bar{a}) \leq B\mu_i - \ell(\bar{a})$. To ease notation we introduce for $i \in [n]$

$$H_i := \frac{185\eta^2\mu_i^2(\bar{a}_i + k_i)\ln(2K^2)}{((\bar{a}_i + k_i)\mu_i - \ell(\bar{a}))^2} .$$

The expected regret satisfies

$$\begin{aligned} \mathcal{R}_K &= \sum_{i=1}^K \mathbb{E}[\ell(a^k) - \ell(\bar{a})] \\ &\leq \sum_{i=1}^n (B\mu_i - \ell(\bar{a})) \mathbb{E}[N_i(K)] \\ &= \sum_{i=1}^n \sum_{k=1}^K (B\mu_i - \ell(\bar{a})) \mathbb{E}[\mathbf{1}(k \text{ is a bad round triggered by } i)] \\ &\leq \max_{i \in [n]} \{(B\mu_i - \ell(\bar{a}))\} \cdot \sum_{t=1}^K \mathbb{P}(\mathcal{E}_k) \\ &\quad + \sum_{i=1}^n (B\mu_i - \ell(\bar{a})) \sum_{k=1}^K \mathbb{E}[\mathbf{1}(k \text{ is a bad round triggered by } i) \mid \neg\mathcal{E}_k] \\ &\leq \max_{i \in [n]} \{(B\mu_i - \ell(\bar{a}))\} \cdot \sum_{t=1}^K \mathbb{P}(\mathcal{E}_k) \\ &\quad + \sum_{i=1}^n (B\mu_i - \ell(\bar{a})) \sum_{k=1}^K \mathbb{E}[\mathbf{1}(N_i(k) = 1 + N_i(k-1) \text{ and } N_i \leq H_i) \mid \neg\mathcal{E}_k] \\ &\leq \max_{i \in [n]} \{(B\mu_i - \ell(\bar{a}))\} \cdot \sum_{k=1}^K \mathbb{P}(\mathcal{E}_k) + \sum_{i=1}^n (B\mu_i - \ell(\bar{a})) H_i \\ &\leq 2n \max_{i \in [n]} \{(B\mu_i - \ell(\bar{a}))\} + \sum_{i=1}^n \frac{185\eta^2\mu_i^2(\bar{a}_i + k_i)(B\mu_i - \ell(\bar{a}))\ln(2K^2)}{((\bar{a}_i + k_i)\mu_i - \ell(\bar{a}))^2} . \end{aligned}$$

□

C.4.3 Proof of Theorem 4.4.2

Let us first restate the theorem.

Theorem 4.4.2. Suppose Assumption 4.3.1 holds and let $\eta := \max_{i \in [n]} \alpha_i / \mu_i$, where $\alpha_i = \|X_i - \mu_i\|_{\psi_1}$. Then, the total expected computation time after K rounds, using the allocation prescribed by ATA with inputs (B, α) satisfies

$$\mathcal{C}_K \leq (1 + 4\eta \ln(B)) \mathcal{C}_K^* + \mathcal{O}(\ln K).$$

Proof. Let \mathbb{E}_k be the expectation with respect to the variables observed up to and including k and \mathcal{F}_k the corresponding filtration. Using the tower rule, we have

$$\sum_{k=1}^K \mathbb{E}[C(a^k)] = \mathbb{E}\left[\sum_{k=1}^K \mathbb{E}_{k-1}[C(a^k)]\right].$$

Consider round $k \in [K]$, let us upper bound $\mathbb{E}_{k-1}[C(a_t)]$ using $\mathbb{E}_{k-1}[\ell(a^k)]$. Recall that $a^k \in \mathcal{F}_{k-1}$, let $Y_i = \sum_{u=1}^{a_i^k} X_{i,k}^{(u)}$, since Y_i is the sum of a_i^k i.i.d samples we have that $\mathbb{E}_{k-1}[Y_i] = a_i^k \mu_i$ and $\|Y_i - a_i^k \mu_i\|_{\psi_1} \leq a_i^k \|X_i - \mu_i\|_{\psi_1}$. Thus, using Lemma C.5.4, we get

$$\begin{aligned} \mathbb{E}_{k-1}[C(a^k)] &= \mathbb{E}_{k-1}\left[\max_{i \in \text{supp}(a^k)} \sum_{u=1}^{a_i^k} X_{i,k}^{(u)}\right] \\ &\leq \max_{i \in \text{supp}(a^k)} a_i^k \mu_i + 4 \max_{i \in \text{supp}(a^k)} a_i^k \alpha_i \cdot \ln(B) \\ &\leq \max_{i \in \text{supp}(a^k)} a_i^k \mu_i + 4 \max_{i \in \text{supp}(a^k)} a_i^k \eta \mu_i \cdot \ln(B) \\ &= (1 + 4\eta \ln(B)) \max_{i \in \text{supp}(a^k)} a_i^k \mu_i. \end{aligned}$$

Moreover, using Jensen's inequality, we have

$$\max_{i \in [n]} a_i^* \mu_i \leq \mathbb{E}\left[\max_{i \in [n]} \sum_{u=1}^{a_{k,i}} X_{i,k}^{(u)}\right] = \mathbb{E}[C(a^*)].$$

Using the last two bounds with the result of Theorem 4.6.1, we get the result. \square

C.5 Technical Lemmas

The lemma below gives a concentration bound on sub-exponential variables. Note that this result can be inferred from Proposition 7 in the work of Maurer & Pontil (2021), although applying the last result directly requires assuming the variables are positive, this is not needed in their proof in the one dimensional case. For completeness, we present the full proof below.

Lemma C.5.1. Let Y_1, \dots, Y_n be iid random variables with $\mathbb{E}[Y_1] = 0$ and $\alpha = \|Y_1\|_{\psi_1} < +\infty$. Then for any $\delta \in (0, 1)$, we have with probability at least

$$1 - \delta$$

$$\left| \frac{1}{n} \sum_{i=1}^n Y_i \right| \leq 2\alpha \left(\sqrt{\frac{\ln(2/\delta)}{n}} + \frac{\ln(2/\delta)}{2n} \right) .$$

Proof. Let $v := 2n\alpha^2$. We have using Lemma C.5.5 that

$$\sum_{i=1}^n \mathbb{E}[Y_i^2] \leq 2n\alpha^2 \quad \text{and} \quad \sum_{i=1}^n \mathbb{E}[(Y_i)_+^q] \leq \frac{q!}{2} v \alpha^{q-2} .$$

Therefore, using Bernstein concentration inequality (Proposition C.5.2) we obtain that

$$\mathbb{P}\left(\left|\sum_{i=1}^n Y_i\right| \geq 2\alpha\sqrt{nt} + \alpha t\right) \leq 2 \exp(-t) .$$

Choosing $t = \ln(2/\delta)$, we obtain the result. \square

Proposition C.5.2 (Theorem 2.10 in the work of Boucheron et al. (2005)). *Let X_1, \dots, X_n be independent real-valued random variables. Assume there exist positive numbers v and c such that*

$$\sum_{i=1}^n \mathbb{E}[X_i^2] \leq v \quad \text{and} \quad \sum_{i=1}^n \mathbb{E}[(X_i)_+^q] \leq \frac{q!}{2} v c^{q-2} , \quad \text{for all integers } q \geq 3 ,$$

where $x_+ := \max\{x, 0\}$. Define the centered sum

$$S := \sum_{i=1}^n (X_i - \mathbb{E}X_i).$$

Then, for every $t > 0$,

$$\mathbb{P}(S \geq \sqrt{2vt} + ct) \leq e^{-t} .$$

Lemma C.5.3. Let X_1, \dots, X_n be i.i.d positive random variables with mean μ and $\|X_1\|_{\psi_1} < +\infty$. Denote $\alpha := \|X_1 - \mu\|_{\psi_1}$. Denote $\hat{X}_n = \frac{1}{n} \sum_{i=1}^n X_i$ and let $\delta \in (0, 1)$. Define $\eta, C_{\cdot, \cdot}$ by:

$$\eta := \frac{\alpha}{\mu} \quad \text{and} \quad C_{n,\delta} := 2\sqrt{\frac{\ln(2/\delta)}{n}} + 2\frac{\ln(2/\delta)}{n} .$$

Then with probability at least $1 - \delta$ we have

$$\mu \geq \hat{X}_n(1 - \eta \cdot C_{n,\delta})_+ ,$$

where we use the notation $(a)_+ = \max\{0, a\}$. Moreover, if $\eta \cdot C_{n,\delta} \leq 1/4$, then with probability least $1 - \delta$

$$\hat{X}_n(1 - \eta \cdot C_{n,\delta})_+ \leq \mu \leq \hat{X}_n \left(1 + \frac{4}{3}\eta \cdot C_{n,\delta}\right) .$$

Proof. Fix n, δ . We work on the event

$$\mathcal{E}_{n,\delta} = \left\{ |\hat{X}_n - \mu| \leq \alpha \cdot C_{n,\delta} \right\}$$

that holds with probability at least $1 - \delta$ if we apply Lemma C.5.1 to $X_i - \mu$.

Proof of $\mu \geq \hat{X}_n(1 - \eta \cdot C_{n,\delta})_+$:

If $\eta C_{n,\delta} \geq 1$, we have $(1 - \eta \cdot C_{n,\delta})_+ = 0$ and the result follows from the fact that X is non-negative which gives $\mu \geq 0$.

Suppose now that $\eta C_{n,\delta} < 1$. Recall that event $\mathcal{E}_{n,\delta}$ implies that

$$\hat{X}_n \leq \mu(1 + \eta C_{n,\delta}) .$$

Therefore, we have

$$\frac{\hat{X}_n}{1 + \eta C_{n,\delta}} \leq \mu .$$

Using $1 - \eta C_{n,\delta} \leq \frac{1}{1 + \eta C_{n,\delta}}$ with the bound above, we obtain

$$\hat{X}_n(1 - \eta C_{n,\delta})_+ = \hat{X}_n(1 - \eta C_{n,\delta}) \leq \frac{\hat{X}_n}{1 + \eta \cdot C_{n,\delta}} \leq \mu .$$

Proof of $\mu \leq (1 + \frac{4}{3}\eta \cdot C_{n,\delta})$. Recall that event $\mathcal{E}_{n,\delta}$ gives

$$\hat{X}_n \geq \mu - \alpha C_{n,\delta} = \mu(1 - \eta C_{n,\delta}) .$$

Suppose that $\eta C_{n,\delta} \leq 1/4$. We therefore have

$$\mu \leq \frac{\hat{X}_n}{1 - \eta C_{n,\delta}} .$$

Next, we use the fact that for any $x \in [0, 1/4]$, we have

$$\frac{1}{1 - x} \leq 1 + \frac{4}{3}x ,$$

which gives

$$\mu \leq \hat{X}_n \left(1 + \frac{4}{3}\eta \cdot C_{n,\delta} \right) ,$$

when $\eta C_{n,\delta} \leq 1/4$. □

The following lemma provides a useful bound on the expectation of the maximum of independent sub-exponential random variables.

Lemma C.5.4. Let X_1, \dots, X_n be a sequence of independent random variables with finite Orlicz norm $\|X_i\|_{\psi_1} < +\infty$ and let $\mathbb{E}[X_i] = \mu_i$. Then we have

$$\mathbb{E} \left[\max_{i \in [n]} X_i \right] \leq \max_{i \in [n]} \mu_i + 4\alpha \ln(n) ,$$

where $\alpha = \max_{i \in [n]} \|X_i - \mu_i\|_{\psi_1}$.

Proof. If $n = 1$ the bound is straightforward, suppose that $n \geq 2$. Let $Y_i := X_i - \mu_i$, then $\alpha = \max_{i \in [n]} \|Y_i\|_{\psi_1}$. Let $\lambda \in (0, 1/\alpha)$, we have

$$\begin{aligned}\max_{i \in [n]} Y_i &= \frac{1}{\lambda} \ln \left(\exp \left(\lambda \max_{i \in [n]} Y_i \right) \right) \\ &\leq \frac{1}{\lambda} \ln \left(\sum_{i=1}^n \exp (\lambda Y_i) \right) .\end{aligned}$$

Taking the expectation and using Lemma C.5.5, we have

$$\begin{aligned}\mathbb{E} \left[\max_{i \in [n]} Y_i \right] &\leq \frac{1}{\lambda} \ln \left(\sum_{i=1}^n \mathbb{E} [\exp (\lambda Y_i)] \right) \\ &\leq \frac{1}{\lambda} \ln \left(\frac{n}{1 - \lambda \alpha} \right) .\end{aligned}$$

We choose

$$\lambda = \frac{1 - 1/n}{\alpha} ,$$

which gives

$$\begin{aligned}\mathbb{E} \left[\max_{i \in [n]} Y_i \right] &\leq \frac{\alpha}{1 - \frac{1}{n}} \ln(n^2) \\ &= 2\alpha \frac{n}{n-1} \ln(n) \\ &\leq 4\alpha \ln(n) .\end{aligned}\tag{C.14}$$

Let $i^* \in \arg \max_{i \in [n]} X_i$, we have

$$\begin{aligned}\max_{i \in [n]} X_i - \max_{i \in [n]} \mu_i &= X_{i^*} - \max_{i \in [n]} \mu_i \\ &\leq X_{i^*} - \mu_{i^*} \leq \max_{i \in [n]} (X_i - \mu_i) = \max_{i \in [n]} Y_i .\end{aligned}$$

Combining the last bound with (C.14) we obtain

$$\mathbb{E} \left[\max_{i \in [n]} X_i \right] \leq \max_{i \in [n]} \mu_i + 4\alpha \ln(n) .$$

□

Lemma below is based on a standard argument we give here for completeness.

Lemma C.5.5. Let Y be a variable such that $\alpha = \|Y\|_{\psi_1} < +\infty$. Then we have for any $\lambda \in (-1/\alpha, 1/\alpha)$

$$\mathbb{E} [\exp (\lambda Y)] \leq \frac{1}{1 - |\lambda| \alpha} .$$

Moreover, we have for any $q \geq 3$

$$\mathbb{E}[Y^2] \leq 2\alpha^2 \quad \text{and} \quad \mathbb{E}[(Y)_+^q] \leq \frac{q!}{2} \cdot (2\alpha^2) \cdot \alpha^{q-2} .$$

Proof. Let $Z = |Y|/\alpha$. First observe that we have

$$\sum_{k \geq 0} \frac{\mathbb{E}[Z^k]}{k!} = \mathbb{E}[\exp(Z)] = \mathbb{E}\left[\exp\left(\frac{|Y|}{\alpha}\right)\right] \leq 2 ,$$

so,

$$\sum_{k \geq 1} \frac{\mathbb{E}[Z^k]}{k!} \leq 1 .$$

This implies

$$\mathbb{E}[|Y|^k] \leq k! \alpha^k$$

for all $k \geq 1$. Using this bound, we estimate the moment generating function

$$\begin{aligned} \mathbb{E}[\exp(\lambda Y)] &\leq \mathbb{E}[\exp(|\lambda| |Y|)] \\ &= \sum_{k \geq 0} \frac{|\lambda|^k \mathbb{E}[|Y|^k]}{k!} \\ &\leq 1 + \sum_{k \geq 1} |\lambda|^k \alpha^k \\ &= \frac{1}{1 - |\lambda| \alpha} . \end{aligned}$$

The remaining bounds follow from

$$\mathbb{E}[|Y|^k] \leq k! \alpha^k .$$

□

Khalas