MindFlayer SGD: Efficient Parallel SGD in the Presence of Heterogeneous and Random Worker Compute Times

Artavazd Maranjyan[1] **Omar Shaikh Omar**[1] **Peter Richtárik**[1]

[1]King Abdullah University of Science and Technology, Thuwal, Saudi Arabia

Abstract

We investigate the problem of minimizing the expectation of smooth nonconvex functions in a distributed setting with multiple parallel workers that are able to compute stochastic gradients. A significant challenge in this context is the presence of arbitrarily heterogeneous and stochastic compute times among workers, which can severely degrade the performance of existing parallel stochastic gradient descent (SGD) methods. While some parallel SGD algorithms achieve optimal performance under deterministic but heterogeneous delays, their effectiveness diminishes when compute times are random—a scenario not explicitly addressed in their design. To bridge this gap, we introduce MindFlayer SGD, a novel parallel SGD method specifically designed to handle stochastic and heterogeneous compute times. Through theoretical analysis and empirical evaluation, we demonstrate that MindFlayer SGD consistently outperforms existing baselines, particularly in environments with heavy-tailed noise. Our results highlight its robustness and scalability, making it a compelling choice for large-scale distributed learning tasks.

1 INTRODUCTION

We address the nonconvex optimization problem:

$$\min_{x\in\mathbb{R}^d}\left\{f(x) := \mathbb{E}_{\xi\sim\mathcal{D}}\left[f(x;\xi)\right]\right\}, \tag{1}$$

where $f : \mathbb{R}^d \times \mathbb{S} \to \mathbb{R}$, and ξ is a random variable with some distribution $\mathcal{D}$ on $\mathbb{S}$. In the context of machine learning, $\mathbb{S}$ could represent the space of all possible data, $\mathcal{D}$ denotes the distribution of the training dataset, and $f(\cdot,\xi)$ denotes the loss of a data sample ξ.

The function f is assumed to be differentiable, and its gradient is L–Lipschitz continuous (see Assumptions 5.1–5.2). We assume that we have n workers available to work in parallel, each able to compute independent, unbiased stochastic gradients of f, whose variance is bounded by σ^2 (see Assumption 5.3). In this paper, we study the time complexity of methods working in this setup.

We also assume access to n parallel workers capable of computing independent stochastic gradients, in which case the classical approach is Minibatch SGD [Cotter et al., 2011, Goyal et al., 2017, Gower et al., 2019].

1.1 MINIBATCH SGD

Minibatch SGD awaits the completion of all workers' computations of a single stochastic gradient before executing a gradient-type step:

1. receive a single stochastic gradient $\nabla f(x^k;\xi_i^k)$ from each worker $i \in [n]$,
2. update the model:

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

where $[n] := \{1,\dots,n\}$, $\gamma > 0$ is a stepsize, ξ_i^k are i.i.d. samples from $\mathcal{D}$, and the gradients $\nabla f(x^k;\xi_i^k)$ are calculated in parallel.

In real systems, each worker's computational power may differ from the others, leading to varying completion times of gradient computation. A notable drawback of Minibatch SGD is its failure to account for these differences in compute times across workers. The duration of each step is determined by the slowest worker's computation time. As a result, all other workers remain idle after completing their tasks, waiting for the slowest device to finish. Meanwhile, this idle time could potentially be used in a more efficient way to improve the overall time complexity. Clearly, a redesign of the algorithm is necessary.

1.2 ASYNCHRONOUS SGD

As a result, a new generation of algorithms emerged—asynchronous stochastic gradient descent (ASGD) methods—designed to fully utilize all available computational resources [Recht et al., 2011, Agarwal and Duchi, 2011, Feyzmahdavian et al., 2016, Mania et al., 2017, Nguyen et al., 2018, Arjevani et al., 2020, Cohen et al., 2021, Mishchenko et al., 2022, Koloskova et al., 2022, Islamov et al., 2024, Maranjyan et al., 2025b].

Here, the server performs a gradient-type update immediately after receiving a stochastic gradient from any worker, without waiting for the others. The updated model is then sent back to the worker, which immediately begins computing a new stochastic gradient based on the updated model. By the time the worker finishes computing this gradient, the model may have already been updated multiple times on the server due to gradients received from other workers. This creates a delay in the model update, denoted as δ_k. The algorithm can be described as follows:

1. receive a stochastic gradient $\nabla f(x^{k-\delta_k}; \xi^{k-\delta_k})$ from any worker,
2. update the model:
$$x^{k+1} = x^k - \gamma \nabla f(x^{k-\delta_k}; \xi^{k-\delta_k}),$$
3. send new x^{k+1} to the worker so the worker computes $\nabla f(x^{k+1}; \xi^{k+1})$.

Cohen et al. [2021], Mishchenko et al. [2022], Koloskova et al. [2022] showed that ASGD is provably faster in terms of time complexity than Minibatch SGD.

However, it turns out that this untamed and wild asynchrony can be detrimental. The drawback of ASGD lies in the assumption that all worker computations are beneficial. It suffers from the issue of updating the model with potentially significantly delayed gradients, which ultimately harms convergence and, consequently, the overall time complexity, as discussed in the work of Tyurin and Richtárik [2024]. To address this, there was a need to introduce a method that ignores outdated gradients while still maximizing the utilization of available computational resources.

1.3 RENNALA SGD

Such a method was proposed in a recent breakthrough by Tyurin and Richtárik [2024]. Their method, Rennala SGD, is a semi-asynchronous variant of Minibatch SGD. At each iteration, the server asynchronously collects a batch of gradients, allowing workers to send as many gradients as they can on the same point x^k. Then, using this batch, Rennala SGD proceeds with a gradient-type update using this batch as in Minibatch SGD:

1. wait until the server receives B stochastic gradients at point x^k,
2. update the model:
$$x^{k+1} = x^k - \frac{\gamma}{B} \sum_{j=1}^{B} \nabla f(x^k; \xi_j^k),$$

more details on Rennala SGD are in Appendix D. In this case, the faster the worker, the more gradients it sends. For the struggling workers, it may happen that they are ignored.

Worker Time. Their approach assumes each worker i requires a fixed $\tau_i > 0$ seconds to compute a stochastic gradient. For the first time lower bounds on time complexity were obtained for first order asynchronous methods in the fixed compute time regime for nonconvex functions with Lipschitz gradients. They showed that Rennala SGD is minimax optimal in this setup in terms of time complexity.

Although it may seem like the story ends here, we challenge the fixed time assumption, arguing that a random time model better reflects reality. The claim of optimality no longer holds due to this randomness, suggesting that the algorithms should be reevaluated and redesigned. In this paper, we focus on this redesign, aiming to better align the algorithms with a more realistic model.

2 PROBLEM SETUP AND CONTRIBUTIONS

The deterministic compute time setup considered by Tyurin and Richtárik [2024], where Rennala SGD is optimal, fails to capture the complexities of real-world distributed learning environments. In practice, compute times are often uncertain due to various factors such as failing hardware, preemption by other jobs, delays in GPU computation, and inconsistencies in network communications [Chen et al., 2016, Dutta et al., 2018]. This uncertainty is even more pronounced in federated learning scenarios, where client unreliability can lead to unpredictable computation times or even incomplete tasks [Kairouz et al., 2021].

To address these real-world challenges, we propose a practical setup that incorporates randomness in compute times. Specifically, we consider a scenario where the stochastic gradient computation time of worker i is given by:

$$\tau_i + \eta_i, \tag{2}$$

where $\tau_i > 0$ is a constant representing the minimum time for client i to complete the gradient computation, and η_i is a non-negative random variable drawn from some distribution $\mathcal{J}_i$, modeling the aforementioned uncertainties.

In this more realistic setting, existing methods like Rennala SGD and ASGD can perform poorly or even fail to converge. We can illustrate this with a simple example:

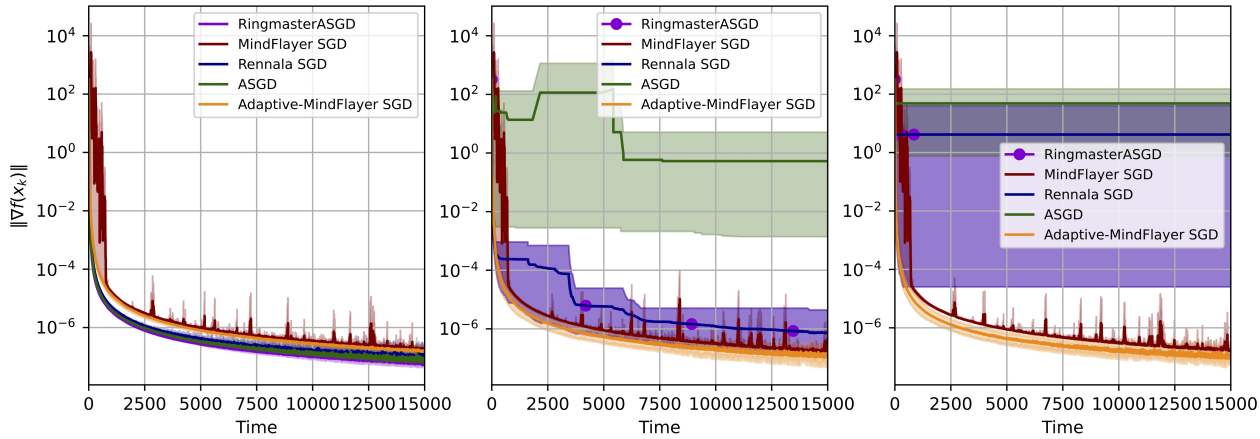

Figure 1: We conducted an empirical evaluation on a quadratic optimization problem (see Appendix A for details on the setup). In this experiment, we modeled the computation time for each worker as $\tau_i + \eta_i$, where $\tau_i = \sqrt{i}$ is a fixed constant and $\eta_i \sim \text{lognormal}(0, s)$ introduces noise for each worker $i \in [n]$. To investigate the effect of increasing noise variance, we tested different values of s: $s = 1$ (left plot), $s = 10$ (middle plot), and $s = 100$ (right plot). As the variance increases, MindFlayer SGD remains robust to the noise in worker computation times, whereas the convergence of the other methods Rennala SGD, ASGD, and Ringmaster ASGD [1], degrade significantly. This highlights how MindFlayer SGD maintains efficiency even in the presence of heavy-tailed noise. Finally, we observe that Adaptive-MindFlayer SGD, which does not assume prior knowledge of the distributions and adjusts the thresholds for each client adaptively based on observed times, performs better than MindFlayer SGD. This improvement comes from its ability to adaptively select batch sizes for each worker based on random times, using a single batch size that is the sum of all individual ones.

Consider a scenario where each time we request a device to compute a stochastic gradient, one of two outcomes occurs. Either the device completes the computation exactly after the minimum time τ without any delays, or something goes wrong and the computation is never completed. This situation can be modeled using a random time η as follows:

$$\eta = \begin{cases} 0, & \text{with probability } 1 - q, \\ \infty^2, & \text{with probability } q, \end{cases} \tag{3}$$

where $0 < q < 1$.

In this scenario, any method that waits for a certain number of batches on each iteration to perform a step runs the risk of never receiving the required batch and getting stuck. This includes methods like Rennala SGD or ASGD. Specifically, if the algorithm waits for a single stochastic gradient on each iteration, with probability q^n, it will never receive it and consequently never proceed.

To address these limitations, we propose a new method that, unlike Rennala SGD or ASGD, does not wait for a fixed number of gradients. Instead, it allocates a specific time for computing each stochastic gradient. If a client fails to complete its computation within the designated time, the partial computation is discarded, and a new computation is initiated. Our main contributions are as follows.

- In Section 5, we introduce a time-efficient parallel SGD method, MindFlayer SGD (Algorithm 1), designed for the heterogeneous and random worker compute time regime described in (2). To the best of our knowledge, MindFlayer SGD is the first algorithm tailored to this regime. We demonstrate that our method generalizes Rennala SGD, making it optimal in the deterministic compute time setup. Furthermore, when the distribution of computation times is positively skewed, we show that MindFlayer SGD outperforms the other methods, with the performance gap widening as the skewness coefficient increases. As illustrated in Figure 1, where $\mathcal{J}_i = \text{lognormal}(0, s)$, increasing s results in a higher skewness coefficient, which worsens the performance of Rennala SGD and ASGD. In contrast, MindFlayer SGD remains robust to changes in variance.

  Additional experiments using various functions and distributions are presented in Appendix A. For distributions, we explore lognormal, log-Cauchy, and Infinite-Bernoulli (defined in (3)). For functions, we examine a quadratic loss and a neural network trained on the MNIST [LeCun et al., 1998] dataset. This diverse testing framework highlights MindFlayer SGD's robustness and effectiveness across a wide range of challenging scenarios.

[1]This algorithm is presented by Maranjyan et al. [2025b] and was published after our work. It is included here due to its relevance, but despite its novelty, our method still outperforms Ringmaster ASGD.

[2]We can view η as an extended real random variable, or just assume that ∞ is a very big number.

- In Section 6, we introduce Adaptive-MindFlayer SGD, a version of our algorithm that enhances practicality in two ways: it adapts to the computation time distribution during the learning process, eliminating the need for prior knowledge of the distribution, and it reduces the number of hyperparameters by treating all workers as a single entity, using only two hyperparameters in total. These improvements make Adaptive-MindFlayer SGD more suitable for real-world implementation.
- In Section 7, we compare our theoretical time complexity with that of Rennala SGD, which is optimal in the deterministic time setting. We show that the time complexity of Rennala SGD worsens as the distribution's tails become heavier or as the skewness coefficient increases, leading to an arbitrary performance gap compared to MindFlayer SGD.
- In Appendix C, we expand our theory to develop Vecna SGD, designed for the heterogeneous case, where workers have datasets that are coming from different distributions.

3 RELATED WORK

There are several other related works. Dutta et al. [2018] explore the error-runtime trade-offs in distributed SGD, revealing how slower and stale gradients can sometimes enhance convergence processes. Woodworth et al. [2020] compare local SGD with minibatch SGD, analyzing the efficiency of local updates in different distributed settings. Wu et al. [2022] advance the understanding of asynchronous methods by proposing delay-adaptive step-sizes that adjust to asynchronous learning environments, optimizing the convergence rates. Furthermore, Hanna et al. [2022, 2020] focus on adaptive stochastic gradient descent to improve communication efficiency in distributed learning, offering strategies that reduce communication demands while maintaining fast convergence.

4 MOTIVATION AND SINGLE DEVICE CASE

To illustrate the motivation behind the design of our method, let us consider a single device setup. Recall the scenario introduced in Equation (3) where we have single device and it either returns a gradient after τ time or gets stuck with probability q. A straightforward and optimal workaround to this issue is to wait exactly τ seconds. If we do not receive a gradient within this time frame, it indicates that we will never receive it, so there is no point in waiting longer. In this case, we discard the current computation, which would never finish anyway, and request the device to compute the gradient again. The probability of getting stuck again is lower, so eventually, we will receive a gradient and proceed. Consider the following two strategies for each step.

- **Strategy 1:** Rennala SGD. We wait for the first B stochastic gradients. Thus, the time for one step for this strategy is the random variable:

$$T_B = \sum_{j=1}^{B} (\tau + \eta^j).$$

- **Strategy 2:** MindFlayer SGD. We repeat the following random trial B times: allocate time $\tau + t$ to computing a stochastic gradient. If no gradient is received within that time, discard the current computation and start over. Then the time for the j-th trial is given by:

$$T^j(t) = \begin{cases} \tau + \eta^j, & \text{if } \eta^j \leq t, \\ \tau + t, & \text{if } \eta^j > t. \end{cases}$$

Thus, the time per step is a random variable:

$$\tilde{T}_B(t) = \sum_{j=1}^{B} T^j(t).$$

In the second case, rather than waiting for B gradients, we attempt to compute B gradients. Essentially, we limit the time spent on computing a stochastic gradient. In expectation, Strategy 2 will collect Bp gradients per iteration, where $p = P(\eta \leq t)$ is the probability of collecting a gradient within a trial. Setting $t = \infty$ removes this restriction, resulting in the same strategy as the first one.

The fact that MindFlayer SGD receives Bp gradients on average makes it effectively a scaled-down version of Rennala SGD. Consequently, MindFlayer SGD is expected to require $1/p$ times more iterations than Rennala SGD to achieve the same level of convergence. However, this trade-off is intentional and necessary to handle the heavy-tailed nature of computation times.

We have the following proposition.

Proposition 4.1 (Proof in Appendix F)**.** *Consider the single device setup. Let K be the number of iterations required by* Rennala SGD *with batch size B to find an ε-stationary point. For sufficiently small ε,* MindFlayer SGD *with the same batch size B, needs K/p iterations to find an ε-stationary point.*

Thus, the time complexities in this setting are given by:

$$T_{\text{RennalaSGD}} = K\mathbb{E}\left[T_B\right] = KB(\tau + \mathbb{E}\left[\eta\right]), \tag{4}$$

$$\begin{aligned} T_{\text{MindFlayerSGD}}(t) &= \frac{K}{p}\mathbb{E}\left[\tilde{T}_B(t)\right] \\ &= \frac{K}{p}B(\tau + (1-p)t + p\mathbb{E}\left[\tau | \tau \leq t\right]) \\ &\leq \frac{K}{p}B(\tau + t). \end{aligned} \tag{5}$$

This leads us to the following remark.

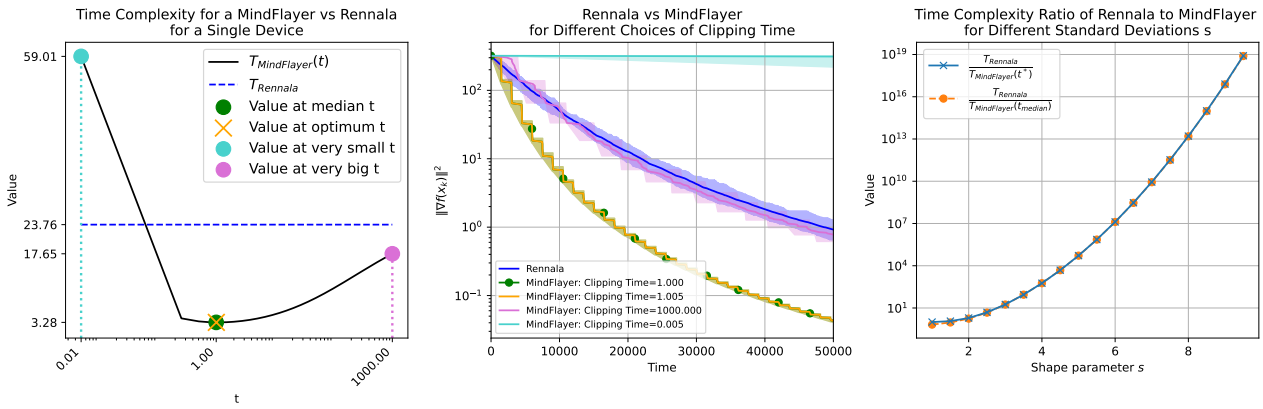

Figure 2: We consider a single-device setting with computation time $\tau + \eta$, where $\tau = 1$ and $\eta \sim \text{lognormal}(0, s)$ for $s = 2.5$. **On the left**, we compare the expected time complexity of MindFlayer SGD (5) as a function of the time threshold t with the constant time complexity of Rennala SGD (4), showing the efficiency of MindFlayer SGD for various choices of t. We set $K = 1$ since it appears in both algorithms and does not affect the comparison. Four key values of t are highlighted: a very small one, the optimal choice from (6), the median of η, and a very large number. As expected, MindFlayer SGD degrades for small t due to the server receiving fewer gradients, remains unchanged between the median and optimal values, and worsens as t increases, approaching Rennala SGD. **In the middle**, we present an empirical evaluation on a quadratic optimization problem (see Appendix A), confirming time complexity reductions for MindFlayer SGD across the same four thresholds. **On the right**, we plot the ratio of time complexities between Rennala SGD and MindFlayer SGD across different standard deviations s, revealing exponential efficiency gains at optimal clipping times, with similar trends at median values.

Remark 4.2. For the case where $n = 1$, MindFlayer SGD is faster than Rennala SGD if there exists a time threshold $t > 0$ such that the following inequality holds:

$$P(\eta \leq t) \geq \frac{\tau + t}{\tau + \mathbb{E}[\eta]}.$$

The optimal choice of t is given by

$$t = \arg\min_{t>0} \frac{\tau + t}{P(\eta \leq t)}. \tag{6}$$

Note that it is necessary that $t \leq \mathbb{E}[\eta]$. In case where $\mathbb{E}[\eta] = \infty$, this assumption hold for any finite values of $t > 0$. An example of this scenario is given in (3). Many other distributions also have an infinite expectation, including Pareto, log-Cauchy, Lévy, log-t, and Landau distributions, among others.

A less restrictive example of distributions are positively skewed distributions. Let

$$s = \mathbb{E}[\eta] - \text{Med}[\eta]$$

be the skewness coefficient of the distribution $\mathcal{J}$, where $\text{Med}[\eta]$ denotes the median of η. If $s > 0$ we say that the distribution is positively skewed. Then we have the following proposition.

Proposition 4.3 (Proof in Appendix F). *For the $n = 1$ case, if $s > \tau + \text{Med}[\eta]$ then* MindFlayer SGD *is faster than* Rennala SGD. *Moreover, if $s = (\tau + \text{Med}[\eta])(2\alpha - 1)$ then*

$$\frac{T_{\text{RennalaSGD}}}{T_{\text{MindFlayerSGD}}(\text{Med}[\eta])} \geq \alpha.$$

Therefore, Rennala SGD can be arbitrarily bad. As an example consider the lognormal(μ, σ^2) distribution. For this distribution, we have:

$$s = \mathbb{E}[\eta] - \text{Med}[\eta] = \exp\left(\mu + \frac{\sigma^2}{2}\right) - \exp(\mu).$$

Thus, as we increase σ, the difference becomes arbitrarily large. To verify this, we also conducted a small experiment, see Figure 2. The right plot showcases how the ratio of time complexity between Rennala SGD and MindFlayer SGD can get arbitrarily large for the optimal clipping time

$$t^* := \arg\min_t T_{\text{MindFlayerSGD}}(t)$$

and even the median of the distribution $t_{\text{median}} = \text{Med}[\eta]$. The left and middle plots showcase the potential improvement, and even loss resulting from different choices of clipping time t.

5 MINDFLAYER SGD

Here, we introduce MindFlayer SGD for the multi-device setting ($n > 1$) Algorithm 1. For the heterogeneous case, see Appendix C.

MindFlayer SGD starts at an initial point $x^0 \in \mathbb{R}^d$ with stepsize $\gamma > 0$, time allowances $t_i > 0$, and trial counts $B_i \geq 0$ for each client. At each iteration k, ranging from 1 to K, the server sends the current point x^k to all clients.

Algorithm 1 MindFlayer SGD [3]

1: **Input:** Initial point $x^0 \in \mathbb{R}^d$, stepsize $\gamma > 0$, allotted times $t_1, \ldots, t_n \geq 0$, number of trials per client $B_1, \ldots, B_n \geq 0$, probabilities $p_i = P(\eta_i \leq t_i)$
2: **for** $k = 1, 2, \ldots, K$ **do**
3: Send x^k to all clients $i \in [n]$, each executes B_i trials
4: Compute the gradient estimate:
$g^k = \sum_{i=1}^n \sum_{j=1}^{B_i} I(\eta_i^j < t_i) \nabla f(x^k; \xi_i^j)$
5: Update: $x^{k+1} = x^k - \frac{\gamma}{B} g^k$, where $B = \sum_{i=1}^n p_i B_i$
6: **end for**

Each client i makes B_i attempts to compute stochastic gradients. During each attempt, it computes a stochastic gradient, but if the computation exceeds the allotted time $\tau_i + t_i$, the gradient is discarded, and a new attempt begins. This behavior is captured by the indicator function $I(\eta_i^j < t_i)$ in line 4 of Algorithm 1.

The probability of completing the computation within the time limit is defined as $p_i := P(\eta_i^j < t_i)$. Thus, the number of stochastic gradients received from client i is a random variable ranging from 0 to B_i, with an expected value of $p_i B_i$. Summing over all clients, the expected total number of stochastic gradients is $B = \sum_{i=1}^n p_i B_i$. Finally, after aggregating the collected gradients, the server updates the point using an unbiased gradient estimator, following the update rule $x^{k+1} = x^k - \frac{\gamma}{B} g^k$.

In the special case where the computation time is deterministic, i.e., $\eta_i = 0$ for every worker $i \in [n]$, we have $p_i = 1$ for all i. While Rennala SGD does not explicitly specify the number of gradient computations B_i for each client, in the deterministic setting, each client will send a fixed number of gradients per communication round. Consequently, for any $t > 0$, MindFlayer SGD Algorithm 1, by choosing B_i appropriately, reduces to Rennala SGD Algorithm 5.

However, the situation changes when $\eta_i > 0$ is not deterministic. If we set $t_i = \infty$ for all $i \in [n]$, MindFlayer SGD Algorithm 1 does not reduce to Rennala SGD Algorithm 5. This is because, in the case of Rennala SGD, the randomness in each iteration causes the number of stochastic gradients computed by each client to vary across different communication rounds. Nevertheless, this scenario is not our primary focus, as we will demonstrate that allowing each worker to complete its gradient computation by setting $t_i = \infty$ is inefficient when dealing with positively skewed distributions.

Since Algorithm 1 has multiple hyperparameters, we provide a more practical version in Section 6, where a single batch size B and a single probability p are used instead of client-specific values. While this simplification is useful in practice, the more fine-grained version with per-client hyperparameters is generally more effective when computation time distributions are known. In such cases, choosing the hyperparameters optimally is straightforward, making this version preferable when efficiency is a priority. For this reason, we begin with a theoretical analysis of the more flexible version before introducing its practical counterpart.

[3] We name our method MindFlayer SGD, drawing inspiration from The Mind Flayer from *Stranger Things*, due to its ability to precisely control its clients, analogous to the creature's supreme control over its victims (The Flayed).

5.1 THEORETICAL ANALYSIS

We consider standard assumptions used in nonconvex optimization.

Assumption 5.1. Function f is differentiable, and its gradient is L–Lipschitz continuous, i.e.,

$$\|\nabla f(x) - \nabla f(y)\| \leq L \|x - y\|, \text{ for all } x, y \in \mathbb{R}^d.$$

Assumption 5.2. The function $f(x)$ is bounded below, and we denote its infimum by $f^{\inf} \in \mathbb{R}$. Let x^0 be the initial point of the optimization method, define $\Delta := f(x^0) - f^{\inf}$.

Assumption 5.3. For all $x \in \mathbb{R}^d$, stochastic gradients $\nabla f(x; \xi)$ are unbiased and σ^2-variance-bounded, i.e.,

$$\mathbb{E}_\xi \left[\nabla f(x; \xi)\right] = \nabla f(x),$$
$$\mathbb{E}_\xi \left[\|\nabla f(x; \xi) - \nabla f(x)\|^2\right] \leq \sigma^2,$$

where $\sigma^2 \geq 0$.

5.1.1 Convergence Theory

The following theorem gives iterations guarantees for the convergence of MindFlayer SGD.

Even though MindFlayer SGD is similar to Rennala SGD the convergence analysis require additional considerations, since the batch size is a random variable here as apposed to the case of Rennala SGD.

Theorem 5.4 (Proof in Appendix G.1)**.** *Assume that Assumptions 5.1, 5.2 and 5.3 hold. Let*

$$B = \sum_{i=1}^n p_i B_i \quad \textit{and} \quad \gamma = \frac{1}{2L} \min\left\{1, \frac{\varepsilon B}{\sigma^2}\right\}$$

in Algorithm 1. Then, the method guarantees that $\frac{1}{K} \sum_{k=0}^{K-1} \mathbb{E}\left[\left\|\nabla f(x^k)\right\|^2\right] \leq \varepsilon$ *after*

$$K \geq \max\left\{1, \frac{\sigma^2}{\varepsilon B}\right\} \frac{8L\left(f(x^0) - f^{\inf}\right)}{\varepsilon}$$

iterations.

Note that the rate is inversely proportional to the probabilities p_i, which is expected—smaller p_i implies a potentially smaller batch size, leading to more iterations.

In the deterministic case where $\eta_i = 0$ for all $i \in [n]$, we have $p_i = P(\eta_i \leq t_i) = 1$ for all $i \in [n]$, resulting in the same rate with $B = \sum_{i=1}^n B_i$. This recovers the rate of Rennala SGD, up to a constant factor. Similarly, as $t_i \to \infty$ for all i, we get $p_i \to 1$, again leading to the same rate.

On the other hand, if $t_i = 0$ for all $i \in [n]$, then $K = \infty$, which is expected—if the success probability is zero for all clients, the server never receives any stochastic gradients, making progress impossible.

5.1.2 Time Complexity

The following theorem gives time complexity for MindFlayer SGD.

Theorem 5.5 (Proof in Appendix G.2)**.** *Assume that Assumptions 5.1, 5.2 and 5.3 hold. Let*

$$B = \sum_{i=1}^n p_i B_i \quad \text{and} \quad \gamma = \frac{1}{2L} \min\left\{1, \frac{\varepsilon B}{\sigma^2}\right\}$$

in Algorithm 1. Let $t = (t_1, \ldots, t_n)$, $t_1, \ldots, t_n \geq 0$. Without loss of generality assume that

$$0 < \tau_1 + t_1 \leq \cdots \leq \tau_n + t_n.$$

Let

$$t(m) = \left(\sum_{j=1}^m \frac{p_j}{\tau_j + t_j}\right)^{-1} \left(S + \sum_{j=1}^m p_j\right),$$

where $S = \max\left\{1, \frac{\sigma^2}{\varepsilon}\right\}$. Let $m^* = \arg\min_{m \in [n]} t(m)$, if there are several minimizers we take the smallest one. Put

$$B_i = \lceil b_i \rceil, \quad b_i = \begin{cases} \frac{t(m^*)}{\tau_i + t_i} - 1, & \text{if } i \leq m^*, \\ 0, & \text{if } i > m^*. \end{cases}$$

Then, MindFlayer SGD guarantees to find an ϵ-stationary point within

$$T_{\text{MindFlayerSGD}}(t) \leq \frac{8\Delta L}{\varepsilon} \times \min_{m \in [n]} \left\{ \left(\frac{1}{m} \sum_{j=1}^m \frac{p_j}{\tau_j + t_j}\right)^{-1} \left(\frac{S}{m} + \bar{p}\right) \right\}$$

seconds, where $\bar{p} = \frac{1}{m} \sum_{j=1}^m p_j$.

The term $\tau_i + t_i / p_i$ also appears in the single-device case (Remark 4.2), where the optimal choice of t_i minimizes this quantity, similar to (6). As in the single-device case, we observe an inverse dependence on p_i: smaller p_i increases $\tau_i + t_i / p_i$, which may lead to certain devices being excluded from m^*. This is expected, as a small p_i indicates an unreliable device.

For the first m^* selected workers, the optimal allocation of B_i depends on $\tau_i + t_i$: the smaller this value (i.e., the faster the device), the more trials it should receive. Thus, the choice of B_i takes into account both the speed of a device (τ_i) and its reliability (p_i). Some devices may be fast (small τ_i) but unreliable (small p_i), leading to $B_i = 0$ for such devices.

This allocation strategy ensures that devices with high computation times and low reliability are excluded. For example, a device might be computationally fast but suffer from frequent network issues, preventing gradients from reaching the server. In such cases, the server should not rely on this device and should exclude it from gradient computations.

In the deterministic case where $\eta_i = 0$, we have $p_i = 1$ for all $i \in [n]$. The optimal choice in this setting is $t_i = 0$, which recovers the same time complexity as Rennala SGD.

This theorem assumes that the computation time distributions of the devices are known. While this may seem restrictive, it is often reasonable in distributed systems, where models are trained repeatedly. Over time, these repeated executions provide enough data to approximate the distributions with high confidence, making such an assumption practical in many cases.

However, relying on this prior knowledge is not always feasible, especially in settings like federated learning, where devices may be unpredictable or newly introduced into the system. To address this, in the next section, we present a practical version of the algorithm that does not require knowledge of computation time distributions, making it more adaptable to real-world scenarios.

6 PRACTICAL MINDFLAYER SGD

To run our algorithm, we need to specify B_i for each client. One possible approach is to learn the distribution of each client's behavior on the fly and assign an appropriate B_i accordingly. Similar ideas are explored by Maranjyan et al. [2025a]. However, this is not straightforward in our setting, since we perform thresholding and cannot directly observe or learn the distribution of clients' compute times. Instead, in this work, we replace the client-specific parameters B_i with a single global parameter B. To make this possible, we treat all clients in a uniform way. If every client had the same probability p of successfully sending a gradient of equal quality, such a simplification would be justified.

To enforce this uniform probability, we need to set t_i such that each client has a probability p of completing its computation within the given threshold. Since we do not assume prior knowledge of the computation time distribution, we estimate t_i empirically. For each client, we need to solve the following problem: given p, find t such that

$$P(\tau + \eta \leq t) = p.$$

Since $P(\tau + \eta < t)$ is a non-decreasing function of t and $I(\tau + \eta \leq t)$ is an unbiased estimator, we can solve this using Robbins-Monro stochastic approximation method [Robbins and Monro, 1951]. This algorithm iteratively updates the threshold t using the rule:

$$t^{r+1} = t^r - \alpha^r \left(I(\tau + \eta^r \leq t^r) - p \right), \tag{7}$$

where α^r is the stepsize, which we take $\alpha^r = \alpha^0/i$ with $\alpha^0 > 0$.

Note that we do not need to know τ and η^r; we only require $I(\tau + \eta^r \leq t^r)$, which is 1 if the worker finishes the computation within the threshold and 0 otherwise.

Putting all the pieces together we derive our algorithm Algorithm 2.

Algorithm 2 Adaptive-MindFlayer SGD

Input: Initial point $x^0 \in \mathbb{R}^d$, stepsize $\gamma > 0$, probability $p \in (0, 1]$, number of trials $B \geq 0$
Robbins-Monro inputs: initial thresholds t_i^0, initial stepsizes $\alpha_i^0 > 0$ for all $i \in [n]$
for $k = 1, 2, \ldots, K$ **do**
  Each worker $i \in [n]$ computes a gradient at x^k
  Initialize $g^k = 0$ and $b = 0$
  while $b < B$ **do**
    Wait for the fastest client i to finish its trial
    Receive $g = I(\tau_i + \eta_i^{k_b} \leq t_i^{k_b}) \nabla f(x^k; \xi_i^{k_b})$
    Update $t_i^{k_b}$ using (7)
    Update: $g^k = g^k + g$; $b = b + 1$
  end while
  Update $x^{k+1} = x^k - \frac{\gamma}{pB} g^k$
end for

The algorithm is controlled by two key parameters: p and B. The parameter p represents system reliability and is shared across all clients, allowing us to define a single global parameter B. Although the Robbins-Monro process requires per-client parameters, these are not critical for the algorithm's performance. For the initial threshold, one can simply start with a large value, as the algorithm will automatically adjust it over time. Similarly, the step size is not highly sensitive; in our experiments, we used 1, but the algorithm performs well with other choices.

By dynamically adjusting t_i based on real-time observations of worker compute times, Adaptive-MindFlayer SGD continuously aligns the clipping threshold with the desired completion probability p. This eliminates the need for extensive manual tuning of hyperparameters and improves robustness to variability in compute times.

Even without assuming prior knowledge of computation time distributions, this empirical threshold selection performs nearly as well as the previous approach, which relied on such knowledge. In Figure 1, we show that Adaptive-MindFlayer SGD achieves comparable performance to MindFlayer SGD while simplifying hyperparameter selection, making it particularly practical for distributed systems with heterogeneous and unpredictable worker compute times.

7 COMPARING TO RENNALA SGD

Comparing the theoretical performance of Rennala SGD and MindFlayer SGD is particularly challenging due to the inherent randomness in the time complexity of Rennala SGD and the dependence of MindFlayer SGD on optimizing time variables t_i. A comparison based on expected time complexity overlook the nuances of each algorithm's performance across different distributions. Therefore, we turn to an empirical comparison to provide insights into their practical behavior. Particularly, we demonstrate how MindFlayer SGD can achieve arbitrarily small time complexity in heavy-tailed distributions.

To begin, we derive the time complexity of Rennala SGD in the context of random times. Let

$$\mathcal{B} := \left\{ (B_1, B_2, \ldots, B_n) : B_i \in \mathbb{N}_0; \sum_{i=1}^n B_i = B \right\}$$

be the set of all possible batch sizes for each device, the time T_B required for one step with batch size B of Rennala SGD is given by:

$$\begin{aligned} T_B &= \min_{\mathcal{B}} \left\{ \max_{i \in [1,n]} \left\{ B_i \tau_i + \sum_{j=1}^{B_i} \eta_i^j \right\} \right\} \geq T_1 \\ &= \min_{i \in [n]} \left\{ \tau_i + \eta_i^1 \right\} \geq \min_{i \in [n]} \tau_i + \min_{i \in [n]} \eta_i^1. \end{aligned}$$

Thus, the expected time to collect a batch B is

$$\mathbb{E}\left[T_B\right] \geq \tau_{\min} + \mathbb{E}\left[\min_{i \in [n]} \eta_i\right].$$

Note that if the distribution of $\min_{i \in [n]} \eta_i$ is heavy-tailed, then the expected time complexity may be infinite, thus favoring MindFlayer SGD over Rennala SGD. A simple illustration of this occurs when extending the Equation (3) case, where η is either zero or infinite, to scenarios involving multiple devices. In such cases, the expectation of the minimum time across devices, $\min_{i \in [n]} \eta_i$, also results in an infinite expected time complexity.

While a detailed theoretical comparison is intractable, we conduct an empirical comparison to highlight practical differences between the two algorithms. To capture the randomness in Rennala SGD's rate, we generate a histogram of T_B and convolve it with itself K times, where K is the number of iterations needed for ϵ-convergence. We set $n = 100$ and $B = 250$. Additional details are provided in Appendix A.1.

For MindFlayer SGD, we compare the theoretical time complexity from Theorem 5.5. We evaluate two strategies for

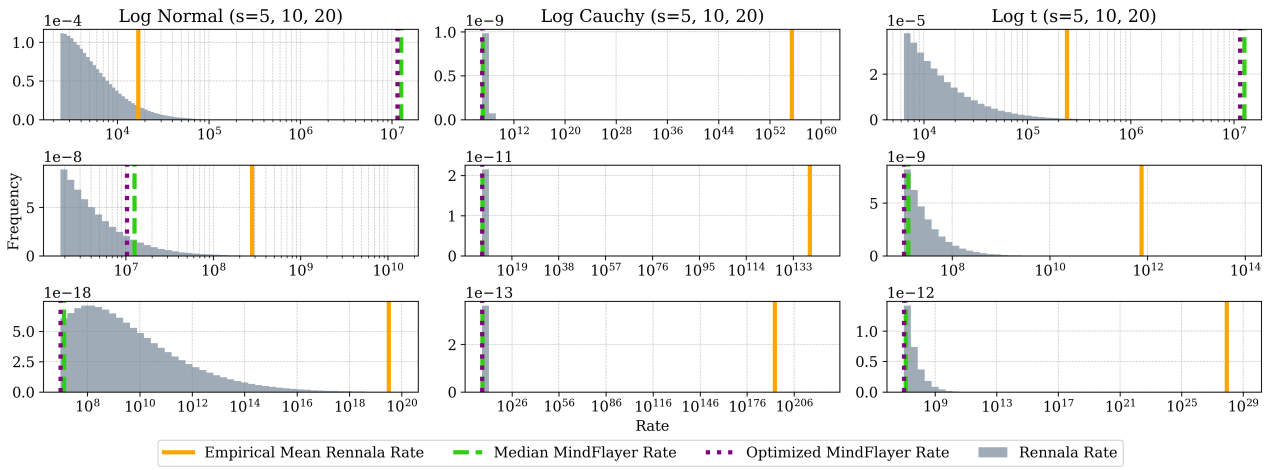

Figure 3: Empirical comparison of the theoretical time complexities of Rennala SGD and MindFlayer SGD. We set $n = 100$ and $B = 250$ (see Appendix A.1 for details). Since Rennala SGD's time complexity is a random variable, we plot a histogram with its empirical mean. For MindFlayer SGD, we compare the theoretical time complexity from Theorem 5.5, evaluating two strategies for selecting t_i: (1) using the median of the distributions $\mathcal{J}_i$, and (2) finding the optimal t_i from (6). We examine three distributions: lognormal, log-Cauchy, and log-t (5 degrees of freedom), shown in each column. As variance increases (rows: 5, 10, 20), MindFlayer SGD increasingly outperforms Rennala SGD.

selecting t_i: (1) using the median of the distributions $\mathcal{J}_i$, and (2) solving the following optimization problem:

$$\text{Fix } m \in [n], \text{ minimize } t(m) \text{ over } t = (t_1, \cdots, t_n), \text{ (remember } p_j = F_j(t_j)).$$

We optimize this using the `L-BFGS-B` algorithm, a well-suited method for solving smooth, convex, or mildly non-convex problems due to its efficiency and robustness [Zhu et al., 1997]. For each m, we take the minimum over all possible configurations.

Our empirical results, illustrated in Figure 3, demonstrate that as the variance of the underlying distribution increases, MindFlayer SGD consistently outperforms Rennala SGD. The heavy-tailed nature of the distributions causes Rennala SGD to experience extreme slowdowns, while MindFlayer SGD maintains robust performance.

8 CONCLUSION AND FUTURE WORK

In this paper, we address the problem of minimizing the expectation of nonconvex functions with Lipschitz gradients, with the use of parallel workers computing stochastic gradients. Our focus lies on the challenging scenario where worker compute times are heterogeneous and random, expanding on recent developments in ASGD methods like Rennala SGD. We observe that while Rennala SGD performs optimally in environments with deterministic compute times, its effectiveness diminishes under random compute conditions.

To better understand and improve stochastic optimization in these conditions, we introduce a novel asynchronous SGD method named MindFlayer SGD. This method adjusts to the randomness in computation times by not adhering to a fixed batch size but rather setting specific times for computing single stochastic gradients. If a client fails to deliver within this time frame, the computation is discarded, and the process restarts. This flexibility allows MindFlayer SGD to perform robustly across various conditions, notably outperforming both Rennala SGD and standard Asynchronous SGD (ASGD) in our theoretical and empirical analysis.

Our results demonstrate that MindFlayer SGD significantly reduces time complexity, particularly in environments characterized by positively skewed distribution of computation times. We empirically validate this in simulations with several distributions conditions where MindFlayer SGD consistently outperforms the other methods, particularly in high-variance scenarios. This showcases its superiority in adapting to the unpredictable duration of gradient computations typical in real-world applications such as federated learning environments.

In this study, our analysis was confined to computation times, with no consideration given to communication times. Future research will extend our investigation to include communication times. Moreover, we plan to explore the application of gradient estimators with varying variance bounds across different clients. We hypothesize that controlling these variance bounds could yield further benefits in the optimization process.

Acknowledgements

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) Center of Excellence for Generative AI, under award number 5940, iii) SDAIA-KAUST Center of Excellence in Artificial Intelligence and Data Science.

References

Alekh Agarwal and John C Duchi. Distributed delayed stochastic optimization. *Advances in neural information processing systems*, 24, 2011.

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.

Arda Aytekin, Hamid Reza Feyzmahdavian, and Mikael Johansson. Analysis and implementation of an asynchronous optimization algorithm for the parameter server. *arXiv preprint arXiv:1610.05507*, 2016.

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

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.

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.

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.

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.

Saeed Ghadimi and Guanghui Lan. Stochastic first-and zeroth-order methods for nonconvex stochastic programming. *SIAM Journal on Optimization*, 23(4):2341–2368, 2013.

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.

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.

Serge Kas Hanna, Rawad Bitar, Parimal Parag, Venkat Dasari, and Salim El Rouayheb. Adaptive distributed stochastic gradient descent for minimizing delay in the presence of stragglers. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4262–4266, 2020. doi: 10.1109/ICASSP40776.2020.9053961.

Serge Kas Hanna, Rawad Bitar, Parimal Parag, Venkat Dasari, and Salim El Rouayheb. Adaptive stochastic gradient descent for fast and communication-efficient distributed learning, 2022.

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

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.

Ahmed Khaled and Peter Richtárik. Better theory for SGD in the nonconvex world. *arXiv preprint arXiv:2002.03329*, 2020.

Anastasiia Koloskova, Sebastian U Stich, and Martin Jaggi. Sharper convergence guarantees for Asynchronous SGD for distributed and federated learning. *Advances in Neural Information Processing Systems*, 35:17202–17215, 2022.

Jakub Konečný, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Horia Mania, Xinghao Pan, Dimitris Papailiopoulos, Benjamin Recht, Kannan Ramchandran, and Michael I Jordan. Perturbed iterate analysis for asynchronous stochastic optimization. *SIAM Journal on Optimization*, 27(4): 2202–2229, 2017.

Artavazd Maranjyan, 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*, 2025a.

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

Konstantin Mishchenko, Franck Iutzeler, Jérôme Malick, and Massih-Reza Amini. A delay-tolerant proximal-gradient algorithm for distributed learning. In *International Conference on Machine Learning*, pages 3587–3595. PMLR, 2018.

Konstantin Mishchenko, Francis Bach, Mathieu Even, and Blake Woodworth. Asynchronous SGD beats minibatch SGD under arbitrary delays. *arXiv preprint arXiv:2206.07638*, 2022.

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.

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.

Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. Hogwild!: A lock-free approach to parallelizing stochastic gradient descent. *Advances in Neural Information Processing Systems*, 24, 2011.

Herbert Robbins and Sutton Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, 22(3):400–407, 1951. doi: 10.1214/aoms/1177729586.

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.

Blake Woodworth, Kumar Kshitij Patel, Sebastian Stich, Zhen Dai, Brian Bullins, Brendan Mcmahan, Ohad Shamir, and Nathan Srebro. Is local SGD better than minibatch SGD? In *International Conference on Machine Learning*, pages 10334–10343. PMLR, 2020.

Xuyang Wu, Sindri Magnusson, Hamid Reza Feyzmahdavian, and Mikael Johansson. Delay-adaptive step-sizes for asynchronous learning. *arXiv preprint arXiv:2202.08550*, 2022.

Ciyou Zhu, Richard H Byrd, Peihuang Lu, and Jorge Nocedal. Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization. *ACM Transactions on Mathematical Software (TOMS)*, 23(4):550–560, 1997.

MindFlayer: Efficient Parallel SGD in the Presence of Heterogeneous and Random Worker Compute Times (Supplementary Material)

Artavazd Maranjyan[1] **Omar Shaikh Omar**[1] **Peter Richtárik**[1]

[1]King Abdullah University of Science and Technology, Thuwal, Saudi Arabia

CONTENTS

1 Introduction **1**

1.1 Minibatch SGD 1

1.2 Asynchronous SGD 2

1.3 Rennala SGD 2

2 Problem Setup and Contributions **2**

3 Related Work **4**

4 Motivation and Single Device Case **4**

5 MindFlayer SGD **5**

5.1 Theoretical Analysis 6

5.1.1 Convergence Theory 6

5.1.2 Time Complexity 7

6 Practical MindFlayer SGD **7**

7 Comparing to Rennala SGD **8**

8 Conclusion and Future Work **9**

A Experimental details **14**

A.1 Comparing Theoretical Time Complexities 14

B Table of Notations **16**

C Heterogeneous Regime **16**

C.1 Related work and discussion . . . 16
C.2 Vecna SGD . . . 17
C.3 Convergence theory . . . 17
C.4 Time Complexity . . . 18

D The Rennala Algorithm **18**

E The classical SGD theory **19**

F Proofs for Propositions in Section 4 **19**

G Proofs for Homogeneous Regime **20**
G.1 Proof of Theorem 5.4 . . . 20
G.1.1 Proof of Lemma G.1 . . . 21
G.1.2 Proof of Lemma G.2 . . . 22
G.2 Proof of Theorem 5.5 . . . 23
G.2.1 Proof of Lemma G.3 . . . 25

H Proofs for Heterogeneous Regime **25**
H.1 Proof of Theorem C.1 . . . 25
H.1.1 Proof of Lemma H.1 . . . 26
H.1.2 Proof of Lemma H.2 . . . 27
H.2 Proof of Theorem C.2 . . . 28
H.2.1 Proof of Lemma H.3 . . . 29

A EXPERIMENTAL DETAILS

In this section we explain the setup for comparing MindFlayer SGD, Rennala SGD, and ASGD, which we used throughout this paper. We compare the algorithms' performance on a quadratic optimization (8) task with access to a stochastic gradient. The parallelism was simulated on a machine with 2 Intel(R) Xeon(R) Gold 6226R CPUs @ 2.90GHz, with a total of 64 logical CPUs. For each setting of the algorithm, we run 10 different seeds for the random time and plot the average, minimum and maximum, see Figure 1, Figure 2, etc.

We use a similar setup to the one employed by Tyurin and Richtárik [2024], but modify it so that we have a known expected variance. We make this choice, so we can compare theoretical parameters, as we did in Figure 2.

Furthermore, we consider the homogeneous optimization problem 1, with the convex quadratic function:

$$f(x) = \frac{1}{2}x^\top Ax - b^\top x \qquad \forall x \in \mathbb{R}^d \,. \tag{8}$$

We take $d = 1000$,

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

Assume that all n workers has access to the following unbiased stochastic gradients:

$$\left[\nabla f(x,\xi)\right]_j := \nabla_j f(x) + \xi \,,$$

where $\xi \sim \mathcal{N}(0, 0.0003^2)$, thus, we get that in Assumption 5.3 we have,

$$\sigma^2 = 0.0003^2 \cdot d = 0.0003^2 \cdot 1000 \,.$$

Now setting the convergence threshold $\epsilon = 10^{-4}$, we can infer all theoretical parameters. To find the optimal time corresponding to Rennala SGD we need to fix the times, we do that by either removing the randomness, or adding the expected randomness. On the other hand, for MindFlayer SGD we use the results from Theorem 5.5 to set the theoretical number of trials for each client. For some experiments we used theoretical stepsizes, e.g. Figure 2, for others we used the range of stepsizes from a set $\{2^i | i \in [-10, 10]\}$, e.g. Figures 1, 1, and 4, similarly to Tyurin and Richtárik [2024]. Finally, for the nonconvex problem in Figure 5 we tried the set $\{0.01, 0.001, 0.0001\}$.

In addition to the experimental results shown throughout the paper, we ran two more experiments. One with the Infinite-Bernoulli distribution on the same quadratic problem, and a second with the log-Cauchy distribution with a small two-layer neural network on the MNIST dataset, see Figure 4 and Figure 5.

A.1 COMPARING THEORETICAL TIME COMPLEXITIES

To produce the empirical comparison demonstrated in Figure 3, we design a synthetic setup to compare the time complexities of MindFlayer SGD and Rennala SGD. We use distributions with varying tail behaviors and levels of variance. Specifically, the base task time for each worker is defined as $\tau_i = 100\sqrt{i}$, which accentuates systematic variation in task durations across workers. On top of this base time, random delays are added, drawn from one of the distributions listed in Table 1. These distributions are parameterized by varying scale factors $s = 5, 10, 20$. The log-t distribution uses df $= 5$ to control its tail behavior.

The rate of MindFlayer SGD is derived from Theorem 5.5, expressed as:

$$T_{\text{MindFlayerSGD}}(t) \leq 8 \times \min_{m\in[n]} \left\{ \left(\frac{1}{m}\sum_{j=1}^{m} \frac{p_j}{\tau_j + t_j} \right)^{-1} \left(\frac{S}{m} + \frac{1}{m}\sum_{j=1}^{m} p_j \right) \frac{\Delta L}{\varepsilon} \right\}$$

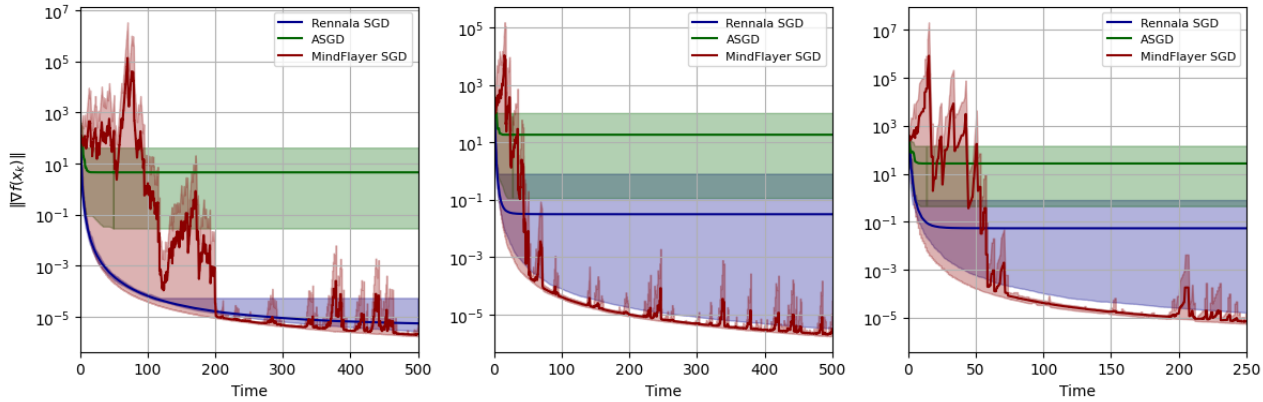

Figure 4: We ran an experiment as described in Appendix A where we employ the same $\mathcal{J}_i = \text{InfBernoulli}(q)$ distribution for all clients $i \in [n]$, with different q values. From left to right we have $q = 0.6, 0.7, 0.8$. Additionally, we set $\tau_i = \sqrt{i}$. As we observe, with an increase of the probability of failure q unlike Rennala SGD and ASGD, MindFlayer SGD demonstrates the ability to continue optimizing and not be stuck

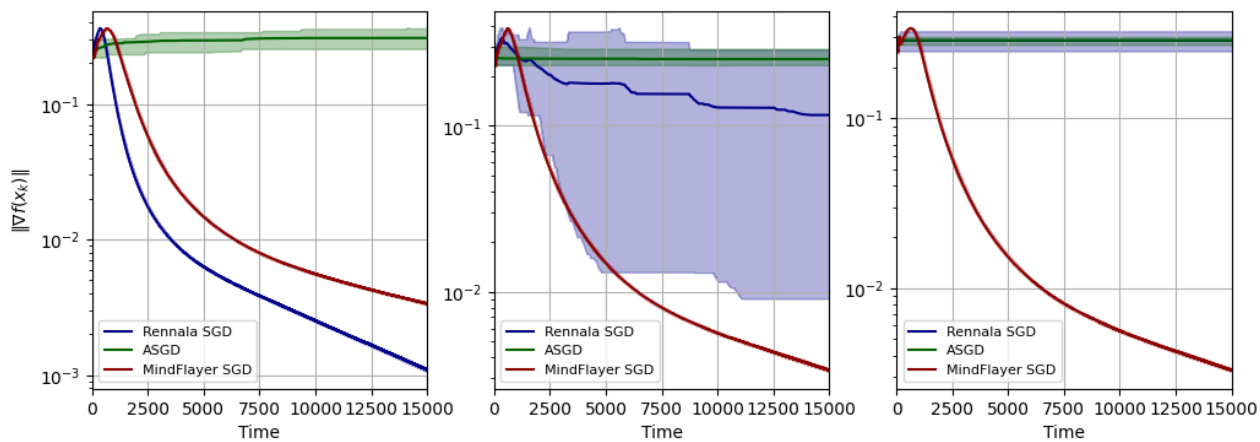

Figure 5: We train a two layer Neural Network on the MNIST dataset where we set the distribution $\mathcal{J}_i = \text{log-Cauchy}(s)$ for all clients $i \in [n]$, with different scale values s. From left to right we have $s = 1, 10, 100$. Additionally, we set $\tau_i = \sqrt{i}$. We observe that MindFlayer SGD convergence doesn't suffer from the increase in the scale parameter s. On the other hand, Rennala and ASGD are delayed significantly with bigger scale parameters s

Here, the rates depend on the clipping times $t = (t_1, \ldots, t_n)$, which determine $p_i := F_i(t_i)$. As described in Section 7, we use both the medians of the distributions and the output of an optimizer as baselines for t.

For Rennala SGD, the time complexity is modeled as an approximation of the random variable $\sum_{i=1}^{K} T_B^i$, where $K = \max\left\{1, \frac{\sigma^2}{\varepsilon}\right\} \frac{8\Delta L}{\varepsilon}$ is the iteration complexity, as demonstrated in the proof of Proposition 4.1 in Section F. The random variables T_B^i represent the time to collect a theoretical batch of size $B = \lceil \sigma^2/\varepsilon \rceil$. As described in Section 7, we obtain an approximation of the distribution of the sum using convolution.

Both rates depend on the problem constants $\varepsilon, \sigma^2, L, \Delta$, which are defined in Table 2.

Distribution	**Parameters**
log-Normal	Scale $s = 5, 10, 20$, Mean $\mu = 0$
log-Cauchy	Location $\mu = 0$, Scale $s = 5, 10, 20$
log-t	Degrees of freedom df $= 5$, Location $\mu = 0$, Scale $s = 5, 10, 20$

Table 1: Random Time Setup for Worker Task Times

Symbol	Description	Value
σ	Standard deviation of noise	$\sqrt{0.025}$
ϵ	Convergence threshold	10^{-4}
$\Delta \cdot L$	Lipschitz constant multiplied by suboptimality	1
B	Rennala SGD batch size	$\lceil \sigma^2/\epsilon \rceil$
S	Theoretical Parameter in MindFlayer SGD	$\lceil \sigma^2/\epsilon \rceil$
n	Number of workers	100
n_{samples}	Number of random samples to estimate T_B	1000
τ_i	Fixed compute time for worker i	$100\sqrt{i}$

Table 2: Experimental Parameters

B TABLE OF NOTATIONS

Notation	Meaning
$[n]$	$\{1, \ldots, n\}$
L	Lipschitz constant of gradients, i.e., $\|\nabla f(x) - \nabla f(y)\| \leq L \|x - y\|$ (Assumption 5.1)
$f^{\inf}$	Minimum value of the function, i.e., $f^{\inf} \leq f(x)$ (Assumption 5.2)
Δ	$\Delta := f(x^0) - f^{\inf}$, where x^0 is the starting point of optimization methods
σ^2	Variance bound on gradients, i.e., $\mathbb{E}_\xi \left[\|\nabla f(x;\xi) - \nabla f(x)\|^2\right] \leq \sigma^2$ (Assumption 5.3)
γ	Stepsize
τ_i	Minimum time required for client i to compute a gradient
η_i	Additional random time taken while computing the gradient
$\mathcal{J}_i$	Distribution of the non-negative random variable η_i
t_i	Allotted time for worker i to compute a gradient

C HETEROGENEOUS REGIME

So far, we have discussed the scenario where all workers compute i.i.d. stochastic gradients. However, in distributed optimization and federated learning [Konečný et al., 2016], workers may have different datasets. Consider the following optimization problem:

$$\min_{x \in \mathbb{R}^d} \left\{ f(x) := \frac{1}{n} \sum_{i=1}^{n} \mathbb{E}_{\xi_i \sim \mathcal{D}_i} \left[f_i(x; \xi_i) \right] \right\}, \tag{9}$$

where $f_i : \mathbb{R}^d \times \mathbb{S}_i \to \mathbb{R}^d$ and ξ_i are random variables with some distributions $\mathcal{D}_i$ on $\mathbb{S}_i$. Problem (9) generalizes problem (1).

C.1 RELATED WORK AND DISCUSSION

The optimization problem (9) has been thoroughly studied in many papers, including [Aytekin et al., 2016, Mishchenko et al., 2018, Nguyen et al., 2022, Wu et al., 2022, Koloskova et al., 2022, Mishchenko et al., 2022]. There have been attempts to analyze Asynchronous SGD in the heterogeneous setting. For example, Mishchenko et al. [2022] demonstrated convergence only to a neighborhood of the solution. In general, achieving good rates for Asynchronous SGD is difficult without making additional assumptions about the similarity of the functions f_i [Koloskova et al., 2022, Mishchenko et al., 2022].

In the deterministic case, when $\sigma^2 = 0$, Wu et al. [2022] analyzed the PIAG method in the deterministic heterogeneous regime and showed convergence. Although the performance of PIAG can be good in practice, in the worst case PIAG requires $\mathrm{O}\left(\tau_n \widehat{L} \Delta / \varepsilon\right)$ seconds to converge, where τ_n is the time delay of the slowest worker, $\widehat{L} := \sqrt{\sum_{i=1}^n L_i^2}$, and L_i is a Lipschitz constant of ∇f_i. Note that the synchronous Minibatch SGD (see Section 1.1) method has the complexity $\mathrm{O}\left(\tau_n L \Delta / \varepsilon\right)$, which

is always better.[1]

Tyurin and Richtárik [2024] proposed an optimal method in the regime where worker computation times are deterministic, similar to the homogeneous setup.

C.2 VECNA SGD

Here we describe our method called Vecna SGD.

Algorithm 3 Vecna SGD [2]

Input: starting point $x^0 \in \mathbb{R}^d$, stepsize $\gamma > 0$, allotted times $t_1, \ldots, t_n \geq 0$, number of trials per client $B_1, \ldots, B_n \geq 0$

for $k = 1, 2, \ldots, K$ **do**
  Put $g_i^k = 0$
  Send x^k to all clients
  Run Method 4 in all clients $i = 1, 2, \ldots, n$
  while there is a client that has trials to perform **do**
    Wait for the fastest client
    Receive gradient g_i from client i
    $g_i^k = g_i^k + g$
  end while
  $g^k = \frac{1}{n} \sum_{i=1}^n \frac{g_i^k}{p_i B_i}$, ◇ $p_i = F_i(t_i) = P(\eta_i \leq t_i)$.
  $x^{k+1} = x^k - \gamma g^k$
end for

Algorithm 4 Client i-s k-th step

Receive x^k from the server
for $j = 1, 2, \ldots, B_i$ **do**
  Sample $\eta_i^j \sim \mathcal{J}_i$ ◇ Start computing gradient estimator.
  if $\eta_i^j \leq t_i$ **then**
    $g = \nabla f(x^k; \xi_i^j), \;\; \xi_i^j \sim \mathcal{D}$ ◇ The computation completes within the allotted time t_i.
    Send g to the server
  end if
end for

The Vecna SGD algorithm begins with an initialization at a starting point x^0 in $\mathbb{R}^d$, with a specified stepsize γ, time allowances t_i, and trial counts B_i for each client. In each iteration k, ranging from $k = 1$ to K, the server distributes the current point x^k to all clients. Each client i then executes a subroutine (Algorithm 4) to attempt to compute B_i stochastic gradients from samples ξ_i^j drawn from a distribution $\mathcal{D}$. During each attempt, client i starts computing a stochastic gradient; if the computation exceeds the allotted time t_i, they discard the current gradient and begin another computation. Consequently, the actual number of stochastic gradients received from each client i becomes a random variable, ranging from 0 to B_i. The expected number of gradients from client i is given by $p_i B_i$. The server normalizes the gradients by the expected batch size $p_i B_i$ and then aggregates them. Finally, the point is updated to $x^{k+1} = x^k - \gamma g^k$ following each aggregation round.

C.3 CONVERGENCE THEORY

The following theorem gives iterations guarantees for the convergence of Vecna SGD.

[1] In the nonconvex case, $\widehat{L}$ can be arbitrarily larger than L.

[2] We name our method Vecna SGD, drawing inspiration from Vecna from *Stranger Things*.

Theorem C.1 (Proof in Appendix H.1)**.** *Assume that Assumptions 5.1, 5.2 and 5.3 hold for function f_i for all $i \in [n]$. Let $\gamma = \min\left\{\frac{1}{\sqrt{L\alpha K}}, \frac{1}{L\beta}, \frac{\varepsilon}{2L\zeta}\right\}$ in Algorithm 3. Then after*

$$K \geq \frac{12\Delta L}{\varepsilon} \max\left\{\beta, \frac{12\Delta\alpha}{\varepsilon}, \frac{2\zeta}{\varepsilon}\right\},$$

iterations, the method guarantees that $\min_{0\leq k\leq K} \mathbb{E}\left[\left\|\nabla f(x^k)\right\|^2\right] \leq \varepsilon$, and

$$\alpha = \frac{L}{n}\max_{i\in[n]}\left\{\frac{1-p_i}{p_iB_i}\right\}, \quad \beta = 1, \quad \zeta = \frac{\sigma^2}{n^2}\sum_{i=1}^n \frac{1}{p_iB_i} + \frac{2L}{n}\max_{i\in[n]}\left\{\frac{1-p_i}{p_iB_i}\right\}\Delta^{\inf},$$

where $\Delta^{\inf} := \sum_{i=1}^n \left(f^{\inf} - f_i^{\inf}\right)$.

C.4 TIME COMPLEXITY

The following theorem gives time complexity for Vecna SGD.

Theorem C.2 (Proof in Appendix H.2)**.** *Assume that Assumptions 5.1, 5.2 and 5.3 hold for function f_i for all $i \in [n]$. Let $\gamma = \min\left\{\frac{1}{\sqrt{L\alpha K}}, \frac{1}{L}, \frac{\varepsilon}{2L}\right\}$ in Algorithm 3, where*

$$\alpha = \frac{L}{n}\max_{i\in[n]}\left\{\frac{1-p_i}{p_iB_i}\right\}, \quad \zeta = \frac{\sigma^2}{n^2}\sum_{i=1}^n \frac{1}{p_iB_i} + \frac{2L}{n}\max_{i\in[n]}\left\{\frac{1-p_i}{p_iB_i}\right\}\Delta^{\inf}.$$

Let $t = (t_1, \ldots, t_n)$, $t_1, \ldots, t_n \geq 0$. Without loss of generality assume that $0 < \tau_1 + t_1 \leq \cdots \leq \tau_n + t_n$. Let

$$T = \tau_n + t_n + \left[\frac{1}{n}\sum_{i=1}^n \frac{\tau_i + t_i}{p_i}\right]\frac{\sigma^2}{n\varepsilon} + \max_{i\in[n]}\left\{\frac{1-p_i}{p_i}(\tau_i + t_i)\right\}\frac{L\left(\Delta + \Delta^{\inf}\right)}{n\varepsilon}.$$

Put

$$B_i = \lceil b_i \rceil, \quad b_i = \frac{T}{\tau_i + t_i}.$$

Then, Vecna SGD guarantees to find an ϵ-stationary point after

$$T_{\text{VecnaSGD}}(t) \leq 288 \times \frac{\Delta L}{\varepsilon}\left(\tau_n + t_n + \left[\frac{1}{n}\sum_{i=1}^n \frac{\tau_i + t_i}{p_i}\right]\frac{\sigma^2}{n\varepsilon} + \max_{i\in[n]}\left\{\frac{1-p_i}{p_i}(\tau_i + t_i)\right\}\frac{L\left(\Delta + \Delta^{\inf}\right)}{n\varepsilon}\right)$$

seconds.

D THE RENNALA ALGORITHM

Algorithm 5 Rennala SGD

Input: starting point x^0, stepsize γ, batch size S
Run Method 6 in all workers
for $k = 0, 1, \ldots, K-1$ **do**
  Init $g^k = 0$ and $s = 1$
  while $s \leq S$ **do**
    Wait for the next worker
    Receive gradient and iteration index (g, k')
    if $k' = k$ **then**
      $g^k = g^k + \frac{1}{S}g; \quad s = s + 1$
    end if
    Send (x^k, k) to the worker
  end while
  $x^{k+1} = x^k - \gamma g^k$
end for

Algorithm 6 Worker's Infinite Loop

Init $g = 0$ and $k' = -1$
while True **do**
  Send (g, k') to the server
  Receive (x^k, k) from the server
  $k' = k$
  $g = \nabla f(x^k; \xi), \quad \xi \sim \mathcal{D}$
end while

We mention the Rennala SGD throughout the paper, here we provide a brief introduction to the method and its development. Algorithm 5 shows the work done by the server. Essentially, the server asynchronously waits to collect a batch of size S, whenever it receives a gradient from a worker that has the same iteration as the algorithm, it assigns it to compute a gradient at the same point x_k. After collecting the batch, we preform a synchronous update (given that all gradients were made on the same point x_k), using an average of the collected batch.

E THE CLASSICAL SGD THEORY

In this section, we present the classical SGD theory as developed by Ghadimi and Lan [2013] and Khaled and Richtárik [2020]. Our analysis will follow the approach of the latter.

We consider the stochastic gradient descent (SGD) method:

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

where $x^0 \in \mathbb{R}^d$ is the initial point, and $g(x)$ is a stochastic gradient estimator at x.

We make the following assumption:

Assumption E.1. The stochastic gradient estimator $g(x)$ satisfies:

$$\begin{aligned}
&\mathbb{E}\left[g(x)\right] = \nabla f(x) \\
&\mathbb{E}\left[\|g(x)\|^2\right] \leq 2\alpha\left(f(x) - f^{\inf}\right) + \beta\|\nabla f(x)\|^2 + \zeta,
\end{aligned}$$

for all $x \in \mathbb{R}^d$ and some constants $\alpha, \beta, \zeta \geq 0$.

This assumption is both general and reasonable, and it is satisfied by many modern SGD-type methods. For further details, refer to Khaled and Richtárik [2020].

Under this assumption, we can derive the following convergence result.

Theorem E.2 (Corollary 1 [Khaled and Richtárik, 2020])**.** *Assume that Assumptions 5.1, 5.2 and E.1 hold. Then for any* $\varepsilon > 0$

$$\min_{0 \leq k \leq K} \mathbb{E}\left[\left\|\nabla f(x^k)\right\|^2\right] \leq \varepsilon$$

for

$$\gamma = \min\left\{\frac{1}{\sqrt{L\alpha K}}, \frac{1}{L\beta}, \frac{\varepsilon}{2L\zeta}\right\},$$

and

$$K \geq \frac{12L\left(f(x_0) - f^{\inf}\right)}{\varepsilon} \max\left\{\beta, \frac{12\Delta\alpha}{\varepsilon}, \frac{2\zeta}{\varepsilon}\right\}.$$

F PROOFS FOR PROPOSITIONS IN SECTION 4

Proposition 4.1. *Consider the single device setup. Let K be the number of iterations required by* Rennala SGD *with batch size B to find an ε-stationary point. For sufficiently small ε,* MindFlayer SGD *with the same batch size B, needs K/p iterations to find an ε-stationary point.*

Proof. The iterations of Rennala SGD can be viewed as iterations of Minibatch SGD. Thus, we can apply the classical SGD theory (Theorem E.2) to derive its iteration complexity:

$$K = \max\left\{1, \frac{\sigma^2}{\varepsilon B}\right\} \frac{8L(f(x^0) - f^{\inf})}{\varepsilon}.$$

For MindFlayer SGD, the iteration complexity follows from Theorem 5.4. Therefore, the number of iterations K_M required for MindFlayer SGD to guarantee that

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

is given by

$$K_M=\max\left\{1,\frac{\sigma^2}{\varepsilon Bp}\right\}\frac{8L(f(x^0)-f^{\inf})}{\varepsilon}.$$

If $\varepsilon\leq\frac{\sigma^2}{B}$, we have

$$K_M=\frac{K}{p}.$$

□

Proposition 4.3. *For the $n=1$ case, if $s>\tau+\mathrm{Med}[\eta]$ then* MindFlayer SGD *is faster than* Rennala SGD*. Moreover, if $s=(\tau+\mathrm{Med}[\eta])(2\alpha-1)$ then*

$$\frac{T_{\text{RennalaSGD}}}{T_{\text{MindFlayerSGD}}(\mathrm{Med}[\eta])}\geq\alpha.$$

Proof. Let $t=\mathrm{Med}[\eta]=:m$, recall that $s=\mathbb{E}[\eta]-m$, then we have:

$$T_{\text{MindFlayerSGD}}(m)\leq\frac{K}{p}B(\tau+t)=2KB(\tau+m),$$

$$T_{\text{RennalaSGD}}=KB(\tau+\mathbb{E}[\eta])=KB(\tau+m+s).$$

Thus, if $s>\tau+m$ then MindFlayer SGD is faster than Rennala SGD.

Now, let $s=(\tau+m)(2\alpha-1)$ then

$$\frac{T_{\text{RennalaSGD}}}{T_{\text{MindFlayerSGD}}(m)}\geq\frac{\tau+m+s}{2(\tau+m)}=\frac{2\alpha(\tau+m)}{2(\tau+m)}=\alpha.$$

□

G PROOFS FOR HOMOGENEOUS REGIME

G.1 PROOF OF THEOREM 5.4

First, we rewrite MindFlayer SGD in a classical SGD way where we do gradient step with an unbiased estimator of the gradient at each iteration.

Algorithm 7 MindFlayer SGD

1: **Input:** starting point x^0, stepsize γ, time budgets $t_1,\ldots,t_n\geq 0$, batch sizes $B_1,\ldots,B_n\geq 0$,
2: **for** $k=0,1,\ldots,K-1$ **do**
3: $\quad g^k=\frac{1}{B}\sum_{i=1}^n\sum_{j=1}^{B_i}I\left(\eta_i^j\leq t_i\right)\nabla f\left(x^k;\xi_i^j\right)$
4: $\quad x^{k+1}=x^k-\gamma g^k$
5: **end for**

where $B=\sum_{i=1}^n p_iB_i$, $p_i=F(t_i)=P(\eta_i\leq t_i)$ and $I(\cdot)$ denotes the indicator function. To prove the theorem we need to establish some properties of the gradient estimator. First, we need an unbiased estimator.

Lemma G.1 (Proof in Appendix G.1.1)**.** *The gradient estimator in Algorithm 7 given by*

$$g(x):=\frac{1}{B}\sum_{i=1}^n\sum_{j=1}^{B_i}I\left(\eta_i^j\leq t_i\right)\nabla f\left(x;\xi_i^j\right)$$

is unbiased, i.e., $\mathbb{E}[g(x)]=\nabla f(x)$ for all $x\in\mathbb{R}^d$.

Next, we obtain an upper bound for the variance of this estimator.

Lemma G.2 (Proof in Appendix G.1.2)**.** *The gradient estimator in Algorithm 7 given by*

$$g(x) := \frac{1}{B}\sum_{i=1}^{n}\sum_{j=1}^{B_i} I\left(\eta_i^j \le t_i\right)\nabla f\left(x;\xi_i^j\right)$$

satisfies

$$\mathbb{E}\left[\left\|g(x)^2\right\|\right] \le 2\left\|\nabla f(x)\right\|^2 + \frac{1}{B}\sigma^2.$$

We are ready to prove the Theorem 5.4.

Theorem 5.4. *Assume that Assumptions 5.1, 5.2 and 5.3 hold. Let*

$$B = \sum_{i=1}^{n} p_i B_i \quad \text{and} \quad \gamma = \frac{1}{2L}\min\left\{1, \frac{\varepsilon B}{\sigma^2}\right\}$$

in Algorithm 1. Then, after

$$K \ge \max\left\{1, \frac{\sigma^2}{\varepsilon B}\right\}\frac{8L\left(f(x^0) - f^{\inf}\right)}{\varepsilon}$$

iterations, the method guarantees that $\frac{1}{K}\sum_{k=0}^{K-1}\mathbb{E}\left[\left\|\nabla f(x^k)\right\|^2\right] \le \varepsilon$.

Proof. Note that Algorithm 1 can be viewed as a special case of classical stochastic gradient descent (SGD), as reformulated in Algorithm 7. We need to verify that the gradient estimator fulfills the conditions required by classical SGD (Theorem E.2). The two preceding lemmas address this requirement precisely. Specifically, Lemma G.1 confirms that the gradient estimator used in Algorithm 7 is unbiased, while Lemma G.2 verifies that the variance of this estimator meets the conditions specified in Assumption E.1, with $\alpha = 0$, $\beta = 2$ and $\zeta = \frac{\sigma^2}{B}$. Consequently, it remains to apply Theorem E.2. □

G.1.1 Proof of Lemma G.1

Lemma G.1. *The gradient estimator in Algorithm 7 given by*

$$g(x) := \frac{1}{B}\sum_{i=1}^{n}\sum_{j=1}^{B_i} I\left(\eta_i^j \le t_i\right)\nabla f\left(x;\xi_i^j\right)$$

is unbiased, i.e., $\mathbb{E}\left[g(x)\right] = \nabla f(x)$ *for all* $x \in \mathbb{R}^d$*, where* $B = \sum_{i=1}^{n} p_i B_i$.

Proof. This follows from direct computation:

$$\begin{aligned}
\mathbb{E}\left[g(x)\right] &= \mathbb{E}\left[\frac{1}{B}\sum_{i=1}^{n}\sum_{j=1}^{B_i} I\left(\eta_i^j \le t_i\right)\nabla f\left(x;\xi_i^j\right)\right] \\
&= \frac{1}{B}\sum_{i=1}^{n}\sum_{j=1}^{B_i}\mathbb{E}\left[I\left(\eta_i^j \le t_i\right)\nabla f\left(x;\xi_i^j\right)\right] \\
&\overset{(\eta_i^j \perp \xi_i^j)}{=} \frac{1}{B}\sum_{i=1}^{n}\sum_{j=1}^{B_i}\mathbb{E}\left[I\left(\eta_i^j \le t_i\right)\right]\mathbb{E}\left[\nabla f\left(x;\xi_i^j\right)\right] \\
&= \frac{1}{B}\sum_{i=1}^{n}\sum_{j=1}^{B_i} p_i \nabla f(x) \\
&= \nabla f(x)\frac{1}{B}\sum_{i=1}^{n} p_i B_i \\
&= \nabla f(x).
\end{aligned}$$

□

G.1.2 Proof of Lemma G.2

Lemma G.2. *The gradient estimator in Algorithm 7 given by*

$$g(x) := \frac{1}{B}\sum_{i=1}^{n}\sum_{j=1}^{B_i} I\left(\eta_i^j \le t_i\right)\nabla f\left(x;\xi_i^j\right)$$

satisfies

$$\mathbb{E}\left[\left\|g(x)^2\right\|\right] \le 2\left\|\nabla f(x)\right\|^2 + \frac{1}{B}\sigma^2,$$

where $B = \sum_{i=1}^{n} p_i B_i$.

Proof. In order to simplify notation, let

$$a_i := \sum_{j=1}^{B_i} b_i^j,$$

where

$$b_i^j := I\left(\eta_i^j \le t_i\right)\nabla f\left(x;\xi_i^j\right).$$

Step 1 (Initial expression). We express $\mathbb{E}\left[\|g(x)\|^2\right]$ in terms of a_i:

$$\mathbb{E}\left[\|g(x)\|^2\right] = \mathbb{E}\left[\left\|\frac{1}{B}\sum_{i=1}^{n} a_i\right\|^2\right] = \frac{1}{B^2}\mathbb{E}\left[\sum_{i=1}^{n}\|a_i\|^2 + \sum_{i\neq j}\langle a_i, a_j\rangle\right].$$

We further simplify both terms via:

$$\|a_i\|^2 = \left\|\sum_{j=1}^{B_i} b_i^j\right\|^2 = \sum_{j=1}^{B_i}\left\|b_i^j\right\|^2 + \sum_{k\neq l}\left\langle b_i^k, b_i^l\right\rangle, \tag{10}$$

$$\langle a_i, a_j\rangle = \left\langle \sum_{k=1}^{B_i} b_i^k, \sum_{l=1}^{B_j} b_j^l\right\rangle = \sum_{k=1}^{B_i}\sum_{l=1}^{B_j}\left\langle b_i^k, b_j^l\right\rangle. \tag{11}$$

Step 2. (Finding the expectations). Further

$$\begin{aligned}
\mathbb{E}\left[\left\|b_i^j\right\|^2\right] &= \mathbb{E}\left[\left(I\left(\eta_i^j \le t_i\right)\right)^2\left\|\nabla f\left(x;\xi_i^j\right)\right\|^2\right] \\
&\overset{(\eta_i^j \perp\!\!\!\perp \xi_i^j)}{=} \mathbb{E}\left[\left(I\left(\eta_i^j \le t_i\right)\right)^2\right]\mathbb{E}\left[\left\|\nabla f\left(x;\xi_i^j\right)\right\|^2\right] \\
&\le p_i\left(\|\nabla f(x)\|^2 + \mathbb{E}\left[\left\|\nabla f\left(x;\xi_i^j\right) - \nabla f(x)\right\|^2\right]\right) \\
&\overset{(Assumption\ 5.3)}{\le} p_i\left(\|\nabla f(x)\|^2 + \sigma^2\right),
\end{aligned} \tag{12}$$

and

$$\begin{aligned}
\mathbb{E}\left[\left\langle b_i^k, b_j^l\right\rangle\right] &= \mathbb{E}\left[\left\langle I\left(\eta_i^k \le t_i\right)\nabla f\left(x;\xi_i^k\right), I\left(\eta_j^l \le t_j\right)\nabla f\left(x;\xi_j^l\right)\right\rangle\right] \\
&\overset{(\perp\!\!\!\perp)}{=} \mathbb{E}\left[I\left(\eta_i^k \le t_i\right)\right]\mathbb{E}\left[I\left(\eta_j^l \le t_j\right)\right]\left\langle\mathbb{E}\left[\nabla f\left(x;\xi_i^k\right)\right], \mathbb{E}\left[\nabla f\left(x;\xi_j^l\right)\right]\right\rangle \\
&= p_i p_j\|\nabla f(x)\|^2.
\end{aligned} \tag{13}$$

Step 3 (Putting everything together). We start with

$$\begin{aligned}\mathbb{E}\left[\|a_i\|^2\right] &\overset{(10,12,13)}{\leq} B_i p_i \left(\|\nabla f(x)\|^2 + \sigma^2\right) + B_i \left(B_i - 1\right) p_i^2 \|\nabla f(x)\|^2 \\ &\leq B_i p_i \left(\|\nabla f(x)\|^2 + \sigma^2\right) + B_i^2 p_i^2 \|\nabla f(x)\|^2,\end{aligned}$$

using this and recalling the definition of B, we get

$$\mathbb{E}\left[\sum_{i=1}^n \|a_i\|^2\right] \leq B \|\nabla f(x)\|^2 + B\sigma^2 + \|\nabla f(x)\|^2 \sum_{i=1}^n B_i^2 p_i^2.$$

Next

$$\langle a_i, a_j \rangle \overset{(11,13)}{=} B_i p_i B_j p_j \|\nabla f(x)\|^2,$$

finally,

$$\begin{aligned}\mathbb{E}\left[\|g(x)\|^2\right] &= \frac{1}{B^2}\mathbb{E}\left[\sum_{i=1}^n \|a_i\|^2 + \sum_{i\neq j} \langle a_i, a_j \rangle\right] \\ &\leq \frac{1}{B^2}\left[B \|\nabla f(x)\|^2 + B\sigma^2 + \left(\sum_{i=1}^n B_i^2 p_i^2 + \sum_{i\neq j} B_i p_i B_j p_j\right) \|\nabla f(x)\|^2\right] \\ &= \frac{1}{B^2}\left(B + B^2\right) \|\nabla f(x)\|^2 + \frac{\sigma^2}{B} \\ &\leq 2 \|\nabla f(x)\|^2 + \frac{\sigma^2}{B}.\end{aligned}$$

□

G.2 PROOF OF THEOREM 5.5

The following lemma gives time complexity for any choice of $B_1, \ldots, B_n$ and $t = (t_1, \ldots, t_n)$ in MindFlayer SGD.

Lemma G.3 (Proof in Appendix G.2.1)**.** *Assume that Assumptions 5.1, 5.2 and 5.3 hold. Let*

$$B = \sum_{i=1}^n p_i B_i \quad \textit{and} \quad \gamma = \frac{1}{2L} \min\left\{1, \frac{\varepsilon B}{\sigma^2}\right\}$$

in Algorithm 1. Then after

$$T_{\text{MindFlayerSGD}}(t) \leq \max_{i\in[n]}\{B_i \left(\tau_i + t_i\right)\} \max\left\{1, \frac{\sigma^2}{\varepsilon B}\right\} \frac{8L\left(f(x_0) - f^{\inf}\right)}{\varepsilon}$$

seconds, the method guarantees to find an ϵ-stationary point.

Now we are ready to prove the theorem.

Theorem 5.5. *Assume that Assumptions 5.1, 5.2 and 5.3 hold. Let*

$$B = \sum_{i=1}^n p_i B_i \quad \textit{and} \quad \gamma = \frac{1}{2L} \min\left\{1, \frac{\varepsilon B}{\sigma^2}\right\}$$

in Algorithm 1. Let $t = (t_1, \ldots, t_n)$, $t_1, \ldots, t_n \geq 0$. Without loss of generality assume that $0 < \tau_1 + t_1 \leq \cdots \leq \tau_n + t_n$. Let

$$t(m) = \left(\sum_{j=1}^m \frac{p_j}{\tau_j + t_j}\right)^{-1} \left(S + \sum_{j=1}^m p_j\right),$$

where $S = \max\left\{1, \frac{\sigma^2}{\varepsilon}\right\}$. *Let* $m^* = \arg\min_{m\in[n]} t(m)$, *if there are several minimizers we take the smallest one. Put*

$$B_i = \lceil b_i \rceil, \quad b_i = \begin{cases} \frac{t(m^*)}{\tau_i + t_i} - 1, & \text{if } i \leq m^*, \\ 0, & \text{if } i > m^*. \end{cases}$$

Then, MindFlayer SGD *guarantees to find an* ϵ*-stationary point after*

$$T_{\text{MindFlayerSGD}}(t) \leq 8 \times \min_{m\in[n]} \left\{ \left(\frac{1}{m} \sum_{j=1}^{m} \frac{p_j}{\tau_j + t_j} \right)^{-1} \left(\frac{S}{m} + \frac{1}{m} \sum_{j=1}^{m} p_j \right) \frac{\Delta L}{\varepsilon} \right\}$$

seconds.

Proof. First we show that B_i-s are valid choice, i.e. $b_i > 0$ for $i \leq m^*$. If $m^* = 1$, then $t(1) = \frac{\tau_1 + t_1}{p_1}(S + p_1)$, thus $b_1 = \frac{S}{p_1} > 0$. If $m^* > 1$, then, by its definition, $t(m^*) < t(m^* - 1)$. This implies

$$\left(\sum_{j=1}^{m^*} \frac{p_j}{\tau_j + t_j} \right)^{-1} \left(S + \sum_{j=1}^{m^*} p_j \right) < \left(\sum_{j=1}^{m^*-1} \frac{p_j}{\tau_j + t_j} \right)^{-1} \left(S + \sum_{j=1}^{m^*-1} p_j \right),$$

leading to

$$\left(\sum_{j=1}^{m^*-1} \frac{p_j}{\tau_j + t_j} \right) \left(S + \sum_{j=1}^{m^*} p_j \right) < \left(\sum_{j=1}^{m^*} \frac{p_j}{\tau_j + t_j} \right) \left(S + \sum_{j=1}^{m^*-1} p_j \right)$$

and

$$p_{m^*} \left(\sum_{j=1}^{m^*} \frac{p_j}{\tau_j + t_j} \right) < \frac{p_{m^*}}{\tau_{m^*} + t_{m^*}} \left(S + \sum_{j=1}^{m^*} p_j \right).$$

From the last inequality, we get that $\tau_{m^*} + t_{m^*} < t(m^*)$, thus $b_i \geq b_{m^*} > 0$ for all $i \leq m^*$.

It remains to find the time complexity with these choices of B_i. From Lemma G.3, we have that the time complexity is

$$T_{\text{MindFlayerSGD}}(t) \leq \max_{i\in[n]} \{B_i (\tau_i + t_i)\} \max\left\{1, \frac{\sigma^2}{\varepsilon B}\right\} \frac{8\Delta L}{\varepsilon}.$$

Then,

$$\max_{i\in[n]} \{B_i (\tau_i + t_i)\} \leq \max_{b_i \neq 0} \{(b_i + 1)(\tau_i + t_i)\} = t(m^*).$$

On the other hand

$$\begin{aligned} B = \sum_{i=1}^{n} p_i B_i \geq \sum_{i=1}^{n} p_i b_i &= \sum_{i=1}^{m^*} t(m^*) \frac{p_i}{\tau_i + t_i} - \sum_{i=1}^{m^*} p_i \\ &= \left(\sum_{j=1}^{m^*} \frac{p_j}{\tau_j + t_j} \right)^{-1} \left(S + \sum_{j=1}^{m^*} p_j \right) \sum_{i=1}^{m^*} \frac{p_i}{\tau_i + t_i} - \sum_{i=1}^{m^*} p_i = S \geq \frac{\sigma^2}{\varepsilon}. \end{aligned}$$

Therefore, the time complexity is

$$\begin{aligned} T_{\text{MindFlayerSGD}}(t) &\leq t(m^*) \frac{8\Delta L}{\varepsilon} \\ &= \min_{m\in[n]} \left\{ \left(\sum_{j=1}^{m} \frac{p_j}{\tau_j + t_j} \right)^{-1} \left(S + \sum_{j=1}^{m} p_j \right) \right\} \frac{8\Delta L}{\varepsilon}. \end{aligned}$$

□

G.2.1 Proof of Lemma G.3

Lemma G.3. *Assume that Assumptions 5.1, 5.2 and 5.3 hold. Let* $B = \sum_{i=1}^n p_i B_i$ *and* $\gamma = \frac{1}{2L}\min\left\{1, \frac{\varepsilon B}{\sigma^2}\right\}$ *in Method 1. Then after*

$$T_{\text{MindFlayerSGD}}(t) \leq \max_{i\in[n]}\{B_i(\tau_i + t_i)\}\max\left\{1, \frac{\sigma^2}{\varepsilon B}\right\}\frac{8L\left(f(x_0) - f^{\inf}\right)}{\varepsilon}$$

seconds, the method guarantees to find an ϵ*-stationary point.*

Proof. Let $T_i^j(t_i)$ be the random time taken by client i in the j-th attempt of calculating gradient estimator. We have

$$T_i^j(t_i) = \begin{cases} \tau_i + \eta_i^j, & \text{if } \eta_i^j \leq t_i, \\ \tau_i + t_i, & \text{if } \eta_i^j > t_i. \end{cases} \tag{14}$$

Thus, the random time taken for client i to finish it's all b_i trials is

$$\mathcal{T}_i(t_i) := \sum_{j=1}^{B_i} T_i^j(t_i) \leq B_i(\tau_i + t_i). \tag{15}$$

Finally, let $\mathcal{T}$ be the random time required for one iteration of MindFlayer SGD. We get

$$\mathcal{T} = \max_{i\in[n]}\mathcal{T}_i(t_i) \leq \max_{i\in[n]}\{B_i(\tau_i + t_i)\}. \tag{16}$$

It remains to multiply $\mathcal{T}$ with the number of iterations K given by Theorem 5.4. □

H PROOFS FOR HETEROGENEOUS REGIME

H.1 PROOF OF THEOREM C.1

Here, we rewrite Vecna SGD (Algorithm 3) in a classical SGD way.

Algorithm 8 Vecna SGD

1: **Input:** starting point x^0, stepsize γ, time budgets $t_1, \ldots, t_n \geq 0$, batch sizes $b_1, \ldots, b_n \geq 0$,
2: **for** $k = 0, 1, \ldots, K-1$ **do**
3: $\quad g^k = \frac{1}{n}\sum_{i=1}^n \frac{1}{p_i B_i}\sum_{j=1}^{B_i} I\left(\eta_i^j \leq t_i\right)\nabla f_i\left(x^k; \xi_i^j\right)$
4: $\quad x^{k+1} = x^k - \gamma g^k$
5: **end for**

where $p_i = F(t_i) = P(\eta_i \leq t_i)$.

To prove the theorem we need to establish some properties of the gradient estimator. First, we need an unbiased estimator.

Lemma H.1 (Proof in Appendix H.1.1)**.** *The gradient estimator in Algorithm 8 given by*

$$g(x) := \frac{1}{n}\sum_{i=1}^n \frac{1}{p_i B_i}\sum_{j=1}^{B_i} I\left(\eta_i^j \leq t_i\right)\nabla f_i\left(x; \xi_i^j\right)$$

is unbiased, i.e., $\mathbb{E}[g(x)] = \nabla f(x)$ *for all* $x \in \mathbb{R}^d$.

Next, we obtain an upper bound for the variance of this estimator.

Lemma H.2 (Proof in Appendix H.1.2)**.** *The gradient estimator in Algorithm 8 given by*

$$g(x) := \frac{1}{n}\sum_{i=1}^{n}\frac{1}{p_i B_i}\sum_{j=1}^{B_i} I\left(\eta_i^j \le t_i\right)\nabla f_i\left(x;\xi_i^j\right)$$

satisfies

$$\mathbb{E}\left[\left\|g(x)^2\right\|\right] \le \frac{2L\Delta}{n}\max_{i\in[n]}\left\{\frac{1-p_i}{p_i B_i}\right\} + \|\nabla f(x)\|^2 + \frac{\sigma^2}{n^2}\sum_{i=1}^{n}\frac{1}{p_i B_i} + \frac{2L}{n}\max_{i\in[n]}\left\{\frac{1-p_i}{p_i B_i}\right\}\Delta^{\inf}.$$

We are ready to prove Theorem C.1. First, let us restate the theorem.

Theorem C.1. *Assume that Assumptions 5.1, 5.2 and 5.3 hold for function f_i for all $i \in [n]$. Let*

$$\gamma = \min\left\{\frac{1}{\sqrt{L\alpha K}}, \frac{1}{L\beta}, \frac{\varepsilon}{2L\zeta}\right\}$$

in Algorithm 3. Then after

$$K \ge \frac{12\Delta L}{\varepsilon}\max\left\{\beta, \frac{12\Delta\alpha}{\varepsilon}, \frac{2\zeta}{\varepsilon}\right\},$$

iterations, the method guarantees that $\min_{0\le k\le K}\mathbb{E}\left[\left\|\nabla f(x^k)\right\|^2\right] \le \varepsilon$*, and*

$$\alpha = \frac{L}{n}\max_{i\in[n]}\left\{\frac{1-p_i}{p_i B_i}\right\}, \quad \beta = 1, \quad \zeta = \frac{\sigma^2}{n^2}\sum_{i=1}^{n}\frac{1}{p_i B_i} + \frac{2L}{n}\max_{i\in[n]}\left\{\frac{1-p_i}{p_i B_i}\right\}\Delta^{\inf}.$$

Proof. Note that Algorithm 3 can be viewed as a special case of classical stochastic gradient descent (SGD), as reformulated in Algorithm 8. We need to verify that the gradient estimator fulfills the conditions required by classical SGD (Theorem E.2). The two preceding lemmas address this requirement precisely. Specifically, Lemma H.1 confirms that the gradient estimator used in Algorithm 8 is unbiased, while Lemma H.2 verifies that the variance of this estimator meets the conditions specified in Assumption E.1. Consequently, it remains to apply Theorem E.2. □

H.1.1 Proof of Lemma H.1

Lemma H.1.1. *The gradient estimator in Algorithm 8 given by*

$$g(x) := \frac{1}{n}\sum_{i=1}^{n}\frac{1}{p_i B_i}\sum_{j=1}^{B_i} I\left(\eta_i^j \le t_i\right)\nabla f_i\left(x;\xi_i^j\right)$$

is unbiased, i.e., $\mathbb{E}\left[g(x)\right] = \nabla f(x)$ *for all* $x \in \mathbb{R}^d$*.*

Proof. This follows from direct computation:

$$\begin{aligned}
\mathbb{E}\left[g(x)\right] &= \mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n}\frac{1}{p_iB_i}\sum_{j=1}^{B_i} I\left(\eta_i^j \le t_i\right)\nabla f_i\left(x;\xi_i^j\right)\right] \\
&= \frac{1}{n}\sum_{i=1}^{n}\frac{1}{p_iB_i}\sum_{j=1}^{B_i}\mathbb{E}\left[I\left(\eta_i^j \le t_i\right)\nabla f_i\left(x;\xi_i^j\right)\right] \\
&\overset{(\eta_i^j \perp \xi_i^j)}{=} \frac{1}{n}\sum_{i=1}^{n}\frac{1}{p_iB_i}\sum_{j=1}^{B_i}\mathbb{E}\left[I\left(\eta_i^j \le t_i\right)\right]\mathbb{E}\left[\nabla f_i\left(x;\xi_i^j\right)\right] \\
&= \frac{1}{n}\sum_{i=1}^{n}\frac{1}{p_iB_i}\sum_{j=1}^{B_i} p_i\nabla f_i(x) \\
&= \frac{1}{n}\sum_{i=1}^{n}\nabla f_i(x) \\
&= \nabla f(x).
\end{aligned}$$

□

H.1.2 Proof of Lemma H.2

Lemma H.2. *The gradient estimator in Algorithm 8 given by*

$$g(x) := \frac{1}{n}\sum_{i=1}^{n}\frac{1}{p_iB_i}\sum_{j=1}^{B_i} I\left(\eta_i^j \le t_i\right)\nabla f_i\left(x;\xi_i^j\right)$$

satisfies

$$\mathbb{E}\left[\left\|g(x)^2\right\|\right] \le \frac{2L\Delta}{n}\max_{i\in[n]}\left\{\frac{1-p_i}{p_iB_i}\right\} + \left\|\nabla f(x)\right\|^2 + \frac{\sigma^2}{n^2}\sum_{i=1}^{n}\frac{1}{p_iB_i} + \frac{2L}{n}\max_{i\in[n]}\left\{\frac{1-p_i}{p_iB_i}\right\}\Delta^{\inf}.$$

Proof. Since η_i^j and ξ_i^j are independent of each other for all $i \in [n]$ and j, we have

$$\mathrm{Var}\left(g(x)\right) = \frac{1}{n^2}\sum_{i=1}^{n}\frac{1}{p_i^2B_i^2}\sum_{j=1}^{B_i}\mathrm{Var}\left(I\left(\eta_i^j \le t_i\right)\nabla f_i\left(x;\xi_i^j\right)\right),$$

then we use the fact that

$$\mathrm{Var}\left(XY\right) = \mathrm{Var}\left(X\right)\mathrm{Var}\left(Y\right) + \mathrm{Var}\left(X\right)\mathbb{E}\left[Y\right]^2 + \mathrm{Var}\left(Y\right)\mathbb{E}\left[X\right]^2,$$

where X and Y are independent random variables. Hence, we obtain the following bound on the variance

$$\begin{aligned}
\mathrm{Var}\left(I\left(\eta_i^j \le t_i\right)\nabla f_i\left(x;\xi_i^j\right)\right) &\le p_i(1-p_i)\sigma^2 + p_i\left(1-p_i\right)\left\|\nabla f_i(x)\right\|^2 + \sigma^2p_i^2 \\
&= p_i\sigma^2 + p_i\left(1-p_i\right)\left\|\nabla f_i(x)\right\|^2.
\end{aligned}$$

As a result, the variance of $g(x)$ is bounded by

$$\begin{aligned}
\mathrm{Var}\left(g(x)\right) &\le \frac{1}{n^2}\sum_{i=1}^{n}\frac{1}{p_i^2B_i^2}\sum_{j=1}^{B_i}\left(p_i\sigma^2 + p_i\left(1-p_i\right)\left\|\nabla f_i(x)\right\|^2\right) \\
&= \frac{1}{n^2}\sum_{i=1}^{n}\frac{1}{p_iB_i}\left(\sigma^2 + \left(1-p_i\right)\left\|\nabla f_i(x)\right\|^2\right).
\end{aligned}$$

Finally

$$\begin{aligned}\mathbb{E}\left[\left\|g(x)^2\right\|\right] &= \operatorname{Var}\left(g(x)\right) + \left\|\mathbb{E}\left[g(x)\right]\right\|^2 \\ &\leq \left\|\nabla f(x)\right\|^2 + \frac{1}{n^2}\sum_{i=1}^n \frac{1-p_i}{p_i B_i}\left\|\nabla f_i(x)\right\|^2 + \frac{\sigma^2}{n^2}\sum_{i=1}^n \frac{1}{p_i B_i}.\end{aligned}$$

Next we use $\left\|\nabla f_i(x)\right\|^2 \leq 2L\left(f_i(x) - f_i^{\inf}\right)$, thus

$$\begin{aligned}\mathbb{E}\left[\left\|g(x)^2\right\|\right] &\leq \frac{2L}{n^2}\sum_{i=1}^n \frac{1-p_i}{p_i B_i}\left(f_i(x) - f_i^{\inf}\right) + \left\|\nabla f(x)\right\|^2 + \frac{\sigma^2}{n^2}\sum_{i=1}^n \frac{1}{p_i B_i} \\ &\leq \frac{2L}{n}\max_{i\in[n]}\left\{\frac{1-p_i}{p_i B_i}\right\}\frac{1}{n}\sum_{i=1}^n\left(f_i(x) - f_i^{\inf}\right) + \left\|\nabla f(x)\right\|^2 + \frac{\sigma^2}{n^2}\sum_{i=1}^n \frac{1}{p_i B_i} \\ &= \frac{2L}{n}\max_{i\in[n]}\left\{\frac{1-p_i}{p_i B_i}\right\}\left(f(x) - f^{\inf}\right) + \left\|\nabla f(x)\right\|^2 \\ &\quad + \frac{\sigma^2}{n^2}\sum_{i=1}^n \frac{1}{p_i B_i} + \frac{2L}{n}\max_{i\in[n]}\left\{\frac{1-p_i}{p_i B_i}\right\}\frac{1}{n}\sum_{i=1}^n\left(f^{\inf} - f_i^{\inf}\right).\end{aligned}$$

□

H.2 PROOF OF THEOREM C.2

The following lemma gives time complexity for any choice of $B_1, \ldots, B_n$ and $t = (t_1, \ldots, t_n)$ in Vecna SGD.

Lemma H.3 (Proof in Appendix H.2.1)**.** *Assume that Assumptions 5.1, 5.2 and 5.3 hold for function f_i for all $i \in [n]$. Let $\gamma = \min\left\{\frac{1}{\sqrt{L\alpha K}}, \frac{1}{L}, \frac{\varepsilon}{2L\zeta}\right\}$ in Algorithm 3. Then after*

$$T_{\text{VecnaSGD}}(t) \leq \max_{i\in[n]}\{B_i\left(\tau_i + t_i\right)\}\frac{12\Delta L}{\varepsilon}\max\left\{1, \frac{12\Delta\alpha}{\varepsilon}, \frac{2\zeta}{\varepsilon}\right\}$$

seconds, where the method guarantees to find an ϵ-stationary point, where $\Delta = f(x_0) - f^{\inf}$ and

$$\alpha = \frac{L}{n}\max_{i\in[n]}\left\{\frac{1-p_i}{p_i B_i}\right\}, \quad \zeta = \frac{\sigma^2}{n^2}\sum_{i=1}^n \frac{1}{p_i B_i} + \frac{2L}{n}\max_{i\in[n]}\left\{\frac{1-p_i}{p_i B_i}\right\}\Delta^{\inf}.$$

Now we are ready to prove the theorem.

Theorem C.2. *Assume that Assumptions 5.1, 5.2 and 5.3 hold for function f_i for all $i \in [n]$. Let $\gamma = \min\left\{\frac{1}{\sqrt{L\alpha K}}, \frac{1}{L}, \frac{\varepsilon}{2L}\right\}$ in Algorithm 3, where*

$$\alpha = \frac{L}{n}\max_{i\in[n]}\left\{\frac{1-p_i}{p_i B_i}\right\}, \quad \zeta = \frac{\sigma^2}{n^2}\sum_{i=1}^n \frac{1}{p_i B_i} + \frac{2L}{n}\max_{i\in[n]}\left\{\frac{1-p_i}{p_i B_i}\right\}\Delta^{\inf}.$$

Let $t = (t_1, \ldots, t_n)$, $t_1, \ldots, t_n \geq 0$. Without loss of generality assume that $0 < \tau_1 + t_1 \leq \cdots \leq \tau_n + t_n$. Let

$$T = \tau_n + t_n + \left[\frac{1}{n}\sum_{i=1}^n \frac{\tau_i + t_i}{p_i}\right]\frac{\sigma^2}{n\varepsilon} + \max_{i\in[n]}\left\{\frac{1-p_i}{p_i}\left(\tau_i + t_i\right)\right\}\frac{L\left(\Delta + \Delta^{\inf}\right)}{n\varepsilon}.$$

Put

$$B_i = \lceil b_i \rceil, \quad b_i = \frac{T}{\tau_i + t_i}.$$

Then, Vecna SGD *guarantees to find an ϵ-stationary point after*

$$T_{\mathsf{VecnaSGD}}(t) \leq 288 \times \frac{\Delta L}{\varepsilon}\left(\tau_n + t_n + \left[\frac{1}{n}\sum_{i=1}^n \frac{\tau_i + t_i}{p_i}\right]\frac{\sigma^2}{n\varepsilon} + \left[\frac{1}{n}\sum_{i=1}^n \frac{1-p_i}{p_i}\left(\tau_i + t_i\right)\right]\frac{\Delta L}{n\varepsilon}\right)$$

seconds.

Proof. Since we have $b_i \geq 1$ for all $i \in [n]$, we get

$$\max_{i\in[n]}\{B_i\left(\tau_i + t_i\right)\} \leq \max_{b_i \neq 0}\{\left(b_i + 1\right)\left(\tau_i + t_i\right)\} \leq 2\max_{i\in[n]}\{b_i\left(\tau_i + t_i\right)\} = 2T.$$

It remains to apply Lemma H.3. We get

$$\begin{aligned}\frac{12\Delta\alpha}{\varepsilon} &= \frac{12\Delta L}{\varepsilon n}\max_{i\in[n]}\left\{\frac{1-p_i}{p_i B_i}\right\} \leq \frac{12\Delta L}{\varepsilon n}\max_{i\in[n]}\left\{\frac{1-p_i}{p_i b_i}\right\} \\ &= \frac{12\Delta L}{n\varepsilon}\frac{1}{T}\max_{i\in[n]}\left\{\frac{1-p_i}{p_i}\left(\tau_i + t_i\right)\right\} \leq 12,\end{aligned}$$

and

$$\begin{aligned}\frac{2\zeta}{\varepsilon} &= \frac{2\sigma^2}{\varepsilon n^2}\sum_{i=1}^n \frac{1}{p_i B_i} + \frac{4L\Delta^{\inf}}{n\varepsilon}\max_{i\in[n]}\left\{\frac{1-p_i}{p_i B_i}\right\} \\ &\leq \frac{2\sigma^2}{\varepsilon n^2}\sum_{i=1}^n \frac{1}{p_i b_i} + \frac{4L\Delta^{\inf}}{n\varepsilon}\max_{i\in[n]}\left\{\frac{1-p_i}{p_i b_i}\right\} \\ &\leq \frac{2\sigma^2}{n\varepsilon}\frac{1}{T}\frac{1}{n}\sum_{i=1}^n \frac{\tau_i + t_i}{p_i} + \frac{4L\Delta^{\inf}}{n\varepsilon}\frac{1}{T}\max_{i\in[n]}\left\{\frac{1-p_i}{p_i}\left(\tau_i + t_i\right)\right\} \\ &\leq 4.\end{aligned}$$

Finally, we get that Algorithm 3 returns a solution after

$$\begin{aligned}T_{\mathsf{MindFlayerSGD}}(t) &\leq \max_{i\in[n]}\{B_i\left(\tau_i + t_i\right)\}\frac{12\Delta L}{\varepsilon}\max\left\{1, \frac{12\Delta\alpha}{\varepsilon}, \frac{2\zeta}{\varepsilon}\right\} \\ &\leq 288\frac{\Delta L}{\varepsilon}T \\ &\leq 288\frac{\Delta L}{\varepsilon}\left(\tau_n + t_n + \left[\frac{1}{n}\sum_{i=1}^n \frac{\tau_i + t_i}{p_i}\right]\frac{\sigma^2}{n\varepsilon} + \max_{i\in[n]}\left\{\frac{1-p_i}{p_i}\left(\tau_i + t_i\right)\right\}\frac{L\left(\Delta + \Delta^{\inf}\right)}{n\varepsilon}\right)\end{aligned}$$

seconds. □

H.2.1 Proof of Lemma H.3

Lemma H.3. *Assume that Assumptions 5.1, 5.2 and 5.3 hold for function f_i for all $i \in [n]$. Let $\gamma = \min\left\{\frac{1}{\sqrt{L\alpha K}}, \frac{1}{L}, \frac{\varepsilon}{2L\zeta}\right\}$ in Algorithm 3. Then after*

$$T_{\mathsf{VecnaSGD}}(t) \leq \max_{i\in[n]}\{B_i\left(\tau_i + t_i\right)\}\frac{12\Delta L}{\varepsilon}\max\left\{1, \frac{12\Delta\alpha}{\varepsilon}, \frac{2\zeta}{\varepsilon}\right\}$$

seconds, where the method guarantees to find an ϵ-stationary point, and

$$\alpha = \frac{L}{n}\max_{i\in[n]}\left\{\frac{1-p_i}{p_i B_i}\right\}, \quad \zeta = \frac{\sigma^2}{n^2}\sum_{i=1}^n \frac{1}{p_i B_i} + \frac{2L}{n}\max_{i\in[n]}\left\{\frac{1-p_i}{p_i B_i}\right\}\Delta^{\inf}.$$

Proof. Let $T_i^j(t_i)$ be the random time taken by client i in the j-th attempt of calculating gradient estimator. We have

$$T_i^j(t_i) = \begin{cases} \tau_i + \eta_i^j, & \text{if } \eta_i^j \leq t_i, \\ \tau_i + t_i, & \text{if } \eta_i^j > t_i. \end{cases} \tag{17}$$

Thus, the random time taken for client i to finish it's all B_i trials is

$$\mathcal{T}_i(t_i) := \sum_{j=1}^{B_i} T_i^j(t_i) \leq B_i\left(\tau_i + t_i\right). \tag{18}$$

Finally, let $\mathcal{T}$ be the random time required for one iteration of Vecna SGD. We get

$$\mathcal{T} = \max_{i\in[n]} \mathcal{T}_i(t_i) \leq \max_{i\in[n]} \{B_i\left(\tau_i + t_i\right)\}. \tag{19}$$

It remains to multiply $\mathcal{T}$ with the number of iterations K given by Theorem C.1. □