Byzantine-Resilient Learning Beyond Gradients: Distributing Evolutionary Search (Full Report)

Andrei Kucharavy*
andrei.kucharavy@hevs.ch
IC School, EPFL
Lausanne, Switzerland

Matteo Monti
matteo.monti@epfl.ch
IC School, EPFL
Lausanne, Switzerland

Rachid Guerraoui
rachid.guerraoui@epfl.ch
IC School, EPFL
Lausanne, Switzerland

Ljiljana Dolamic
ljiljana.dolamic@ar.admin.ch
Cyber-Defence Campus, armasuisse
Thun, Switzerland

ABSTRACT

Modern machine learning (ML) models are capable of impressive performances. However, their prowess is not due only to the improvements in their architecture and training algorithms but also to a drastic increase in computational power used to train them.

Such a drastic increase led to a growing interest in distributed ML, which in turn made worker failures and adversarial attacks an increasingly pressing concern. While distributed byzantine resilient algorithms have been proposed in a differentiable setting, none exist in a gradient-free setting.

The goal of this work is to address this shortcoming. For that, we introduce a more general definition of byzantine-resilience in ML - the *model-consensus*, that extends the definition of the classical distributed consensus. We then leverage this definition to show that a general class of gradient-free ML algorithms - $(1, \lambda)$-Evolutionary Search - can be combined with classical distributed consensus algorithms to generate gradient-free byzantine-resilient distributed learning algorithms. We provide proofs and pseudo-code for two specific cases - the Total Order Broadcast and proof-of-work leader election.

To our knowledge, this is the first time a byzantine resilience in gradient-free ML was defined, and algorithms to achieve it - were proposed.

CCS CONCEPTS

• **Computing methodologies → Genetic programming**; *Machine learning algorithms*; *Machine learning*; • **Theory of computation → Distributed algorithms**.

KEYWORDS

Evolutionary Search, Gradient-free optimization, Distributed machine learning, Byzantine Fault Tolerance

*Corresponding author; Now at HES-SO Valais-Wallis, Sierre, Switzerland

ACM Reference Format:
Andrei Kucharavy, Matteo Monti, Rachid Guerraoui, and Ljiljana Dolamic. 2023. Byzantine-Resilient Learning Beyond Gradients: Distributing Evolutionary Search (Full Report). In *Genetic and Evolutionary Computation Conference Companion (GECCO '23 Companion), July 15–19, 2023, Lisbon, Portugal.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3583133.3590719

1 INTRODUCTION

Over the last decade, the machine learning field underwent transformative growth, achieving and surpassing human capabilities in a variety of domains, ranging from image classification and facial recognition to image generation to strategy games [31, 33, 47, 48]. Beyond impressive performance in the academic setting, Machine Learning (ML) and Artificial Intelligence (AI) progressively became central to numerous tasks, ranging from translation to autonomous driving [29, 67]. Perhaps the most impressive recent development is the arrival of conversational agents driven by Large Neural Language Models (LLMs) [20, 43, 52].

However, the emergence of ML and AI as powerful and widely accessible tools is not only due to the discovery of better model architectures and algorithms to train them but also due to the increasing computational capabilities and data volumes available to train them. Empirical demonstrations of the performance of stochastic gradient descent (SGD) applied to artificial neural networks (ANNs) were already available by mid-1980s [35, 62], and theoretically explained by early 1990s [28]. However, it wasn't until sufficient computational power became affordable and sufficiently large training datasets were accumulated that the machine learning revolution truly started [36]. This joint scaling of models and dataset sizes and resources invested in training them still drives ML progress today, notably for LLMs [19, 27, 30].

1.1 Gradient-Free Learning

The ongoing machine learning revolution has not affected all the domains equally, given that best-performing algorithms rely on ANNs and gradient descent. Image processing was one of the first domains that saw early breakthroughs [33], more recently followed by natural language processing as means to make text interpretation and generation continuous through word embedding and positional encoding were perfected [40, 60]. However, a number of problems have so far evaded conversion to a continuous formulation, notably in the control theory domain.

A set of approaches have been developed for such problems, generally referred to as *black-box*, *zero-order*, or *gradient-free* optimization methods. Representing a diversity of underlying approaches - from Evolutionary Strategies and Genetic Algorithms to Swarm Particle Optimization, Simulated Annealing to sample-derived local gradients - they have nonetheless faded from the general ML community attention in recent years. Two exceptions are Reinforcement Learning (RL) [57] and empirical gradients. RL was made famous through its super-human performance in strategy games [47, 53, 61], it became the default approach to gradient-free problems, whereas empirical gradients approximate local gradients through empirical sampling.

However, neither of the approaches scales to large, overparameterized models, known to be needed to train models solving complex problems [37]. Empirical gradient estimation struggles in high dimensions and around saddle points, and is hard to parallelize due to the need for a synchronized round of gradient evaluations pooling, which is expensive computation and communication-wise for larger models. Similar problems exist as well in reinforcement learning. Notable failure modes are the cases where the observations ("rewards") are sparse ("long time horizons") and noisy. In such settings, the policy reward estimator's variance will increase to the point where the learning process becomes unstable. Such instability is not limited to pathological settings - even in cases it performs well, RL requires a hyperparameter space search to find a working training regime even for problems where it performs well [10, 46]. Such instability is not a fluke either. There are theoretical reasons why approaches that reduce learning in a non-differentiable setting to a differentiable one would underperform compared to gradient-free black-box optimization approaches [39, 46].

This is particularly relevant now, given that the latest development in the LLM field is conversation agents, which rely on optimization based on discrete feedback to align their behavior on user expectations and non-differentiable layers of hard attention to solve the issues with rule-following that plague them [20, 43, 51].

1.2 Evolutionary Search

Limitations of RL and empirical gradients approaches led to an increased interest in gradient-free black-box optimization algorithms, notably Evolutionary Algorithms (EAs). Introduced shortly after the SGD itself, [17], EAs are expected to scale well with more computational power, just like the SGD itself. This was confirmed experimentally, including on ranges of control tasks where they outperformed RL approaches, all while allowing better scaling [10, 46, 54, 56]. These empirical results led to a renewed interest in EAs in ML and the discovery of cases where they outperformed RL and other black-boxes approaches, such as model architecture design [44].

Despite its simplicity, the first evolutionary algorithm proposed by [17] in 1966 is still able to match and out-perform RL approaches on complex problems [44, 54, 56]. Akin to SGD, it is an iterative optimization algorithm. However, instead of calculating the local gradient, it samples the neighborhood of the current model parameters to find a better solution and retains the single best one among all sampled ones. It is formally known as $(1, \lambda)$-Evolutionary Search ($(1, \lambda)$-ES) or Evolutionary Strategies[1].

In addition to their reasonable performance, $(1, \lambda)$-ES class algorithms have an additional advantage - scalability. As a population algorithm, every parameter sample can be evaluated independently, and an optimal parameter update - shared among all workers once a desired population of candidate updates has been sampled. Here we focus on the simplest implementation of the $(1, \lambda)$-ES class, which we will refer to as $(1, \lambda)$-ES for simplicity. A modification of that algorithm by [46] reduces the message size to about a dozen bytes regardless of the model size, by leveraging the fact that random parameter perturbations can be deterministically derived by a pseudo-random number generator from a random seed, meaning that sharing only the random seed is sufficient. Unlike gradient-based learning, $(1, \lambda)$-ES allows any worker to verify the validity of an update proposed by another vector with a single forward pass, which is the property we leverage to combine classical distributed consensus algorithms with $(1, \lambda)$-ES to create byzantine-resilient distributed versions of $(1, \lambda)$-ES.

Finally, since at no point $(1, \lambda)$-ES requires a back-propagation, it allows for non-differentiable layers, such as hard attention or deterministic rules, to be included in the model architecture [64]. This makes it interesting even in the setting allowing for gradient-based learning because, unlike differentiable layers, deterministic rules can provide deterministic guarantees on AI model decisions, which is critical in high-stakes applications. In particular, for LLMs, it has the potential of solving the long-term instruction retaining problem, currently limiting their application [20, 51].

1.3 Byzantine-Resilience in Machine Learning

The increasing size of ML models has also made them impossible to train or even run on single machines, making model parallelization and distribution an increasingly pressing issue [4]. With the increase of the computing nodes involved in the training, the chances for an arbitrary complex error to occur increase, making fault tolerance a prime concern. In the field of distributed computing, the tolerance to such faults is known as "Byzantine Tolerance", with the name derived from the seminal paper that introduced that type of faults [34].

The field of machine learning allowing for such "Byzantine" fault tolerance led to the emergence of the field of byzantine-resilient machine learning [7, 8]. Unfortunately, the definition introduced in the process is specific to differentiable manifolds and focuses on the setting where every node trains the same model, only has partial access to the data, and shares non-verifiable gradients calculated on that data. By introducing novel gradient aggregation rules (GARs) for the parameter server, they were able to prove that a byzantine fault impact could be limited to at most a deviation of angle α on the final parameter update for the fraction f of Byzantine workers ((α, f)-Byzantine Resilience).

[1]Given multiple conflicting names for different EA algorithms, we have here adopted the taxonomy from [24]. Notable cases are the use of the "Genetic Algorithm" name to designate $(1, \lambda)$-ES in prior literature, that we reserve to algorithms including "chromosomes" or "recombination" as per the original article [21], or use of Evolutionary Search for Natural Evolutionary Search (NES), that is closer to empirical gradient approach than ES proper [63].

The reason authors had to introduce a new definition of byzantine-resilience rather than to re-use existing ones, is that the latter are poorly suited to the distributed learning setting. If approached from the *Do-All problem* perspective [15], parameter update vectors cannot be verified without repeating the whole computation, meaning that byzantine-resilience would require several workers to perform redundant update calculation. In the setting where training a single model can cost millions in electricity costs alone (c.f., e.g., [1]), direct redundancy is an unrealistic assumption.

In the distributed gradient-based learning, the (α, f)-Byzantine Resilience hence remained the predominant paradigm and has been further expanded to provide guarantees for models trained in a more general distributed setting than federated learning [13, 14, 23, 65, 66].

1.4 Our Contribution

Our main contribution is showing that Evolutionary Search is can be adapted to work as a byzantine-resilient distributed optimization algorithm in a non-differentiable setting.

Specifically, we show that by introducing a new definition of distributed consensus in the ML setting, we can leverage the existing literature on byzantine-resilient distributed computing. In turn, by using the established primitives of the total order broadcast and proof-of-work probabilistic consensus primitives [9, 45] we propose two algorithms for distributed evolutionary search - in permissioned (closed) and permissionless (open) settings and prove the bounds on the computational overhead imposed by distributing the Evolutionary Search.

Interestingly, our new definition of distributed consensus - the *Model-Consensus* generalizes the (α, f)-Byzantine Resilience introduced by [7, 8] and directly interfaces with the more general definition of computational consensus.

2 PRELIMINARIES

2.1 Learning Setting

Our problem consists in learning a general function $f \in \mathbb{F}$, mapping inputs $x \in \mathbb{X}$ to outputs $y \in \mathbb{Y}$, determined by parameters $\boldsymbol{\theta}$, noted $f(., \boldsymbol{\theta})$[2]. A scalar performance metric $\mathcal{L}$ is associated to the function and can be computed for each input/output pair in the training and validation sets $\mathcal{L}(f(x, \boldsymbol{\theta}), y)$. We denote $\mathcal{L}_\theta$, an aggregate performance metric on all input-output pairs. Without loss of generality, $\mathcal{L}$ can correspond conversely to loss, accuracy, total reward, fitness, or another metric of the model performance. The goal of learning is to find parameters that optimize that value. This process can be referred to as parameter optimization, parameter space search, or training. For the sake of simplicity, we adopt the convention that $\mathcal{L}$ is a loss that we seek to minimize, although, in the context of evolution, $-\mathcal{L}$ will be occasionally referred to as fitness due to historical reasons. $\{\mathcal{L}_\theta\}$ will be referred to as loss landscape (conversely fitness landscape). Finally, $v\boldsymbol{L}_\theta$ will refer to the *loss vector* obtained by concatenating the losses for all the input/output pairs $(x, y) \in \mathbb{X} \times \mathbb{Y}$ in the training set for model parameters $\boldsymbol{\theta}$. While we assume a non-differentiable setting, we still assume a smooth loss landscape, ie $\exists k \in \mathbf{R}$ such that $\forall(\boldsymbol{\theta}_i, \boldsymbol{\theta}_j)$, $\frac{|\mathcal{L}_{\theta_i} - \mathcal{L}_{\theta_j}|}{\|\boldsymbol{\theta}_i - \boldsymbol{\theta}_j\|} < k$.

[2]For machine learning, we follow the common notation introduced in [22].

Given that we are interested in distributing the training phase, $\mathcal{L}_\theta$ without further additional notation designates the aggregated performance metric on the training dataset. We assume as well that every worker has access to all of the training data and that $\mathcal{L}_\theta$ is computed in a deterministic manner by each worker, given θ. This setting is different from the one used in distributing the gradient computation, given that the difficulty for the $(1, \lambda)$-ES algorithm is to find a valid update.

2.2 Model-Consensus and ϵ-Optimality

In the machine learning context, the consensus problem is for a set of processes $p \in \Pi$ to *decide* on a common value of model parameters $\boldsymbol{\theta}$ based on model values correct processes can evaluate individually $\boldsymbol{\theta}_p$. A correct process can decide on a value at most once every training session.

For the sake of readability, given the process-worker equivalence, we will be referring to processes p as *workers* and the ensemble of workers trying to solve the machine learning consensus problem for a given task Π - as *worker pool.*

A machine learning consensus protocol must satisfy the following conditions:

- *Liveness*: Each correct worker must eventually decide on a value of $\boldsymbol{\theta}$
- *Consistency*: No two correct workers can decide on a different $\boldsymbol{\theta}$.
- *Validity* (Extended): For all correct workers, only a $\boldsymbol{\theta}$ proposed by a correct process can be decided upon.

Given that ML model training is distributed to improve parameter space search, we expect the workers to propose different values $\boldsymbol{\theta}_p$, so the extended validity is essential. Moreover, we expect some parameters to correspond to a better loss value, and we want our workers to decide on a value of parameters that leads to the lowest loss possible. This leads us to introduce a new constraint on the model-consensus:

- ϵ*-optimality*: If $\boldsymbol{\theta}$ satisfies $\mathcal{L}_\theta \leq min_{p \in \Pi}(\mathcal{L}_{\theta_p}) + \epsilon$, where $\epsilon \geq 0$, the consensus is ϵ*-optimal.*

A special case of ϵ-optimality is the case where $\epsilon = 0$, in which case we will refer to the consensus simply as *optimal.* The two algorithms we propose here are optimal for each update with high probability, whereas (α, f)-Byzantine Resilience [7, 8] is ϵ-optimal with $\epsilon = sin(\alpha) \cdot lr \cdot k_{lipshitz}$, where lr is the effective learning rate and $k_{lipshitz}$ - the Lipschitz constant of the loss landscape.

2.3 $(1, \lambda)$-ES Algorithm

Similar to SGD, algorithms of the $(1, \lambda)$-ES class search for optimal model parameters θ through a series of steps performing an empirical descent of the loss landscape [25]. At each step i, the value of the parameters θ_i is perturbed by a vector $\boldsymbol{\beta}$[3] sampled from a normal distribution $\mathcal{N}(0, I)$ and scaled to a learning rate σ. A number $(k \in [1, .., N])$ of $\boldsymbol{\beta}$ values are tested. The one that improves the model the most $(k_{update} = arg\min_k \mathcal{L}(\theta_i + \sigma\boldsymbol{\beta_k}))$ is retained and becomes the base for the next search $(\theta_{i+1} := \theta_i + \sigma\boldsymbol{\beta_{k_{update}}})$. We refer to $\sigma\boldsymbol{\beta_{k_{update}}}$ as an *update vector,* $\sigma\boldsymbol{\beta_k}$ as *candidate update*

[3]Other works tend to use ϵ to denote it, whereas we use a Greek letter close to the neighborhood notation in topology to avoid confusion with ϵ of ϵ-optimality

vectors and $\boldsymbol{\theta_i + \sigma\beta_k}$ as *candidate parameters*. No update will occur if no tested vectors have improved loss, so only vectors such as $\mathcal{L}(\theta_i + \sigma\boldsymbol{\beta_k}) < \mathcal{L}(\theta_i) + \nu$, where $\nu \geq 0$ is a parameter controlling for a trade-off between random noise due to sampling and gradient descent - a minimal improvement to be achieved before an update is triggered. We will refer to $\sigma\boldsymbol{\beta_k}$ for which this property is true - a *valid update vector*.

2.4 Adapting $(1, \lambda)$-ES for Distributed Setting

As we mentioned in the introduction, an important improvement to the $(1, \lambda)$-ES is for nodes to share only the random seeds used to derive candidate update vector deterministically with a pseudo-random numbers generator, allowing update sharing with short messages (given that randoms seeds are <16 bytes for most ML libraries), and once bundled with the loss parameter, allows any correct node to verify the proposed candidate update vector. In all that follows, we will assume that mode of derivation and refer to such a random seed as a *candidate update vector seed*, noted as $\mathfrak{S}_{\beta_k}$. Formalizing the section above, we assume as well that we have an access to a random generator that is capable of turning a random seed into a non-scaled update vector ($RG : \mathfrak{S}_{\beta_k} \longrightarrow \beta_k$).

To facilitate the proofs for the permissionless setting, we introduce an additional modification of the $(1, \lambda)$-ES algorithm that is run by the workers p. Specifically, to ensure strategy-proofness and more closely match existing proof-of-work, we add a combined hashing of the loss and update vector seed, assumed to be a positive integer below a certain maximal value $\mathfrak{B}_{max}$ (eg. the largest integer that can be encoded with the number of bits in a hash). We refer to the hash of $(\mathfrak{S}_{\beta_k}, \boldsymbol{L}_{\theta_k})$ as *θ-block score* $\mathfrak{B}_{\theta_k}$. In the proof-of-work consensus, it is used as a scheduler for leader election, which is triggered when $\mathfrak{B}_\theta < \mathfrak{B}_{target}$, where $\mathfrak{B}_{target}$ is the value set to control the frequency of leader election given the size of the worker pool and the frequency of evaluation.

The pseudo-code for the complete evolutionary search algorithm is presented in the listing 1.

3 PERMISSIONED DISTRIBUTED EVOLUTIONARY SEARCH

The intuition behind the permissioned setting is to leverage the verifiability of proposed update vectors in $(1, \lambda)$-ES to re-use existing results in classical distributed algorithms. Specifically, given the iterative nature of $(1, \lambda)$-ES, we need the total order broadcast to be able to order the iterative steps between all correct workers.

```
Abstraction:
  EvolutionarySearcher, Instance es

Interface:
  - **Request** <es.Start | $\theta_i$>: starts search
  - **Request** <es.Stop>: ends search
  - **Indication** <es.BestHash | $\theta_i$, $\mathfrak{S}_{\beta_k}$, $\mathcal{L}_{\theta_i+\sigma\beta_k}$>:
    a new seed with a valid hash was found
  - **Indication** <es.BestLoss| $\theta_i$, $\mathfrak{S}_{\beta_k}$, $\mathcal{L}_{\theta_i+\sigma\beta_k}$>:
    a new valid update vector seed was found
  - **Procedure** es.follow($\theta_i$, $\mathfrak{S}_{\beta_k}$) -> $\theta_i + \sigma\beta_k$
    derive candidate parameters for a seed
  - **Procedure** es.evaluate($\theta_i$, $\mathfrak{S}_{\beta_k}$) ->
  ($\mathfrak{B}_{\theta_i+\sigma\beta_k}$, $\mathcal{L}_{\theta_i+\sigma\beta_k}$): evaluates the
    hash and loss of a candidate seed

Algorithm:
  **Implements**:
    EvolutionarySearcher, instance es;
  **Parameters**:
    $\mathcal{L}$: loss function;
    $\sigma$: search radius;
    $\nu$: minimal loss score improvement;
    $\mathfrak{B}_{target}$: target hash threshold;
  **procedure** reset():
    target = $\emptyset$;
    best_hash = {seed: $\emptyset$, score: $+\infty$};
    best_loss = {seed: $\emptyset$, score: $+\infty$};
  **upon** <es.Start | $\theta_i$>:
    reset();
    target = $\theta_i$
  **upon** <es.Stop>:
    reset();
  **procedure** es.follow($\theta_i$, $\mathfrak{S}_{\beta_k}$):
    **If** $\mathfrak{S}_{\beta_k}$ == $\emptyset$:
      return $\theta_i$;
    **Else**:
      $\beta_k$ = RG($\mathfrak{S}_{\beta_k}$);
      return $\theta_i + \sigma\beta_k$;
  **procedure** es.evaluate($\theta_i$, $\mathfrak{S}_{\beta_k}$):
    $\mathcal{L}_{\theta_i+\sigma\beta_k}$, $v\boldsymbol{L}_{\theta_i+\sigma\beta_k}$ = eval($f_{\theta_i+\sigma\beta_k}$);
    $\mathfrak{B}_{\theta_i+\sigma\beta_k}$ = hash($\mathcal{L}_{\theta_i+\sigma\beta_k}$, $v\boldsymbol{L}_{\theta_i+\sigma\beta_k}$);
    return ($\mathfrak{B}_{\theta_i+\sigma\beta_k}$, $\mathcal{L}_{\theta_i+\sigma\beta_k}$);
  **upon** target != null:
    seed = rand();
    $\mathfrak{B}_{\theta_i+\sigma\beta_k}$, $\mathcal{L}_{\theta_i+\sigma\beta_k}$ = es.evaluate($\theta_i$, $\mathfrak{S}_{\beta_k}$);
    **If** $\mathfrak{B}_{\theta_i+\sigma\beta_k} < \mathfrak{B}_{target}$:
      best_hash = {seed: $\mathfrak{S}_{\beta_k}$, score: $\mathfrak{B}_{\theta_i+\sigma\beta_k}$};
      trigger <es.BestHash | $\theta_i$, $\mathfrak{S}_{\beta_k}$, $\mathfrak{B}_{\theta_i+\sigma\beta_k}$;
    **If** $\mathcal{L}(\theta_i + \sigma\beta_k) < \mathcal{L}(\theta_i) + \nu$:
      best_loss = {seed: $\mathfrak{S}_{\beta_k}$, score: $\mathcal{L}_{\theta_i+\sigma\beta_k}$};
      trigger <es.BestLoss | $\theta_i$, $\mathfrak{S}_{\beta_k}$, $\mathcal{L}_{\theta_i+\sigma\beta_k}$>;
```

Listing 1: Single Worker Evolutionary Search

THEOREM 3.1. *The algorithm in listing 2 implements a machine learning consensus protocol that is Byzantine-resilient under the same assumptions as the Total Order Broadcast algorithm used and is optimal with probability bound from above by* $\frac{\Delta|\Pi|}{NT_{eval,average}}$, *where* Δ *is the time needed to perform a Total Order Broadcast, N - the expected number of tries to find a valid update seed and* $T_{eval,average}$ *is the average time needed by a worker to evaluate a candidate update seed.*

PROOF. The Total Order Broadcast ensures that the valid update seeds $\mathfrak{S}_{\beta_k}$ are delivered to all correct workers in the same order after the workers were initialized to the same starting parameters value θ_0. Assuming that a valid update seed exists $\forall i \in [0..Z - 1]$, it will be eventually found and broadcasted by a correct worker.

Liveness: Each correct worker will eventually decide on the final $\theta_Z = \theta_0 + \sum_{i=0}^{Z-1} \sigma\beta_{i,first}$, where $\sigma\beta_{i,first}$ is the update vector derived from the first valid update seed for point θ_i.

Consistency: Thanks to the Total Order Broadcast, $\forall i \in [0..Z - 1]\beta_{i,first}$ are the same values for all workers, and hence for each correct worker, the final parameters of the model $\theta_Z = \theta_0 + \sum_{i=0}^{Z-1} \sigma\beta_{i,first}$ are identical.

Extended Validity: By construction, the first correct worker to have its proposed seed successfully broadcast will have its update

```
Abstraction:
  PermissionedEvolutionarySearch, instance ps

Uses:
  - EvolutionarySearcher, instance es,
      parameters ($\mathcal{L}$, $\sigma$, $\nu$, _)
  - TotalOrderBroadcast, instance tob

Interface:
  - Indication <ps.Output | point>:
    parameters found by the evolutionary search

Algorithm:
  Implements:
     PermissionedEvolutionarySearch, instance ps
  Parameters:
    $\mathcal{L}$: loss function;
    $\sigma$: search radius;
    $\nu$: loss threshold for update;
    $\theta_0$: starting point of the search;
    Z: number of search steps;
  upon <ps.Init>:
    target = $\theta_0$;
    steps = 0;
    trigger <es.Start | target>;
  upon <es.BestLoss | $\theta_i$, $\mathfrak{S}_{\beta_k}$, $\mathcal{L}_{\theta_i+\sigma\beta_k}$>:
    If $\theta_i$ == target And $\mathcal{L}(\theta_i+\sigma\beta_k) < \mathcal{L}(\theta_i)+\nu$:
      trigger <tob.Broadcast |
                ["ValidLoss", $\theta_i$, $\mathfrak{S}_{\beta_k}$]>;
  upon <tob.Deliver |
        source_es ["ValidLoss", $\theta_i$, $\mathfrak{S}_{\beta_k}$]>:
    If $\theta_i$ == target:
      (_, $\mathcal{L}_{\theta_i+\sigma\beta_k}$) =
      es.evaluate($\theta_i$, $\mathfrak{S}_{\beta_k}$))
        // verify that the seed is valid indeed
      If $\mathcal{L}_{\theta_i+\sigma\beta_k} < \mathcal{L}_{\theta_i}+\nu$:
        target = es.follow($\theta_i$, $\mathfrak{S}_{\beta_k}$);
          // is actually $\theta_{i+1} = \theta_i + \sigma\beta_k$
        steps = steps + 1;
        If steps < Z:
          trigger <es.Start | target>;
        Else:
          trigger <es.Stop>;
          trigger <ps.Output | target>;
      Else:
        trigger <tob.Ban | source_es>;
          // optional penalty for misbehaving
```

Listing 2: Permissioned Distributed Search

vector $\sigma\beta_{i,first}$ accepted. A seed that has not been successfully broadcasted cannot be accepted.

Probabilistic Optimality: By construction, at every step, upon the reception of a valid update seed $\mathfrak{S}_{\beta_{i,first}}$ through Total Order Broadcast, a correct worker will switch to searching for a valid update vector for the new parameters $\theta_i+1 = \theta_i+\sigma\beta_{i,first}$. The only way a better update at a given step becomes available without being broadcast first is if one becomes available during the total broadcast. The probability of that happening is proportional to the number of seed evaluations occurring before the broadcast completes times the probability of finding a seed above the threshold and better than the seed in the broadcast. The former is bound by the number of evaluations a worker pool can perform during the broadcast ($\frac{|\Pi|\Delta}{T_{eval,average}}$), whereas the second is bound by the chance of finding a valid seed, which, in case if the seed in broadcast is equal exactly to the validity threshold is $\frac{1}{N}$.

□

4 PERMISSIONLESS DISTRIBUTED EVOLUTIONARY SEARCH

4.1 Proof-of-Work Mechanism for Probabilistic Consensus

The probabilistic consensus algorithm through proof-of-work (PoW) was initially proposed in the Bitcoin blockchain whitepaper [41], as a mechanism to achieve a probabilistic consensus through a leader election process tied to the amount of computational power actively committed to the election process. The principle of the election mechanism leverages the cryptographically secure hash function partial inversion. Based on the information provided by the prior leader election (often the hash of the prior block head), information to be broadcast by the next leader (often the root of the Merkle tree of transactions to be cleared), correct workers try to guess a random string (nonce) that once added to those two values would lead to a hash in the desired domain (for simplicity, $0 < \mathfrak{B} < \mathfrak{B}_{target} < \mathfrak{B}_{max}$). In turn, once a node finds a valid nonce, its leadership can be validated by other nodes by performing a single hash with the found nonce. This process is referred to as "mining" and each new leader election - as a "block minting", and assuming sufficient time between leader elections to allow the previous block value to propagate ($\mathfrak{B}_{target}$ is adjusted based on the number of workers for that reason), ensuring an eventual election of a correct worker as a leader with high probability, assuming that the majority of computational power is controlled by correct workers [42, 45].

Unfortunately, the increasing popularity of PoW-based blockchains led to a combination of a large number of computationally powerful workers joining it and consequently to the difficulty threshold being increased to the point where PoW became a serious environmental problem [55]. This led to heavy criticism of PoW consensus and other protocols - such as proof-of-stake [32] - to be promoted as less harmful alternatives for permissionless distributed consensus.

An alternative approach consisted in trying to highjack the proof of work to instead perform some useful work that would absorb computational resources independently of PoW-based blockchains. Such algorithms - *useful proof-of-work* (UPoW) - have unfortunately been hard to find, given the volume of computational power currently invested into PoW they would need to absorb and strict constraints on PoW to be usable: provably hard-to-find easy-to-verify updates, low communication complexity, and message weight and easily adjustable puzzle difficulty.

4.2 Permissionless Distributed Evolutionary Search as Proof-of-Work

However, given the ever-growing demand for computational power in machine learning, parameter space search problems are sufficiently common to leverage the computational power available to PoW consensus algorithms. Conversely, the distributed $(1, \lambda)$-ES seems to fit the constraints on the UPoWs, given that while hard to find, valid update vectors are straightforwards to validate and that communication overhead in-between iterative steps of $(1, \lambda)$-ES only contains $(\mathfrak{S}_{\beta_k}, \boldsymbol{L}_{\theta_k})$ - candidate update vector seed and associate loss.

To simplify the proofs and enable a direct mechanism for complexity adjustment, rather than using a valid update itself as a proof for leader election, we instead use the θ-block score $\mathfrak{B}_{\theta_k}$, while propagating the best found valid update seed and associated loss $(\mathfrak{S}_{\beta_{k_{update}}}, \mathcal{L}_{\beta_{k_{update}}})$ with the same mechanisms as Merkel tree roots. Intuitively, this is a distributed equivalent of $(1, \lambda)$-ES with a set sampling population, except with the size decided by the expected candidate update samples between leader elections.

Given the variety of available blockchain protocols, we will abstract them away in the same we abstracted the total order broadcast in the permissioned setting and assume they implement an interface described in listing 4.

Theorem 4.1. *Algorithm in Listing 3 is a valid proof-of-work and is a machine learning consensus protocol optimal with probability bound from above by* $\frac{\Delta|\Pi|}{NT_{eval,average}} + e^{\Omega(\delta^2 g^2 d)}$ $\forall \delta > 0$ *if* $g^2\alpha > (1+\delta)\gamma$ *after* d *block added on top of the block minted during the step Z.* Δ *and is the time needed to propagate a block or a value, respectively*[4], *N - the expected number of tries to find a valid update seed, and* $T_{eval,average}$ *is the average time needed by a worker to evaluate a candidate update seed,* α *and* γ *- collective minting rates of correct and faulty nodes and* $g = e^{-\alpha\Delta}$*, the propagation delay penalty for correct nodes.*

Proof. The algorithm in the listing 3 is a valid proof-of-work because the block minting mechanism is equivalent to a partial inversion of a cryptographic function with an unavoidable loss function evaluation overhead.

Since the algorithm in the listing 3 is a valid proof-of-work, the Nakamoto consensus regarding the block propagated at the step Z of ps after d blocks were added on top of it will not change with probability $1 - e^{\Omega(\delta^2 g^2 d)}$ for any $\delta > 0$ as long as $g^2\alpha > (1+\delta)\gamma$, as per [42, 45].

The Nakamoto consensus blockchain blocks are available to all correct workers and are ordered in a unique way for all correct workers. By replacing Total Order Broadcast in the proof of by the blockchain segment read containing blocks corresponding to steps 0 to Z, the proof for 3.1 applies.

□

The intuitive explanation of proof is that the algorithm in listing 3 will fail to register the best random seed with the best candidate update loss in only two cases. First, if the blockchain forked and the bock with the best candidate update random seed ended up on a dead branch. This case occurs with the probability $1 - e^{\Omega(\delta^2 g^2 d)}$. Second, if the candidate update seed with the best loss is found within the time δ from the block update, accounted for by term $\frac{\Delta|\Pi|}{NT_{eval,average}}$. While this is possible if the task supplied is too easy for the size of the blockchain, there is likely a tighter bound, given that if many valid update seeds were found as a single block was

[4]Given that the propagation of a value and block involves evaluating a candidate update, depending on the neighbor propagation topology, $Delta_{block/val}$ can be $O(|\Pi| \cdot T_{eval,average})$, $O(log(|\Pi|) \cdot T_{eval,average})$ or $O(T_{eval,slowest})$. For the sake of generalizability, we keep the same notation as previously

```
Abstraction:
  PermissionlessEvolutionarySearch, instance ps

Uses:
  - EvolutionarySearcher, instance es,
      parameters ($\mathcal{L}$, $\sigma$, $\nu$, $\mathfrak{B}_{target}$)
  - Blockchain, instance bl

Interface:
  - Indication <ps.Output | point>:
    parameters found by the evolutionary search
  - Procedure ps.processNewBlock([$\theta_i$, target, steps,
    es.best_hash, es.best_loss])

Algorithm:
  Implements:
     PermissionlessEvolutionarySearch, instance ps
  Parameters:
    // same as in permissioned
  upon <ps.Init>:
    // same as in permissioned
  procedure processNewBlock([$\theta_i$, target, steps,
    es.best_hash, es.best_loss]):
    If steps < Z:
      trigger <es.Start | target>;
    Else:
      trigger <es.Stop>;
      trigger <ps.Output | target>;
      trigger <bl.loadNext>;
  upon <es.BestLoss | $\theta_i$, $\mathfrak{S}_{\beta_k}$, $\mathcal{L}_{\theta_i+\sigma\beta_k}$>:
    If{$\theta_i$ == target And $\mathcal{L}(\theta_i+\sigma\beta_k) < \mathcal{L}(\theta_i)+\nu$:
      trigger <bl.sendValue |
              ["ValidLoss", $\theta_i$, $\mathfrak{S}_{\beta_k}$, $\mathcal{L}(\theta_i+\sigma\beta_k)$]>;
  upon <bl.deliverValue |
        source_es ["ValidLoss", $\theta_i$,
        $\mathfrak{S}_{\beta_k}$, $\mathcal{L}_{declared}(\theta_i+\sigma\beta_k)$]>:
    If $\theta_i$ == target
       And $\mathcal{L}_{declared}(\theta_i+\sigma\beta_k)$ < es.best_loss[score]:
      (_, $\mathcal{L}_{validated}\theta_i+\sigma\beta_k$) =
      es.evaluate($\theta_i$, $\mathfrak{S}_{\beta_k}$));
        // verify that the sender is not lying
      If $\mathcal{L}_{declared}(\theta_i+\sigma\beta_k)$ == $\mathcal{L}_{validated}(\theta_i+\sigma\beta_k)$:
        es.best_loss = {seed: $\mathfrak{S}_{\beta_k}$, score: $\mathcal{L}_{\theta_i+\sigma\beta_k}$};
        trigger <bl.sendValue |
          ["ValidLoss", $\theta_i$, $\mathfrak{S}_{\beta_k}$, $\mathcal{L}(\theta_i+\sigma\beta_k)$]>;
  upon <es.BestHash | $\theta_i$, $\mathfrak{S}_{\beta_k}$, $\mathfrak{B}_{\theta_i+\sigma\beta_k}$>:
    If $\theta_i$ == target And $\mathfrak{B}_{\theta_i+\sigma\beta_k} < \mathfrak{B}_{target}$:
      target = es.follow($\theta_i$, es.best_loss[seed]);
      steps += 1;
      trigger block = <bl.mintBlock |
                      [$\theta_i$, target, steps,
                      es.best_hash, es.best_loss]>;
      trigger <bl.propagateBlock | block>;
      ps.ProcessNewBlock(block);
    upon <bl.deliverBlock |
          source_es [$\theta_i$, target, steps,
          source_es.best_hash, source_es.best_loss]>:
      If $\theta_i$==target:
        ($\mathfrak{B}_{\theta_i+\sigma\beta_k}$, _) =
          es.evaluate($\theta_i$, source_es.best_hash[seed]);
        If{$\theta_i$ == target And $\mathfrak{B}_{\theta_i+\sigma\beta_k} < \mathfrak{B}_{target}$:
          target = target;
          steps = steps;
          trigger <bl.propagateBlock | block>;
          ps.ProcessNewBlock(block);
```

Listing 3: Permissionless Distributed Search

```
Abstraction:
  Blockchain, instance bl

Interface:
  - Procedure bl.sendValue:
    worker proposes to its neighbours a value to be
    included in the next block
  - Procedure bl.deliverValue:
    delivers a value proposed for inclusion into the
    next block from a neighbour
  - Procedure bl.mintBlock:
    allows a worker to mint a new block that would
    include the best valid updates received
  - Procedure bl.propagageBlock:
    allows a worker to suggests a newly found block
    to be propagated a neighbours
  - Procedure bl.deliverBlock:
    delivers a block proposed for propagation from
    a neighbour
  - Procedure bl.loadNext:
    loads the next distributed search task queued
    in blockchain
```

Listing 4: Expected Blockchain interface

mined, the last one is not necessarily the one that will reach the best loss.

5 DISCUSSION AND RELATED WORK

5.1 Byzantine-Resilient Distributed Machine Learning

As we mentioned in the introduction, while multiple mechanisms to distribute machine learning algorithms were proposed, the only one accounting for Byzantine faults is the (α, f)-Byzantine Resilient learning proposed by [7, 8]. While initially formulated in the context of federated learning with a centralized parameter aggregation server, it has been developed to allow byzantine fault tolerance in the general distributed setting, under realistic assumptions [13].

Compared to our approach, a downside of the (α, f)-Byzantine Resilience is their dependence on direct gradients and a requirement for the model to be sufficiently small for the gradient vector sharing to not become a bottleneck. Modern ML and AI models are often sufficiently large for data transfer time to not be negligible and need to be synchronized often enough to avoid parameter drift between models. For instance, the GPT-3 generative language model has several hundreds of GBs of parameters, and even when split into single attention heads requires each parameter synchronization to transfer several GBs of parameter values.

Conversely, an advantage of (α, f)-Byzantine Resilience is its ability to train on the data that's different on each worker node, assuming the data is uniform across workers. While this assumption is not necessarily true in all settings, our distributed $(1, \lambda)$-ES does not natively support the data distribution[5]. The main problem addressed by distributing $(1, \lambda)$-ES is the difficulty to find a single valid update at each step of the optimization task, particularly in cases where the parameter space is in a very high dimension, leading to long valid update vector search, even when single candidate update vector evaluation is itself fast. This setting is specifically the one in which EAs have been successfully applied in modern machine learning [10, 54, 56]. Notably, the permissionless UPoW $(1, \lambda)$-ES version only makes sense in that setting, complemented with a high iteration number to ensure that most of the communication is carrying only candidate update seeds and associated loss values and minimize training data transfer.

Finally, while there are other distributed approaches to gradient-free optimization, notably for Support Vector Machines (SVMs) [18, 59] and the Genetic Algorithm [5, 21], none to our knowledge allow for byzantine faults.

5.2 Useful proof of work

Given that the computational and energy costs of PoW consensus were already well-known by the mid-2010s [55], multiple attempts were made to leverage the PoW to do useful work. The first proposal was made in 2017, leveraging a set of problems in computational geometry for which the search of a solution is $O(n^2)$ hard and verification is $O(n)$ hard [3]. Unfortunately, insufficient demand and lack of difficulty adjustment mechanism meant that this PoW was impractical. The same year another set of problems - partitioned linear algebra on very large non-sparse matrices was proposed by [50]. Unfortunately, that PoW did not take either, given the rarity of problems involving such matrices and amounts of data (TBs) that would need to be transferred in the process. Finally, still the same year, a highly general framework for turning any computationally intensive task into PoW challenges has been proposed [68]. Relying on the trusted hardware - Intel SGX - it initially showed a great performance but was rapidly rendered obsolete by the rarity of hardware supporting Intel SGX and then the demise of Intel SGX in the wake of Spectre vulnerabilities [11, 12].

The next iteration of the search for a useful proof of work started in 2019, focusing on NP-hard problems, notably the traveling salesman problem and machine learning. While some success has been achieved by using TSP in the context of the container ship sailing route optimization [26], despite being NP-hard TSP has several probabilistic heuristics available and no initial estimation of hardness for a specific problem, making it non-strategy-proof and hence not a suitable PoW for blockchain purposes.

Machine learning PoW has been attempted as well, however, all the approaches we are aware of tried to use gradient-based machine learning and ran into the fundamental issue of non-verifiability of gradient calculation, leading them to waste resources through replication or to leave their PoW non-strategy-proof and often to have the model training itself to be vulnerable to adversaries. Examples of such approaches are Proof of Learning (PoLe) [38], which is essentially a race to a predefined accuracy, and model hyperparameter sweep PoW [2]. Neither schema introduces any replication, leaving ML model vulnerable to attackers, and neither schema is strategy-proof, given that a worker with more computational resources or a good heuristic could consistently "win" each of those competitions.

Proof of Search [49] occupies an interesting spot among the proposed useful PoW in that it doesn't propose a useful PoW per se

[5]Given the linear nature of the loss wrt to data, it is possible to design variants of distributed $(1, \lambda)$-ES that would require a sufficient number of nodes to confirm that a candidate update vector achieves an acceptable loss improvement on their data as well. This would however require additional assumptions and a more restrictive learning setting and is out of the scope of this paper.

but rather a way to transform suitable useful PoW into a blockchain. In that, it is complementary to our approach since it provides a blueprint to implement the blockchain interface described in the listing 4.

Perhaps most relevant to the evolutionary search aspect of our algorithm, two evolutionary proofs of useful work have been proposed, both using the genetic algorithm [6, 58]. One focuses on solving the TSP problem by using a genetic algorithm directly, whereas the other one tries to create a framework similar to the Proof of Search but using genetic algorithms specifically and applied to NP-hard problems such as TSP and Knapsack. Not only are such approaches vulnerable to the issues mentioned above in the context of TSP blockchains, but the genetic algorithm also implies a collaborative phase of parameter mixing during the "chromosome" "cross-over"[6], which runs against the competitive nature of the Proof-of-Work and adds a layer of communication complexity. In addition to that, both approaches require messages between workers to carry whole parameter update vectors, adding a communication bottleneck for larger models.

6 CONCLUSION

In this paper, we present a new definition of distributed machine learning consensus - the *Model-Consensus*, that generalizes the previously proposed (α, f)-Byzantine Resilience and is applicable both in differentiable and non-differentiable settings.

We then present two distributed versions of the $(1, \lambda)$-Evolutionary Search algorithm, both reaching a model-consensus in a gradient-free setting. One leveraged the classical distributed algorithms abstraction of Total Order Broadcast to achieve a consensus in a permissioned setting, whereas the other used the Proof-of-Work leader election consensus to achieve the same result in a permissionless setting.

To our knowledge, our model-consensus definition is the first definition of Byzantine resilient consensus in machine learning that covers both gradient-free and gradient-based learning while generalizing the previously proposed (α, f)-Byzantine Resilience and allowing for direct compatibility with the classical distributed algorithms consensus definition.

To our knowledge, the two algorithms we propose are the first Byzantine-resilient gradient-free learning algorithms and the first byzantine-resilient evolutionary algorithms. Likewise, our permissionless distributed evolutionary search algorithm is the first useful proof-of-work algorithm that minimizes the overhead compared to traditional proof-of-work.

While proposed algorithms work for any black-box optimization problems, they are most suited for high-dimensional optimization problems where the evaluation of a single updated parameter set is quick, but the sheer number of dimensions makes the search for a valid update excessively slow for a single worker, with a notable application being the neuroevolution of large ANNs. We foresee that such a setting is of particular relevance to multipurpose super neural networks, such as PathNet [16], as well as for conversational agents derived from LLMs [20, 51].

[6]As we mention in the introduction, we use Genetic Algorithm only to designate EAs that has "chromosome" and "cross-over" phases, consistently with the nomenclature introduced in [24]

Finally, more accessible and reliable byzantine-resilient machine learning allows a variety of entities to poll together their computational resources to train models they could not have trained individually, which has the potential of democratizing state-of-the-art ML research in a non-differentiable setting.

ACKNOWLEDGMENTS

We would like to thank the Cyber-Defence Campus, armasuisse W+T, VBS for the Distinguished CYD Post-Doctoral Fellowship to AK (ARAMIS CYD-F-2021004), as well as Fabien Salvi and France Faille (EFPL) for their technical and administrative support and the anonymous reviewers for their useful feedback.

REFERENCES

[1] 2020. OpenAI's GPT-3 Language Model: A Technical Overview. https://lambdalabs.com/blog/demystifying-gpt-3/
[2] Alejandro Baldominos and Yago Saez. 2019. Coin.AI: A Proof-of-Useful-Work Scheme for Blockchain-Based Distributed Deep Learning. *Entropy* 21, 8 (2019), 723. https://doi.org/10.3390/e21080723
[3] Marshall Ball, Alon Rosen, Manuel Sabin, and Prashant Nalini Vasudevan. 2017. Proofs of Useful Work. *IACR Cryptol. ePrint Arch.* (2017), 203. http://eprint.iacr.org/2017/203
[4] Paul Barham, Aakanksha Chowdhery, Jeff Dean, Sanjay Ghemawat, Steven Hand, Daniel Hurt, Michael Isard, Hyeontaek Lim, Ruoming Pang, Sudip Roy, et al. 2022. Pathways: Asynchronous distributed dataflow for ml. *Proceedings of Machine Learning and Systems* 4 (2022), 430–449.
[5] Theodore C. Belding. 1995. The Distributed Genetic Algorithm Revisited. In *Proceedings of the 6th International Conference on Genetic Algorithms, Pittsburgh, PA, USA, July 15-19, 1995*, Larry J. Eshelman (Ed.). Morgan Kaufmann, 114–121.
[6] Francesco Bizzaro, Mauro Conti, and Maria Silvia Pini. 2020. Proof of Evolution: leveraging blockchain mining for a cooperative execution of Genetic Algorithms. In *IEEE International Conference on Blockchain, Blockchain 2020, Rhodes, Greece, November 2-6, 2020.* IEEE, 450–455. https://doi.org/10.1109/Blockchain50366.2020.00065
[7] Peva Blanchard, El Mahdi El Mhamdi, Rachid Guerraoui, and Julien Stainer. 2017. Brief Announcement: Byzantine-Tolerant Machine Learning. In *PODC 2017*, Elad Michael Schiller and Alexander A. Schwarzmann (Eds.). ACM, 455–457.
[8] Peva Blanchard, El Mahdi El Mhamdi, Rachid Guerraoui, and Julien Stainer. 2017. Machine Learning with Adversaries: Byzantine Tolerant Gradient Descent. In *Advances in Neural Information Processing Systems 30: NeurIPS 2017*, Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (Eds.). 119–129. https://proceedings.neurips.cc/paper/2017/hash/f4b9ec30ad9f68f89b29639786cb62ef-Abstract.html
[9] Jo-Mei Chang and Nicholas F. Maxemchuk. 1984. Reliable Broadcast Protocols. *ACM Trans. Comput. Syst.* 2, 3 (1984), 251–273.
[10] Edoardo Conti, Vashisht Madhavan, Felipe Petroski Such, Joel Lehman, Kenneth O. Stanley, and Jeff Clune. 2018. Improving Exploration in Evolution Strategies for Deep Reinforcement Learning via a Population of Novelty-Seeking Agents. In *Advances in Neural Information Processing Systems 31: NeurIPS 2018*, Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett (Eds.). 5032–5043.
[11] Intel Corporation. 2018. CVE-2017-5715. Available from MITRE, CVE-ID CVE-2017-5715.. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-5715
[12] Intel Corporation. 2018. CVE-2017-5753. Available from MITRE, CVE-ID CVE-2017-5753.. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-5753
[13] El-Mahdi El-Mhamdi, Sadegh Farhadkhani, Rachid Guerraoui, Arsany Guirguis, Lê-Nguyên Hoang, and Sébastien Rouault. 2021. Collaborative Learning in the Jungle (Decentralized, Byzantine, Heterogeneous, Asynchronous and Nonconvex Learning). In *Advances in Neural Information Processing Systems 34: NeurIPS 2021*, Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan (Eds.). 25044–25057.
[14] El-Mahdi El-Mhamdi, Rachid Guerraoui, Arsany Guirguis, Lê Nguyên Hoang, and Sébastien Rouault. 2020. Genuinely Distributed Byzantine Machine Learning. In *PODC '20: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, August 3-7, 2020*, Yuval Emek and Christian Cachin (Eds.). ACM, 355–364. https://doi.org/10.1145/3382734.3405695
[15] Antonio Fernández, Chryssis Georgiou, Alexander Russell, and Alexander A. Shvartsman. 2005. The Do-All problem with Byzantine processor failures. *Theor. Comput. Sci.* 333, 3 (2005), 433–454. https://doi.org/10.1016/j.tcs.2004.06.034
[16] Chrisantha Fernando, Dylan Banarse, Charles Blundell, Yori Zwols, David Ha, Andrei A Rusu, Alexander Pritzel, and Daan Wierstra. 2017. Pathnet: Evolution channels gradient descent in super neural networks. *arXiv preprint arXiv:1701.08734*

(2017).

[17] L.J. Fogel, A.J. Owens, and M.J. Walsh. 1966. *Artificial intelligence through simulated evolution.* Wiley, Chichester, WS, UK.

[18] Pedro A. Forero, Alfonso Cano, and Georgios B. Giannakis. 2010. Consensus-Based Distributed Support Vector Machines. *J. Mach. Learn. Res.* 11 (2010), 1663–1707. http://portal.acm.org/citation.cfm?id=1859906

[19] Deep Ganguli, Danny Hernandez, Liane Lovitt, Nova DasSarma, Tom Henighan, Andy Jones, Nicholas Joseph, Jackson Kernion, Benjamin Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Dawn Drain, Nelson Elhage, Sheer El Showk, Stanislav Fort, Zac Hatfield-Dodds, Scott Johnston, Shauna Kravec, Neel Nanda, Kamal Ndousse, Catherine Olsson, Daniela Amodei, Dario Amodei, Tom B. Brown, Jared Kaplan, Sam McCandlish, Chris Olah, and Jack Clark. 2022. Predictability and Surprise in Large Generative Models. *CoRR* abs/2202.07785 (2022). arXiv:2202.07785 https://arxiv.org/abs/2202.07785

[20] Amelia Glaese, Nat McAleese, Maja Trebacz, John Aslanides, Vlad Firoiu, Timo Ewalds, Maribeth Rauh, Laura Weidinger, Martin Chadwick, Phoebe Thacker, Lucy Campbell-Gillingham, Jonathan Uesato, Po-Sen Huang, Ramona Comanescu, Fan Yang, Abigail See, Sumanth Dathathri, Rory Greig, Charlie Chen, Doug Fritz, Jaume Sanchez Elias, Richard Green, Sona Mokrá, Nicholas Fernando, Boxi Wu, Rachel Foley, Susannah Young, Iason Gabriel, William Isaac, John Mellor, Demis Hassabis, Koray Kavukcuoglu, Lisa Anne Hendricks, and Geoffrey Irving. 2022. Improving alignment of dialogue agents via targeted human judgements. *CoRR* abs/2209.14375 (2022). https://doi.org/10.48550/arXiv.2209.14375 arXiv:2209.14375

[21] David E. Goldberg. 1989. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, Reading, MA.

[22] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning.* MIT Press. http://www.deeplearningbook.org.

[23] Shangwei Guo, Tianwei Zhang, Han Yu, Xiaofei Xie, Lei Ma, Tao Xiang, and Yang Liu. 2021. Byzantine-resilient decentralized stochastic gradient descent. *IEEE Transactions on Circuits and Systems for Video Technology* (2021).

[24] Nikolaus Hansen, Dirk V. Arnold, and Anne Auger. 2015. Evolution Strategies. In *Springer Handbook of Computational Intelligence*, Janusz Kacprzyk and Witold Pedrycz (Eds.). Springer, 871–898. https://doi.org/10.1007/978-3-662-43505-2_44

[25] Nikolaus Hansen, Dirk V. Arnold, and Anne Auger. 2015. Evolution Strategies. In *Springer Handbook of Computational Intelligence*, Janusz Kacprzyk and Witold Pedrycz (Eds.). Springer Berlin Heidelberg, 871–898. https://doi.org/10.1007/978-3-662-43505-2_44

[26] Mohamed Haouari, Mariem Mhiri, Mazen El-Masri, and Karim Al-Yafi. 2022. A novel proof of useful work for a blockchain storing transportation transactions. *Inf. Process. Manag.* 59, 1 (2022), 102749. https://doi.org/10.1016/j.ipm.2021.102749

[27] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. 2022. Training Compute-Optimal Large Language Models. *CoRR* abs/2203.15556 (2022). https://doi.org/10.48550/arXiv.2203.15556 arXiv:2203.15556

[28] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. 1989. Multilayer feedforward networks are universal approximators. *Neural Networks* 2, 5 (1989), 359 – 366. https://doi.org/10.1016/0893-6080(89)90020-8

[29] Melvin Johnson, Mike Schuster, Quoc V. Le, Maxim Krikun, Yonghui Wu, Zhifeng Chen, Nikhil Thorat, Fernanda B. Viégas, Martin Wattenberg, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2017. Google's Multilingual Neural Machine Translation System: Enabling Zero-Shot Translation. *Trans. Assoc. Comput. Linguistics* 5 (2017), 339–351. https://doi.org/10.1162/tacl_a_00065

[30] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling Laws for Neural Language Models. *CoRR* abs/2001.08361 (2020). arXiv:2001.08361 https://arxiv.org/abs/2001.08361

[31] Tero Karras, Samuli Laine, and Timo Aila. 2019. A Style-Based Generator Architecture for Generative Adversarial Networks. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019.* Computer Vision Foundation / IEEE, 4401–4410. https://doi.org/10.1109/CVPR.2019.00453

[32] Sunny King and Scott Nadal. 2012. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. *self-published paper, August* 19, 1 (2012).

[33] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems 25: NeurIPS 2012*, Peter L. Bartlett, Fernando C. N. Pereira, Christopher J. C. Burges, Léon Bottou, and Kilian Q. Weinberger (Eds.). 1106–1114.

[34] Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. 1982. The Byzantine Generals Problem. *ACM Trans. Program. Lang. Syst.* 4, 3 (1982), 382–401. https://doi.org/10.1145/357172.357176

[35] Yann LeCun. 1985. Une procedure d'apprentissage pour reseau a seuil asymetrique. *Proceedings of Cognitiva 85* (1985), 599–604.

[36] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *nature* 521, 7553 (2015), 436–444.

[37] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. 2018. Visualizing the Loss Landscape of Neural Nets. In *Advances in Neural Information Processing Systems 31: NeurIPS 2018*, Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett (Eds.). 6391–6401. https://proceedings.neurips.cc/paper/2018/hash/a41b3bb3e6b050b6c9067c67f663b915-Abstract.html

[38] Yuan Liu, Yixiao Lan, Boyang Li, Chunyan Miao, and Zhihong Tian. 2021. Proof of Learning (PoLe): Empowering neural network training with consensus building on blockchains. *Comput. Networks* 201 (2021), 108594. https://doi.org/10.1016/j.comnet.2021.108594

[39] Luke Metz, C. Daniel Freeman, Samuel S. Schoenholz, and Tal Kachman. 2021. Gradients are Not All You Need. *CoRR* abs/2111.05803 (2021). arXiv:2111.05803 https://arxiv.org/abs/2111.05803

[40] Tomás Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. In *NeurIPS 2013*, Christopher J. C. Burges, Léon Bottou, Zoubin Ghahramani, and Kilian Q. Weinberger (Eds.). 3111–3119.

[41] Satoshi Nakamoto. 2008. Bitcoin: A Peer-to-Peer Electronic Cash System. (2008), 9.

[42] Jianyu Niu, Chen Feng, Hoang Dau, Yu-Chih Huang, and Jingge Zhu. 2019. Analysis of Nakamoto Consensus, Revisited. *IACR Cryptol. ePrint Arch.* (2019), 1225. https://eprint.iacr.org/2019/1225

[43] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F. Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. *CoRR* abs/2203.02155 (2022). https://doi.org/10.48550/arXiv.2203.02155 arXiv:2203.02155

[44] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. 2019. Regularized Evolution for Image Classifier Architecture Search. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019.* AAAI Press, 4780–4789. https://doi.org/10.1609/aaai.v33i01.33014780

[45] Ling Ren. 2019. Analysis of Nakamoto Consensus. *IACR Cryptol. ePrint Arch.* (2019), 943. https://eprint.iacr.org/2019/943

[46] Tim Salimans, Jonathan Ho, Xi Chen, and Ilya Sutskever. 2017. Evolution Strategies as a Scalable Alternative to Reinforcement Learning. *CoRR* abs/1703.03864 (2017). arXiv:1703.03864 http://arxiv.org/abs/1703.03864

[47] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. 2020. Mastering atari, go, chess and shogi by planning with a learned model. *Nature* 588, 7839 (2020), 604–609.

[48] Florian Schroff, Dmitry Kalenichenko, and James Philbin. 2015. FaceNet: A unified embedding for face recognition and clustering. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015.* IEEE Computer Society, 815–823. https://doi.org/10.1109/CVPR.2015.7298682

[49] Naoki Shibata. 2019. Proof-of-Search: Combining Blockchain Consensus Formation With Solving Optimization Problems. *IEEE Access* 7 (2019), 172994–173006. https://doi.org/10.1109/ACCESS.2019.2956698

[50] Ali Shoker. 2018. Brief Announcement: Sustainable Blockchains through Proof of eXercise. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing, PODC 2018, Egham, United Kingdom, July 23-27, 2018*, Calvin Newport and Idit Keidar (Eds.). ACM, 269–271. https://dl.acm.org/citation.cfm?id=3212781

[51] Kurt Shuster, Mojtaba Komeili, Leonard Adolphs, Stephen Roller, Arthur Szlam, and Jason Weston. 2022. Language Models that Seek for Knowledge: Modular Search & Generation for Dialogue and Prompt Completion. *CoRR* abs/2203.13224 (2022). https://doi.org/10.48550/arXiv.2203.13224 arXiv:2203.13224

[52] Kurt Shuster, Jing Xu, Mojtaba Komeili, Da Ju, Eric Michael Smith, Stephen Roller, Megan Ung, Moya Chen, Kushal Arora, Joshua Lane, Morteza Behrooz, William Ngan, Spencer Poff, Naman Goyal, Arthur Szlam, Y-Lan Boureau, Melanie Kambadur, and Jason Weston. 2022. BlenderBot 3: a deployed conversational agent that continually learns to responsibly engage. *CoRR* abs/2208.03188 (2022). https://doi.org/10.48550/arXiv.2208.03188 arXiv:2208.03188

[53] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy P. Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. 2017. Mastering the game of Go without human knowledge. *Nat.* 550, 7676 (2017), 354–359. https://doi.org/10.1038/nature24270

[54] Kenneth O. Stanley, Jeff Clune, Joel Lehman, and Risto Miikkulainen. 2019. Designing neural networks through neuroevolution. *Nat. Mach. Intell.* 1, 1 (2019), 24–35.

[55] Christian Stoll, Lena Klaaßen, and Ulrich Gallersdörfer. 2019. The carbon footprint of bitcoin. *Joule* 3, 7 (2019), 1647–1661.

[56] Felipe Petroski Such, Vashisht Madhavan, Edoardo Conti, Joel Lehman, Kenneth O. Stanley, and Jeff Clune. 2017. Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement

Learning. *CoRR* abs/1712.06567 (2017). arXiv:1712.06567 http://arxiv.org/abs/1712.06567

[57] Richard S. Sutton. 1991. Dyna, an Integrated Architecture for Learning, Planning, and Reacting. *SIGART Bull.* 2, 4 (1991), 160–163. https://doi.org/10.1145/122344.122377

[58] Willa Ariela Syafruddin, Sajjad Dadkhah, and Mario Köppen. 2019. Blockchain Scheme Based on Evolutionary Proof of Work. In *IEEE Congress on Evolutionary Computation, CEC 2019, Wellington, New Zealand, June 10-13, 2019.* IEEE, 771–776. https://doi.org/10.1109/CEC.2019.8790128

[59] Vladimir Naumovich Vapnik. 1995. *The Nature of Statistical Learning Theory.* Springer. https://doi.org/10.1007/978-1-4757-2440-0

[60] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems 30: NeurIPS 2017*, Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (Eds.). 5998–6008.

[61] Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John P. Agapiou, Max Jaderberg, Alexander Sasha Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, Yury Sulsky, James Molloy, Tom Le Paine, Çaglar Gülçehre, Ziyu Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy P. Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David Silver. 2019. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nat.* 575, 7782 (2019), 350–354. https://doi.org/10.1038/s41586-019-1724-z

[62] P. J. Werbos. 1974. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences.* Ph. D. Dissertation. Harvard University.

[63] Daan Wierstra, Tom Schaul, Jan Peters, and Jürgen Schmidhuber. 2008. Natural Evolution Strategies. In *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2008, June 1-6, 2008, Hong Kong, China.* IEEE, 3381–3387. https://doi.org/10.1109/CEC.2008.4631255

[64] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron C. Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio. 2015. Show, Attend and Tell: Neural Image Caption Generation with Visual Attention. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015 (JMLR Workshop and Conference Proceedings, Vol. 37)*, Francis R. Bach and David M. Blei (Eds.). JMLR.org, 2048–2057.

[65] Zhixiong Yang and Waheed U. Bajwa. 2019. BRIDGE: Byzantine-resilient Decentralized Gradient Descent. *CoRR* abs/1908.08098 (2019). arXiv:1908.08098 http://arxiv.org/abs/1908.08098

[66] Zhixiong Yang and Waheed U. Bajwa. 2019. ByRDiE: Byzantine-Resilient Distributed Coordinate Descent for Decentralized Learning. *IEEE Trans. Signal Inf. Process. over Networks* 5, 4 (2019), 611–627. https://doi.org/10.1109/TSIPN.2019.2928176

[67] Ekim Yurtsever, Jacob Lambert, Alexander Carballo, and K. Takeda. 2020. A Survey of Autonomous Driving: Common Practices and Emerging Technologies. *IEEE Access* 8 (2020), 58443–58469.

[68] Fan Zhang, Ittay Eyal, Robert Escriva, Ari Juels, and Robbert van Renesse. 2017. REM: Resource-Efficient Mining for Blockchains. In *26th USENIX Security Symposium, USENIX Security 2017, Vancouver, BC, Canada, August 16-18, 2017*, Engin Kirda and Thomas Ristenpart (Eds.). USENIX Association, 1427–1444. https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/zhang