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

Artavazd Maranjyan, Peter Richtárik
King Abdullah University of Science and Technology (KAUST)
arto.maranjyan@gmail.com, richtarik@gmail.com

Asynchronous stochastic gradient methods are central to scalable distributed optimization, particularly when devices differ in computational capabilities. Such settings arise naturally in federated learning, where training takes place on smartphones and other heterogeneous edge devices. In addition to varying computation speeds, these devices often hold data from different distributions. However, existing asynchronous SGD methods struggle in such heterogeneous settings and face two key limitations. First, many rely on unrealistic assumptions of similarity across workers' data distributions. Second, methods that relax this assumption still fail to achieve theoretically optimal performance under heterogeneous computation times. We introduce Ringleader ASGD, the first asynchronous SGD algorithm that attains the theoretical lower bounds for parallel first-order stochastic methods in the smooth nonconvex regime, thereby achieving optimal time complexity under data heterogeneity and without restrictive similarity assumptions. Our analysis further establishes that Ringleader ASGD remains optimal under arbitrary and even time-varying worker computation speeds, closing a fundamental gap in the theory of asynchronous optimization.

1. Introduction

Modern machine learning increasingly depends on large-scale distributed training across clusters with hundreds or even thousands of GPUs [1–3]. However, classical synchronous training methods struggle to scale in these settings, as device failures, network instabilities, and synchronization overheads introduce significant inefficiencies [4, 5]. These issues become even more pronounced in environments with heterogeneous computational power, such as Federated Learning (FL), where devices range from high-end datacenter GPUs to resource-constrained edge hardware [6–9]. 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 datacenter 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 [10–14]. 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

Third Conference on Parsimony and Learning (CPAL 2026).

Table 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 (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), the target stationarity ε, the variance bound of the stochastic gradients σ^2 (Assumption 1), and smoothness constants. Specifically, L_f is the smoothness constant of f (Definition 1); $L_{\max} := \max_{i\in[n]} L_{f_i}$ with L_{f_i} the smoothness constant of f_i; and L is a constant associated with our new smoothness-type assumption (Assumption 2). They satisfy $L_f \leq L \leq L_{\max}$ (Lemma 1). 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 and Richtárik [15] 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.

Method	Time Complexity	Optimal	No sync.	No idle workers	No discarded work
Naive Minibatch SGD (Section 3.1)	$\frac{L_f\Delta}{\varepsilon}\left(\tau_n + \tau_n\frac{\sigma^2}{n\varepsilon}\right)$	✘	✘	✘	✔
IA^{2}SGD [16] (Appendix F)	$\frac{L_{\max}\Delta}{\varepsilon}\left(\tau_n + \tau_n\frac{\sigma^2}{n\varepsilon}\right)$ (†)	✘	✔	✔	✔
Malenia SGD [15]	$\frac{L_f\Delta}{\varepsilon}\left(\tau_n + \tau_{\text{avg}}\frac{\sigma^2}{n\varepsilon}\right)$	✔	✘	✔	✘
Ringleader ASGD **(new)** (Algorithm 1; Theorem 2)	$\frac{L\Delta}{\varepsilon}\left(\tau_n + \tau_{\text{avg}}\frac{\sigma^2}{n\varepsilon}\right)$	✔(‡)	✔	✔	✔
Lower Bound [15]	$\frac{L_f\Delta}{\varepsilon}\left(\tau_n + \tau_{\text{avg}}\frac{\sigma^2}{n\varepsilon}\right)$	—	—	—	—

(†) The analysis presented by Wang et al. [16] 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 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, $L = \mathcal{O}(L_f)$.

question: *among all parallel methods, which ones are provably fastest in theory?* To make this question precise, we restrict our attention to smooth nonconvex 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 and Richtárik [15] 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 and Richtárik [15] 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. [17], who, in the *homogeneous data setting*, introduced Ringmaster ASGD—the first asynchronous SGD method to achieve the same optimal time complexity as the synchronous Ren-

nala 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. [17] 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 [8, 18, 19]. 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 [20–23]. 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. [16] took an important step toward removing these restrictive assumptions by proposing Incremental Aggregated Asynchronous SGD (IA^{2}SGD), a method that provably converges without similarity assumptions. However, their method achieves the same time complexity as standard Naive Minibatch SGD (see the first two rows of Table 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 and Richtárik [15], 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. [17].

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 complexity[1] in the *heterogeneous data setting*. Importantly, Ringleader ASGD attains this without relying on restrictive similarity assumptions.

1.1. Contributions

Our main contributions are the following:

- **Optimal asynchronous SGD under data heterogeneity.** We prove that Ringleader ASGD (Algorithm 1) is, to the best of our knowledge, the first asynchronous method in the heterogeneous setting under the fixed computation model (2) matching the lower bounds for parallel methods of Tyurin and Richtárik [15], when the smoothness-type constant L in Assumption 2 is within a constant factor of the smoothness L_f used to obtain the lower bounds (Table 1). Importantly, Ringleader ASGD attains this without any similarity assumptions across clients' data.
- **Additional useful properties.** Beyond achieving optimal time complexity, our method Ringleader ASGD satisfies two additional desirable properties: (i) all workers remain con-

[1] Throughout the paper, we refer to our method as optimal. Formally, this holds whenever the constant L—associated with our new smoothness-type assumption (Assumption 2)—is at most a constant factor larger than the smoothness constant L_f used in the derived lower bounds [15]. See Table 1 for details.

tinuously 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 1 compares Ringleader ASGD against benchmark algorithms with respect to these properties.

- **Parameter-free design.** In contrast to the optimal synchronous method Malenia SGD [15], 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 improvement can also be extended to Malenia SGD, as we discuss in Appendix G.
- **Universal computation model.** In Appendix D, 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 6, 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.

2. 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:

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

Here $f_i\colon \mathbb{R}^d \to \mathbb{R}$ 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 $\mathcal{D}_i$.

2.1. 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 Appendix D.

Following the *fixed computation model* [20], we formalize:

$$\begin{aligned} &\text{Each worker } i \text{ requires } \tau_i \text{ seconds}^2 \text{ to compute one stochastic gradient } \nabla f_i(x,\xi_i)\,. \\ &\text{Without loss of generality, assume } 0 < \tau_1 \le \tau_2 \le \cdots \le \tau_n\,. \end{aligned} \tag{2}$$

We assume communication is infinitely fast (taking 0 seconds), both from workers to the server and from the server to workers[3]. This is a modeling choice—arguably the simplest one—and has been the standard assumption in prior work [15, 17, 20, 21], even if not always stated explicitly. We further discuss the motivation and limitations of this abstraction in Appendix C. A related study by Tyurin et al. [24] 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.

[2] One 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.

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

2.2. 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 $[n] := \{1, 2, \ldots, n\}$ to denote the index set, and $\mathbb{E}\left[\cdot\right]$ for mathematical expectation. For functions $\phi, \psi : \mathcal{Z} \to \mathbb{R}$, we write $\phi = \mathcal{O}(\psi)$ if there exists a constant $C > 0$ such that $\phi(z) \leq C\psi(z)$ for all $z \in \mathcal{Z}$.

2.3. Assumptions

We consider the following standard assumptions for the nonconvex setting.

Assumption 1. For each $i \in [n]$ and every ξ, the function $f_i(x; \xi)$ 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,

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

Assumption 2. Each function f_i is differentiable. There exists a constant $L > 0$ such that, for all $x \in \mathbb{R}^d$ and $y_1, \ldots, y_n \in \mathbb{R}^d$,

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

Recall the standard definition of smoothness

Definition 1 (Smoothness)**.** A differentiable function $\phi\colon \mathbb{R}^d \to \mathbb{R}$ is called L_ϕ–smooth if

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

By convention, L_ϕ denotes the smallest such constant.

Note that Assumption 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 1 (Smoothness Bounds; Proof in Appendix E.1)**.** Suppose Assumption 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 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, including works by Wang et al. [16], Koloskova et al. [21], Nguyen et al. [22].

Assumption 3. 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 $\mathbb{E}\left[\|\nabla f(x)\|^2\right] \leq \varepsilon$.

3. 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 Malenia SGD [15], the first synchronous SGD method to attain optimal time complexity. We then turn to the challenges of asynchronous approaches, focusing on the recent IA^{2}SGD [16] and its limitations. Finally, we outline how these limitations can be addressed and introduce the core idea behind our algorithm.

3.1. Naive Minibatch SGD

Naive Minibatch SGD provides the most straightforward approach to solving problem (1) in a distributed setting.

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

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

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 $\tau_n := \max_{i \in [n]} \tau_i$ (2). Consequently, faster workers remain idle while waiting for stragglers, which leads to inefficient utilization of available computational resources.

Theoretical Performance. From a convergence perspective, Naive Minibatch SGD achieves the iteration complexity $\mathcal{O}\left(L_f \Delta / \varepsilon \left(1 + \sigma^2 / n\varepsilon\right)\right)$ to reach an ε–stationary point [25–27]. 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.

3.2. Malenia SGD

Malenia SGD [15] addresses the straggler problem of 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

$$\left(\frac{1}{n} \sum_{i=1}^{n} \frac{1}{b_i^k}\right)^{-1} \geq \max\left\{1, \frac{\sigma^2}{n\varepsilon}\right\} \tag{3}$$

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

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

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) 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) 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 and Richtárik [15]. 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). 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 [11].

3.3. 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 [20–23], 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) [28–30] and Stochastic Averaged Gradient (SAG) [31, 32], which maintain and aggregate gradients from all workers.

3.4. IA^{2}SGD

As discussed above, the key insight is to construct a gradient estimator using information from all workers at each model update. IA^{2}SGD [16] 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 $\nabla f_{i_k}\left(x^{k-\delta_{i_k}^k}; \xi_{i_k}^{k-\delta_{i_k}^k}\right)$ from worker i_k
2. Update the gradient table entry: $g_{i_k} = \nabla f_{i_k}\left(x^{k-\delta_{i_k}^k}; \xi_{i_k}^{k-\delta_{i_k}^k}\right)$
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. [16], and by a straightforward conversion to the fixed computation time model (see Appendix F), 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.

3.5. 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 [17], 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 IA^{2}SGD. 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.

4. Ringleader ASGD

We now formally introduce our algorithm, Ringleader ASGD.

Ringleader ASGD builds upon three key insights from existing methods. First, inspired by IA^{2}SGD, we maintain a gradient table[4] to ensure that information from all workers is incorporated into every update, which eliminates the need for data similarity assumptions between workers. Second, following

[4]It is not strictly necessary to maintain a table on the server. An alternative, as in IA^{2}SGD [16], is to store one vector on each worker along with an additional vector on the server. However, this can be problematic

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 communication is instantaneous, i.e., takes zero time (Section 2.1). 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 1.

4.1. 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 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.

when workers have limited memory. For clarity and simplicity, we adopt the server-side table formulation in our description.

Algorithm 1 Ringleader ASGD (server algorithm)

1: **Input:** Stepsize $\gamma > 0$, initial point $x^0 \in \mathbb{R}^d$
2: **Initialization:** Broadcast x^0 to all workers, which then start running Algorithm 2 in parallel
3: Set $k = 0$, $S = \emptyset$; initialize $G_i = 0$, $b_i = 0$ for all $i \in [n]$
4: **while** True **do**
5: **— Phase 1: await stochastic gradients from all workers —**
6: **while** $S \neq [n]$ **do**
7: Receive stochastic gradient g_j^k (computed 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: $x^{k+1} = x^k - \gamma \frac{1}{n} \sum_{i=1}^n {}^{G_i}/_{b_i}$ ◇ Update using averaged gradients from all workers
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 for the next round
15: **while** $S \neq \emptyset$ **do**
16: Receive stochastic gradient g_j^k (computed 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: $x^{k+1} = x^k - \gamma \frac{1}{n} \sum_{i=1}^n {}^{G_i}/_{b_i}$
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 for next round
24: **end if**
25: **end while**
26: $G_i = G_i^+$; $b_i = b_i^+$ for all $i \in [n]$; $S = S^+$ ◇ Transfer buffered gradients to main table
27: **end while**

Algorithm 2 Worker i's subroutine

1: **Input:** Model x
2: **while** True **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**

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.

4.2. 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 2 (Bounded Delay)**.** In Ringleader ASGD (Algorithm 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 $0, n, 2n, 3n, \ldots$, 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 $0, 1, \ldots, n-1$, 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 $j \in \{1, 2, \ldots, n\}$, 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 $\delta_i^n \in \{0, 1, \ldots, n-1\}$. 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 $\delta_i^{2n-1} \in \{n-1, n, \ldots, 2n-2\}$ 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 $\{0, 1, \ldots, n-1\}$.

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 $\{0, 1, \ldots, n-1\}$. During each Phase 2, these delays increase by at most $n-1$, giving the bound

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

□

4.3. Comparison to IA^{2}SGD

Our method is a delay-controlled version of IA^{2}SGD. We can recover IA^{2}SGD 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 IA^{2}SGD lacks.

In IA^{2}SGD, 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.

4.4. 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) 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), in Appendix G 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 (2) while becoming parameter-free, which eliminates the need for prior knowledge of σ and ε. Under 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.

5. 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.

5.1. Iteration Complexity

Our convergence analysis follows the structured approach employed by Maranjyan et al. [17], 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 5.

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 (Descent Lemma; Proof in Appendix E.3)**.** Under Assumptions 1 and 2, if the stepsize in Algorithm 1 satisfies $\gamma \leq 1/4L$, then the following inequality holds

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

This descent lemma shares a similar structure with its counterpart in the homogeneous setting analyzed by Maranjyan et al. [17], 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. [17].

Next, similar to the work by Maranjyan et al. [17], we derive a lemma to bound the term involving the difference between current and old points.

Lemma 4 (Residual Estimation; Proof in Appendix E.4)**.** Under Assumption 1, the iterates of Ringleader ASGD (Algorithm 1) with stepsize $\gamma \leq 1/4nL$ satisfy the following bound

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

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

Theorem 1 (Iteration Complexity)**.** Under Assumptions 1, 2, and 3, let the stepsize in Ringleader ASGD (Algorithm 1) be

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

Then,

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

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 over K iterations and dividing both sides by $\gamma/2$

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

We now bound the third term on the right using Lemma 4

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

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

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

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

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.

5.2. 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 5. Each block of n consecutive iterations (each round) of Algorithm 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. □

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

Theorem 2. Let Assumptions 2, 3, and 1 hold. Let the stepsize in Ringleader ASGD (Algorithm 1) be $\gamma = \min\{1/8nL, \varepsilon B/10L\sigma^2\}$. Then, under the *fixed computation model* (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 1

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

The time to do n steps is at most $2\tau_n$ form Lemma 5. 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 5. □

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 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.

Remark 5.1. *The optimality claim for Theorem 2 holds when the smoothness-type constant L in Assumption 2 is within a constant factor of the smoothness L_f used to derive the lower bound [15] (Table 1).*

Under this condition, the resulting time complexity matches the lower bound of Tyurin and Richtárik [15], making Ringleader ASGD the first asynchronous algorithm to achieve optimality under heterogeneous data.

6. Experiments

To validate our theoretical results we perform a toy simulation.

We consider image classification on MNIST [33] and on Fashion-MNIST [34] 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 $\alpha = 0.1$ [35]. For each client $j \in [n]$, we draw proportions $p_j \sim \mathrm{Dirichlet}(\alpha, \ldots, \alpha)$ 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 $\mathrm{Linear}(d, 128) \to \mathrm{ReLU} \to \mathrm{Linear}(128, 10)$ trained with mean cross-entropy. Stochastic gradients at the clients use a minibatch 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:

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

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.

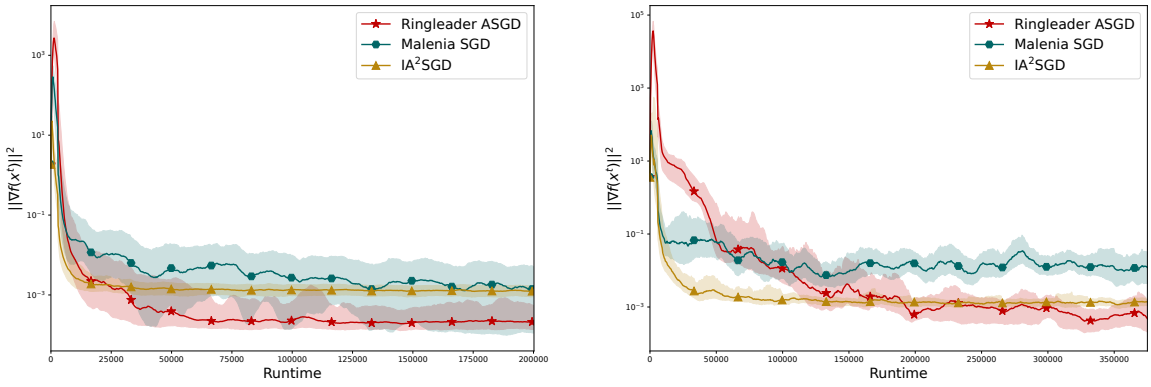
We report the full-batch gradient-norm squared $\|\nabla f(x^k)\|^2$, versus wall-clock time. Each method is run 30 times over different seeds. We report the *median* with *interquartile range* (*IQR*). To reduce high-frequency noise, we apply a centered moving-average smoothing to the aggregated curves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

Figure 1 shows the results. We observe that Ringleader ASGD converges faster compared to Malenia SGD and IA^{2}SGD. Although theory suggests that Ringleader ASGD and Malenia SGD have the same time complexity, in practice Ringleader ASGD benefits from the n updates performed in Phase 2 instead of one synchronous update. This design enables more optimization steps within the same wall-clock budget, which is especially advantageous when updates are sparse.

7. 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



(a) MNIST (b) Fashion-MNIST

Figure 1: Convergence comparison showing median gradient norm squared $\|\nabla f(x^k)\|^2$ (solid lines) with interquartile ranges (shaded regions) versus wall-clock time, averaged over 30 random seeds. **Setup:** Two-layer MLP with architecture $\text{Linear}(d, 128) \to \text{ReLU} \to \text{Linear}(128, 10)$ trained on (a) MNIST and (b) Fashion-MNIST datasets. **Client delays:** Heterogeneous delays simulated as $\tau_i = i + |\eta_i|$ where $\eta_i \sim \mathcal{N}(0, i)$ for client $i \in [n]$, where we choose $n = 100$. **Results:** With optimally tuned stepsizes, Ringleader ASGD achieves faster convergence than both Malenia SGD and IA^{2}SGD, despite Ringleader ASGD and Malenia SGD having equivalent time complexity guarantees.

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 bounds established by Tyurin and Richtárik [15]. 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 (Appendix D).

Acknowledgments

The research reported in this publication was supported by funding from King Abdullah University of Science and Technology (KAUST): i) KAUST Baseline Research Scheme, ii) CRG Grant ORFS-CRG12-2024-6460, and iii) Center of Excellence for Generative AI, under award number 5940.

References

[1] Mohammad Shoeybi, Mostofa 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.

[2] 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, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

[3] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa 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*, pages 1–15, 2021.

[4] Jianmin Chen, Xinghao Pan, Rajat Monga, Samy Bengio, and Rafal Jozefowicz. Revisiting distributed synchronous SGD. *arXiv preprint arXiv:1604.00981*, 2016.

[5] 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.

[6] 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*, 2016.

[7] 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.

[8] 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, 2020.

[9] 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.

[10] 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.

[11] 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.

[12] Alekh Agarwal and John C Duchi. Distributed delayed stochastic optimization. *Advances in Neural Information Processing Systems*, 24, 2011.

[13] 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, editors, *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.

[14] 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.

[15] 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.

[16] 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.

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

[18] 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.

[19] 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.

[20] 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, 2022.

[21] 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.

[22] 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*, pages 3581–3607. PMLR, 2022.

[23] Rustem Islamov, Mher Safaryan, and Dan Alistarh. AsGrad: A sharp unified analysis of asynchronous-SGD algorithms. In *International Conference on Artificial Intelligence and Statistics*, pages 649–657. PMLR, 2024.

[24] 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, 2024.

[25] 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.

[26] 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.

[27] 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*, pages 5200–5209. PMLR, 2019.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] Mark Schmidt, Nicolas Le Roux, and Francis Bach. Minimizing finite sums with the stochastic average gradient. *Mathematical Programming*, 162:83–112, 2017.

[33] Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

[34] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[35] 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, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 7252–7261. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/yurochkin19a.html.

[36] 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.

[37] Shen-Yi Zhao and Wu-Jun Li. Fast asynchronous parallel stochastic gradient descent: A lock-free approach with convergence guarantee. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

[38] 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.

[39] 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.

[40] Lam Nguyen, Phuong Ha Nguyen, Marten Dijk, Peter Richtárik, Katya Scheinberg, and Martin Takác. SGD and Hogwild! convergence without the bounded gradients assumption. In *International Conference on Machine Learning*, pages 3750–3758. PMLR, 2018.

[41] Kaiwen Zhou, Fanhua Shang, and James Cheng. A simple stochastic variance reduced algorithm with fast convergence rates. In *International Conference on Machine Learning*, pages 5980–5989. PMLR, 2018.

[42] Xiangru Lian, Yijun Huang, Yuncheng Li, and Ji Liu. Asynchronous parallel stochastic gradient for nonconvex optimization. *Advances in Neural Information Processing Systems*, 28, 2015.

[43] 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*, pages 803–812. PMLR, 2018.

[44] 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.

[45] 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 *International Conference on Machine Learning*, pages 4120–4129. PMLR, 2017.

[46] Yossi Arjevani, Ohad Shamir, and Nathan Srebro. A tight convergence analysis for stochastic gradient descent with delayed updates. In *Algorithmic Learning Theory*, pages 111–132. PMLR, 2020.

[47] 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.

[48] 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.

[49] Margalit R Glasgow and Mary Wootters. Asynchronous distributed optimization with stochastic delays. In *International Conference on Artificial Intelligence and Statistics*, pages 9247–9279. PMLR, 2022.

[50] Cong Xie, Sanmi Koyejo, and Indranil Gupta. Asynchronous federated optimization. *arXiv preprint arXiv:1903.03934*, 2019.

[51] 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*, 2022.

[52] 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, 2022.

[53] 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.

[54] 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*, pages 41399–41413. PMLR, 2023.

[55] 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.

[56] 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.

[57] 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.

[58] 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, 2024.

[59] Alexander Tyurin and Peter Richtárik. On the optimal time complexities in decentralized stochastic asynchronous optimization. *Advances in Neural Information Processing Systems*, 37, 2024.

[60] 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*, 2025. URL https://openreview.net/forum?id=RNpvu3MSvm.

[61] 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*, pages 1273–1282. PMLR, 2017.

[62] Artavazd Maranjyan, Mher Safaryan, and Peter Richtárik. GradSkip: Communication-accelerated local gradient methods with better computational complexity. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856. URL https://openreview.net/forum?id=6R3fRqFfhn.

[63] 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*, 2025.

[64] Alexander Tyurin. Tight time complexities in parallel stochastic optimization with arbitrary computation dynamics. *arXiv preprint arXiv:2408.04929*, 2024.

[65] Yurii Nesterov. *Lectures on Convex Optimization*, volume 137. Springer, 2018.

A. Appendix

B. Related Work

Research on asynchronous stochastic gradient methods dates back to the seminal work of Tsitsiklis et al. [10], and gained renewed momentum with the introduction of the HOGWILD! algorithm [11]. 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 works by J Reddi et al. [36], Zhao and Li [37], Mania et al. [38], Leblond et al. [39], Nguyen et al. [40], Zhou et al. [41], 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 and Duchi [12], who studied convex objectives, as well as later extensions to the non-convex case such as the work of Lian et al. [42] and Dutta et al. [43], the latter analyzing exponentially distributed computation times. Other relevant results in this line include those of Feyzmahdavian et al. [44], Zheng et al. [45], Arjevani et al. [46], Feyzmahdavian and Johansson [47], all of which assume fixed delays. More recently, delay-adaptive methods have been proposed, aiming to improve performance by down-weighting very stale gradients [20, 21, 48].

Particularly relevant to our work are asynchronous variants of SAGA. Leblond et al. [39] developed a shared-parameter version in the spirit of HOGWILD!, while Glasgow and Wootters [49] 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 works by Mishchenko et al. [20], Koloskova et al. [21], Islamov et al. [23], Glasgow and Wootters [49], Xie et al. [50], Wang et al. [51, 52], Fraboni et al. [53], Zhang et al. [54], Wang et al. [55], Alahyane et al. [56].

More broadly, Assran et al. [57] provide a comprehensive survey of asynchronous optimization methods.

There is another line of work that began with Tyurin and Richtárik [15], 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 [24, 58–60].

Finally, beyond asynchronous approaches, several synchronous methods address system heterogeneity by adapting local training to worker speeds. The canonical method, FedAvg [61], performs multiple local steps on each worker. Variants have adapted the number of local steps to match workers' computation speeds [8, 62], effectively balancing task assignments across heterogeneous systems. More recently, Maranjyan et al. [63] proposed adapting the number of local steps dynamically, without requiring prior knowledge of worker speeds.

C. The Computation-Only Model and the Role of Asynchrony

This section clarifies the scope of our modeling assumptions and the specific phenomenon our results target. Asynchrony is designed to eliminate *waiting time*: the idle time that arises when workers have heterogeneous computation speeds or experience straggling behavior due to hardware stalls, load imbalance, or even network delays. A key point is that the goal of asynchrony is *not* to reduce communication cost, but to ensure that slow or delayed workers do not force faster workers to remain idle.

Critically, even in the *simplest* setting—homogeneous data and *no* communication cost—it was unknown until very recently whether an asynchronous SGD method could match the optimal synchronous rate. In fact, existing asynchronous methods were shown to be *worse* than the optimal synchronous algorithm in this basic regime [15]. The recent work of Maranjyan et al. [17] resolved this foundational case for the first time by showing that, under homogeneous data, an asynchronous method can achieve the same optimal time complexity as the synchronous method Rennala SGD [15]. Our work extends this understanding to the significantly more challenging heterogeneous-data setting and shows that asynchrony *can* solve the problem it is designed for—and that it can do so *optimally*.

Asynchrony and stragglers. Stragglers may be caused by slow computation, device variability, or even *communication delays*. From the server's perspective, a worker whose gradient arrives late because it is still communicating appears identical to one that is slow at computing. Asynchrony ensures that such delays—regardless of source—do not block progress: fast workers continue contributing updates while slow or delayed workers catch up. This ability to eliminate idle time is precisely the purpose of asynchronous methods.

Asynchrony does *not* reduce communication cost. Although asynchrony prevents waiting during communication-induced delays, it does *not* reduce the communication overhead itself. Reducing communication cost requires *orthogonal techniques* such as gradient compression, sparsification, quantization, or local-update schemes (e.g., FedAvg [7]). For example, Tyurin et al. [24] explicitly study communication-aware training and use compression-based methods to reduce communication time—demonstrating that communication efficiency is a *separate algorithmic axis* that must be combined with, rather than replaced by, asynchrony. Thus, asynchrony resolves the *waiting problem* caused by delays, but not the *communication-cost problem*; addressing the latter requires additional mechanisms.

Why communication is not modeled explicitly here. For these reasons, we adopt the standard computation-only model used in essentially all theoretical works on asynchronous SGD [15, 17, 20, 21, 59], which is also the model under which the lower bounds of Tyurin and Richtárik [15] are derived. Our claims of *optimal time complexity* therefore refer to this shared and well-established model. Studying asynchrony under explicit communication cost—where it must interact with compression, local updates, or buffering—requires new lower bounds and a different theoretical framework, and is beyond the scope of this work.

D. Arbitrarily Changing Computation Times

In practice, the *fixed computation model* (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 [60].

In this section we extend our theory to the more general setting of arbitrarily varying computation times.

D.1. Universal Computation Model

To formalize this setting, we adopt the *universal computation model* introduced by Tyurin [64].

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

$$p_i : \mathbb{R}_+ \to \mathbb{R}_+ \,,$$

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* (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.

D.2. Toward an Optimal Method

In the general setting of arbitrarily varying computation times, Algorithm 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 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 1, 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 1) cannot be optimal in the arbitrary-time setting.

D.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 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

$$\left(\frac{1}{n}\sum_{i=1}^{n}\frac{1}{b_i}\right)^{-1} \;\geq\; \max\left\{1, \frac{\sigma^2}{n\varepsilon}\right\}\,, \tag{3}$$

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 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 3. Under Assumptions 1, 2, and 3, 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^K seconds, where

$$K \;:=\; \left\lceil \frac{160L\Delta}{\varepsilon} \right\rceil,$$

and T^K denotes the K-th element of the recursively defined sequence

$$T^k \;=\; \min\left\{ T \geq 0 : \left(\frac{1}{n}\sum_{i=1}^{n}\left\lfloor \int_{T_{k-1}}^{T} p_i(t)\,dt \right\rfloor^{-1}\right)^{-1} \;\geq\; \max\left\{1, \frac{\sigma^2}{n\varepsilon}\right\}\right\}\,,$$

for all $k \geq 1$, with initialization $T^0 = 0$.

This result matches the lower bound derived by Tyurin [64], and therefore the proposed method is optimal.

Proof. Under the condition in (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 1, 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

$$K = \frac{80\,nL\Delta}{\varepsilon}\,.$$

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), which occurs at time

$$T_+^1 = \min\left\{T \geq 0 : \left(\frac{1}{n}\sum_{i=1}^n \left\lfloor \int_0^T p_i(t)\,dt \right\rfloor^{-1}\right)^{-1} \;\geq\; \max\left\{1, \frac{\sigma^2}{n\varepsilon}\right\}\right\}.$$

After Phase 1, to complete n updates in Phase 2 we must wait for the ongoing computations to finish. This requires at most

$$T^1 = \min\left\{T \geq 0 : \left(\frac{1}{n}\sum_{i=1}^n \left\lfloor \int_{T_+^1}^T p_i(t)\,dt \right\rfloor^{-1}\right)^{-1} \;\geq\; 1\right\}.$$

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

$$T^k = \min\left\{T \geq 0 : \left(\frac{1}{n}\sum_{i=1}^n \left\lfloor \int_{T_{k-1}}^T p_i(t)\,dt \right\rfloor^{-1}\right)^{-1} \;\geq\; \max\left\{1, \frac{\sigma^2}{n\varepsilon}\right\}\right\}, \qquad T^0 = 0\,.$$

□

E. Auxiliary Lemmas

Here we provide proofs of lemmas omitted from the main text, along with auxiliary results that will be used later.

E.1. Proof of Lemma 1

We begin with a lemma relating the different smoothness constants.

Lemma 1 (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 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 2 that we assumed the following generalized smoothness condition: for some constant $L > 0$ and for all $x \in \mathbb{R}^d$ and $y_1, \ldots, y_n \in \mathbb{R}^d$,

$$\left\|\nabla f(x) - \frac{1}{n}\sum_{i=1}^n \nabla f_i(y_i)\right\|^2 \;\leq\; \frac{L^2}{n}\sum_{i=1}^n \|x - y_i\|^2\,. \tag{4}$$

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 $y_1 = \cdots = y_n = y$. Then (4) 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,

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

Squaring both sides shows that (4) 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

$$\left\|\nabla f(x) - \frac{1}{n}\sum_{i=1}^{n}\nabla f(y_i)\right\| \leq \frac{1}{n}\sum_{i=1}^{n}\|\nabla f(x) - \nabla f(y_i)\| \leq \frac{L_f}{n}\sum_{i=1}^{n}\|x - y_i\|$$
$$\leq L_f\sqrt{\frac{1}{n}\sum_{i=1}^{n}\|x - y_i\|^2}\,,$$

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., (4) holds with $L \leq L_f$. Combined with $L_f \leq L$, we conclude $L = L_f$. □

E.2. Variance Term

The following lemma bounds the variance of the gradient estimator in Ringleader ASGD.

Lemma 6 (Variance Bound)**.** Under Assumption 1, the following variance-type inequality holds for the gradient estimator used in Algorithm 1:

$$\mathbb{E}\left[\left\|\bar{g}^k - \frac{1}{n}\sum_{i=1}^{n}\nabla f_i\left(x^{k-\delta_i^k}\right)\right\|^2\right] \leq \frac{\sigma^2}{B^k n}\,.$$

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

$$\mathbb{E}\left[\left\|\bar{g}^k - \frac{1}{n}\sum_{i=1}^{n}\nabla f_i\left(x^{k-\delta_i^k}\right)\right\|^2\right] = \mathbb{E}\left[\mathbb{E}\left[\left\|\bar{g}^k - \frac{1}{n}\sum_{i=1}^{n}\nabla f_i\left(x^{k-\delta_i^k}\right)\right\|^2 \,\middle|\, \mathcal{F}^k\right]\right]$$
$$= \mathbb{E}\left[\frac{1}{n^2}\sum_{i=1}^{n}\mathbb{E}\left[\left\|\bar{g}_i^k - \nabla f_i\left(x^{k-\delta_i^k}\right)\right\|^2 \,\middle|\, \mathcal{F}^k\right]\right].$$

For each worker i, the conditional variance of the minibatch gradient estimator is

$$\mathbb{E}\left[\left\|\bar{g}_i^k - \nabla f_i\left(x^{k-\delta_i^k}\right)\right\|^2 \;\middle|\; \mathcal{F}^k\right] = \mathbb{E}\left[\left\|\frac{1}{b_i^k}\sum_{j=1}^{b_i^k} g_i^{k,j} - \nabla f_i\left(x^{k-\delta_i^k}\right)\right\|^2 \;\middle|\; \mathcal{F}^k\right]$$
$$= \frac{1}{b_i^k}\mathbb{E}\left[\left\|g_i^{k,1} - \nabla f_i\left(x^{k-\delta_i^k}\right)\right\|^2 \;\middle|\; \mathcal{F}^k\right] \leq \frac{\sigma^2}{b_i^k},$$

where the last inequality follows from Assumption 1.

Combining these results, we get

$$\mathbb{E}\left[\left\|\bar{g}^k - \frac{1}{n}\sum_{i=1}^n \nabla f_i\left(x^{k-\delta_i^k}\right)\right\|^2\right] \leq \frac{1}{n^2}\sum_{i=1}^n \frac{\sigma^2}{b_i^k} = \frac{\sigma^2}{n}\frac{1}{n}\sum_{i=1}^n \frac{1}{b_i^k} = \frac{\sigma^2}{B^k n},$$

where the last equality uses the definition of the harmonic mean $B^k = \left(\frac{1}{n}\sum_{i=1}^n \frac{1}{b_i^k}\right)^{-1}$. □

E.3. Proof of Lemma 3

We now prove the descent lemma.

Lemma 3 (Descent Lemma)**.** Under Assumptions 1 and 2, if the stepsize in Algorithm 1 satisfies $\gamma \leq 1/4L$, then the following inequality holds

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

Proof. Some proof techniques are adapted from the works of Maranjyan et al. [17] and Wang et al. [16].

From Assumption 2 and Lemma 1, we know that f is L–smooth. Therefore, the following standard inequality holds [65]

$$\mathbb{E}\left[f(x^{k+1})\right] \leq \mathbb{E}\left[f(x^k) - \gamma\left\langle\nabla f(x^k), \bar{g}^k\right\rangle + \frac{L\gamma^2}{2}\left\|\bar{g}^k\right\|^2\right]. \tag{5}$$

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:

$$\mathbb{E}\left[\bar{g}^k \mid \mathcal{F}^k\right] = \frac{1}{n}\sum_{i=1}^n \frac{1}{b_i^k}\sum_{j=1}^{b_i^k}\mathbb{E}\left[g_i^{k,j} \;\middle|\; \mathcal{F}^k\right] = \frac{1}{n}\sum_{i=1}^n \nabla f_i\left(x^{k-\delta_i^k}\right).$$

The last equality follows from the unbiasedness of the stochastic gradient estimator (Assumption 1).

Using this conditional expectation and the law of total expectation, we can now simplify the inner product term in (5):

$$\begin{aligned}
\mathbb{E}\left[\left\langle \nabla f\left(x^k\right), \bar{g}^k\right\rangle\right] &= \mathbb{E}\left[\left\langle \nabla f\left(x^k\right), \frac{1}{n}\sum_{i=1}^n \nabla f_i\left(x^{k-\delta_i^k}\right)\right\rangle\right] \\
&\quad + \mathbb{E}\left[\left\langle \nabla f\left(x^k\right), \bar{g}^k - \frac{1}{n}\sum_{i=1}^n \nabla f_i\left(x^{k-\delta_i^k}\right)\right\rangle\right] \\
&= \mathbb{E}\left[\left\langle \nabla f\left(x^k\right), \frac{1}{n}\sum_{i=1}^n \nabla f_i\left(x^{k-\delta_i^k}\right)\right\rangle\right] \\
&\quad + \mathbb{E}\left[\left\langle \nabla f\left(x^k\right) - \nabla f\left(x^{k-(k \bmod n)}\right), \bar{g}^k - \frac{1}{n}\sum_{i=1}^n \nabla f_i\left(x^{k-\delta_i^k}\right)\right\rangle\right] \\
&\quad + \mathbb{E}\left[\left\langle \nabla f\left(x^{k-(k \bmod n)}\right), \bar{g}^k - \frac{1}{n}\sum_{i=1}^n \nabla f_i\left(x^{k-\delta_i^k}\right)\right\rangle\right] \\
&= \underbrace{\mathbb{E}\left[\left\langle \nabla f\left(x^k\right), \frac{1}{n}\sum_{i=1}^n \nabla f_i\left(x^{k-\delta_i^k}\right)\right\rangle\right]}_{T_1} \\
&\quad + \underbrace{\mathbb{E}\left[\left\langle \nabla f\left(x^k\right) - \nabla f\left(x^{k-(k \bmod n)}\right), \bar{g}^k - \frac{1}{n}\sum_{i=1}^n \nabla f_i\left(x^{k-\delta_i^k}\right)\right\rangle\right]}_{T_2}.
\end{aligned}$$

Next, using Assumption 2, we have

$$\begin{aligned}
2T_1 &= \mathbb{E}\left[2\left\langle \nabla f\left(x^k\right), \frac{1}{n}\sum_{i=1}^n \nabla f_i\left(x^{k-\delta_i^k}\right)\right\rangle\right] \\
&= \mathbb{E}\left[\left\|\nabla f\left(x^k\right)\right\|^2 + \left\|\frac{1}{n}\sum_{i=1}^n \nabla f_i\left(x^{k-\delta_i^k}\right)\right\|^2\right] - \mathbb{E}\left[\left\|\nabla f\left(x^k\right) - \frac{1}{n}\sum_{i=1}^n \nabla f_i\left(x^{k-\delta_i^k}\right)\right\|^2\right] \\
&\geq \mathbb{E}\left[\left\|\nabla f\left(x^k\right)\right\|^2\right] + \mathbb{E}\left[\left\|\frac{1}{n}\sum_{i=1}^n \nabla f_i\left(x^{k-\delta_i^k}\right)\right\|^2\right] - \frac{L^2}{n}\sum_{i=1}^n \mathbb{E}\left[\left\|x^k - x^{k-\delta_i^k}\right\|^2\right].
\end{aligned}$$

Next, we analyze T_2

$$
\begin{aligned}
T_2 &= \mathbb{E}\left[\left\langle \nabla f\left(x^k\right) - \nabla f\left(x^{k-(k \bmod n)}\right), \bar{g}^k - \frac{1}{n}\sum_{i=1}^n \nabla f_i\left(x^{k-\delta_i^k}\right)\right\rangle\right] \\
&\geq -\mathbb{E}\left[\left\|\nabla f\left(x^k\right) - \nabla f\left(x^{k-(k \bmod n)}\right)\right\|\left\|\bar{g}^k - \frac{1}{n}\sum_{i=1}^n \nabla f_i\left(x^{k-\delta_i^k}\right)\right\|\right] \\
&\geq -L\mathbb{E}\left[\left\|x^k - x^{k-(k \bmod n)}\right\|\left\|\bar{g}^k - \frac{1}{n}\sum_{i=1}^n \nabla f_i\left(x^{k-\delta_i^k}\right)\right\|\right] \\
&= -L\mathbb{E}\left[\left\|\gamma \sum_{\ell=k-(k \bmod n)}^{k-1} \bar{g}^\ell\right\|\left\|\bar{g}^k - \frac{1}{n}\sum_{i=1}^n \nabla f_i\left(x^{k-\delta_i^k}\right)\right\|\right] \\
&\geq -L\gamma \sum_{\ell=k-(k \bmod n)}^{k-1} \mathbb{E}\left[\left\|\bar{g}^\ell\right\|\left\|\bar{g}^k - \frac{1}{n}\sum_{i=1}^n \nabla f_i\left(x^{k-\delta_i^k}\right)\right\|\right] \\
&\geq -L\gamma \sum_{\ell=k-(k \bmod n)}^{k-1} \frac{1}{2}\left(\mathbb{E}\left[\left\|\bar{g}^\ell\right\|^2\right] + \mathbb{E}\left[\left\|\bar{g}^k - \frac{1}{n}\sum_{i=1}^n \nabla f_i\left(x^{k-\delta_i^k}\right)\right\|^2\right]\right) \\
&\geq -\frac{L\gamma}{2} \sum_{\ell=k-(k \bmod n)}^{k-1} \mathbb{E}\left[\left\|\bar{g}^\ell\right\|^2\right] - (k \bmod n)\frac{L\gamma\sigma^2}{2B^k n} \\
&\geq -\frac{L\gamma}{2} \sum_{\ell=k-(k \bmod n)}^{k-1} \mathbb{E}\left[\left\|\bar{g}^\ell\right\|^2\right] - \frac{L\gamma\sigma^2}{2B^k}\,.
\end{aligned}
$$

The inequalities follow from the Cauchy-Schwarz inequality, L–smoothness of f, the triangle inequality, Young's inequality, Lemma 6, and finally $(k \bmod n) \leq n-1 < n$.

It remains to bound the term $\mathbb{E}\left[\left\|\bar{g}^k\right\|^2\right]$. Using Young's inequality, we have

$$
\begin{aligned}
\mathbb{E}\left[\left\|\bar{g}^k\right\|^2\right] &\leq 2\mathbb{E}\left[\left\|\frac{1}{n}\sum_{i=1}^n \nabla f_i\left(x^{k-\delta_i^k}\right)\right\|^2\right] + 2\mathbb{E}\left[\left\|\bar{g}^k - \frac{1}{n}\sum_{i=1}^n \nabla f_i\left(x^{k-\delta_i^k}\right)\right\|^2\right] \\
&\leq 2\mathbb{E}\left[\left\|\frac{1}{n}\sum_{i=1}^n \nabla f_i\left(x^{k-\delta_i^k}\right)\right\|^2\right] + \frac{2\sigma^2}{B^k n}\,,
\end{aligned}
$$

where in the last step we used Lemma 6.

Now, by combining all terms in (5), we obtain

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

This completes the proof under the stepsize condition $\gamma \leq 1/4L$ and $B := \inf_{k\geq 0} B^k$. □

E.4. Proof of Lemma 4

The following lemma provides an upper bound on the residual error due to delays.

Lemma 4 (Residual Estimation). Under Assumption 1, the iterates of Ringleader ASGD (Algorithm 1) with stepsize $\gamma \leq 1/4nL$ satisfy the following bound

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

Proof. By Young's inequality, we have

$$\begin{aligned}
\mathbb{E}\left[\left\|x^k - x^{k-\delta_i^k}\right\|^2\right] &= \mathbb{E}\left[\left\|\gamma \sum_{\ell=k-\delta_i^k}^{k-1} \bar{g}^\ell\right\|^2\right] \\
&\leq 2\gamma^2 \mathbb{E}\left[\left\|\sum_{\ell=k-\delta_i^k}^{k-1} \frac{1}{n}\sum_{j=1}^n \nabla f_j\left(x^{\ell-\delta_j^\ell}\right)\right\|^2\right] \\
&\quad + 2\gamma^2 \mathbb{E}\left[\left\|\sum_{\ell=k-\delta_i^k}^{k-1} \left(\bar{g}^\ell - \frac{1}{n}\sum_{j=1}^n \nabla f_j\left(x^{\ell-\delta_j^\ell}\right)\right)\right\|^2\right] \\
&\leq \underbrace{2\gamma^2\, \delta_i^k \sum_{\ell=k-\delta_i^k}^{k-1} \mathbb{E}\left[\left\|\frac{1}{n}\sum_{j=1}^n \nabla f_j\left(x^{\ell-\delta_j^\ell}\right)\right\|^2\right]}_{T_{ik}} + 2\gamma^2(\delta_i^k)^2 \frac{\sigma^2}{Bn}\,.
\end{aligned}$$

In the last inequality, we used Jensen's inequality and Lemma 6.

Next, we estimate the sum of T_{ik}

$$\begin{aligned}
\sum_{k=0}^{K-1}\frac{1}{n}\sum_{i=1}^n T_{ik} &= \sum_{k=0}^{K-1}\frac{1}{n}\sum_{i=1}^n \delta_i^k \sum_{\ell=k-\delta_i^k}^{k-1} \mathbb{E}\left[\left\|\frac{1}{n}\sum_{j=1}^n \nabla f_j\left(x^{\ell-\delta_j^\ell}\right)\right\|^2\right] \\
&= \frac{1}{n}\sum_{i=1}^n \sum_{k=0}^{K-1} \delta_i^k \sum_{\ell=k-\delta_i^k}^{k-1} \mathbb{E}\left[\left\|\frac{1}{n}\sum_{j=1}^n \nabla f_j\left(x^{\ell-\delta_j^\ell}\right)\right\|^2\right] \\
&\leq 2\sum_{i=1}^n \sum_{k=0}^{K-1} \sum_{\ell=k-\delta_i^k}^{k-1} \mathbb{E}\left[\left\|\frac{1}{n}\sum_{j=1}^n \nabla f_j\left(x^{\ell-\delta_j^\ell}\right)\right\|^2\right] \\
&\leq 2\sum_{i=1}^n \delta_i^{\max} \sum_{k=0}^{K-1} \mathbb{E}\left[\left\|\frac{1}{n}\sum_{j=1}^n \nabla f_j\left(x^{k-\delta_j^k}\right)\right\|^2\right] \\
&\leq 4n^2 \sum_{k=0}^{K-1} \mathbb{E}\left[\left\|\frac{1}{n}\sum_{j=1}^n \nabla f_j\left(x^{k-\delta_j^k}\right)\right\|^2\right].
\end{aligned}$$

In the first and last inequality, we used the bound $\delta_i^{\max} \leq 2n$ from Lemma 2. Finally, applying the stepsize condition $\gamma \leq 1/4nL$ yields the result. □

F. Time Complexity of IA²SGD

The iteration complexity of IA²SGD [16] 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* (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 $\delta^{\max}$. 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.

G. Improved Malenia SGD

Malenia SGD has the following iteration complexity [15]

$$K \;\geq\; \frac{12 \Delta L_f}{\varepsilon} \;+\; \frac{12 \Delta L_f \sigma^2}{\varepsilon^2 n S} \,,$$

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 [15], this bound follows from the condition in (3), which fixes the same value of S across all iterations.

In the fixed-time regime (2), however, this condition is no longer necessary. By adopting the same strategy as Ringleader ASGD (Algorithm 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 5 and proceeding as in the proof of Theorem 2, we obtain the same overall time complexity as before—this time *without* requiring condition (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 Appendix D.