PsFiT: A GPU-Accelerated Distributed Framework for Scalable Sparse Machine Learning

Alireza Olama[1,*], Andreas Lundell[1], Jan Kronqvist[2], Elham Ahmadi[3] and Jerker Björkqvist[1]

[1]Department of Information Technology, Åbo Akademi University, Rantakatu 2, 65100 Vaasa, Finland
[2]Department of Mathematics, KTH Royal Institute of Technology, Lindstedtsvägen 25, 100 44 Stockholm, Sweden
[3]School of Technology and Innovations, University of Vaasa, Wolffintie 32, 65200 Vaasa, Finland

Abstract

Training sparse and interpretable machine learning models over large distributed datasets is a fundamental challenge in modern AI applications, particularly where computational scalability and decentralized processing are critical. This paper introduces `PsFiT`, an open-source Python toolbox for distributed sparse machine learning (SML), built around a novel optimizer called the Bi-linear consensus Alternating Direction Method of Multipliers (Bi-cADMM). The toolbox supports a broad family of explicit cardinality-constrained models, including sparse linear, logistic, and softmax regression. Bi-cADMM reformulates the underlying non-convex ℓ_0 norm constraint via a bi-linear consensus approach and employs a hierarchical decomposition strategy. This design distributes computation by offloading data-intensive, parallel operations to GPUs while reserving lighter parameter-server coordination tasks for CPUs, making `PsFiT` adaptable across configurations from single nodes to distributed networks. Numerical experiments on synthetic and real-world datasets (MNIST and IMDB) demonstrate the framework's strong scalability with respect to both the number of features and data points.

Keywords

ADMM, Distributed Training, GPU Acceleration, Model Pruning

1. Introduction

In recent years, the exponential growth of data has posed significant challenges in both the design and training of Machine Learning (ML) models Verbraeken et al. [1], Bertsimas et al. [2]. Traditional approaches struggle to handle the increasing scale and complexity of datasets, leading to issues of computational inefficiency, overfitting, and lack of interpretability Bertsimas et al. [3], Olama et al. [4]. Learning *sparse* and *interpretable* ML models can be beneficial to address these challenges Olama et al. [5], Bertsimas et al. [6], Bertsimas and Mundru [7], Bertsimas and Van Parys [8], Bertsimas and King [9], Bertsimas et al. [3], Bai et al. [10].

Training sparse models pose a significant challenge, particularly in scenarios involving decentralized data storage systems, where Distributed Data-Parallel (DDP) training plays a vital role [11], or within the framework of Federated Learning (FL), where preserving data privacy and conducting decentralized computations are paramount concerns [12]. Furthermore, the mathematical optimization problems inherent in SML training tend to be non-convex, as they involve satisfying an ℓ_0 norm constraint, resulting in a solution space that is a union of finitely many subspaces [13, 14]. Numerous studies propose distributed sparse training algorithms through convex relaxations of the original non-convex problem. One popular method is distributed Lasso, where an ℓ_1 norm relaxation is used (see Boyd et al. [15] and the references therein). However, despite the favorable computational properties of Lasso, this method can have some shortcomings, which are well-studied in the statistics community Bertsimas et al. [2, 6]. For example, finding the correct sparsity pattern for general learning problems with the ℓ_1 norm penalty cannot be guaranteed.

TKTP 2026: Annual Doctoral Symposium of Computer Science, 8.—9.6.2026 Turku, Finland
*Corresponding author.
Alireza.Olama@abo.fi (A. Olama); Andreas.Lundell@abo.fi (A. Lundell); jankr@kth.se (J. Kronqvist); Elham.Ahmadi@uwasa.fi (E. Ahmadi); Jerker.Bjorkqvist@abo.fi (J. Björkqvist)
0000-0002-1787-3487 (A. Olama)
© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073
published 2026-11-02

CEUR-WS.org/Vol-4309/paper13.pdf

A few papers in the literature address the ℓ_0 norm-induced sparsity in DDP and FL scenarios. For example, Tong et al. [16] presented a federated iterative hard thresholding method for sparse FL in decentralized settings. In Olama et al. [5], the authors propose a Distributed Primal Outer Approximation (DiPOA) algorithm designed for SML problems featuring a separable structure. DiPOA is designed based on the outer approximation algorithm Duran and Grossmann [17] and the Relaxed Hybrid Alternating Direction Method of Multipliers (RH-ADMM) Olama et al. [18]. By integrating RH-ADMM into the OA framework, DiPOA distributes the computational load across multiple processors, rendering it suitable for contemporary multi-core architectures. A recent contribution Olama et al. [4] introduces the Distributed Hybrid Outer Approximation (DiHOA) algorithm, which utilizes a branch-and-bound algorithm tailored for the SML problem structure. DiHOA serves as the core algorithm in the Sparse Convex Optimization Toolkit (SCOT) proposed in Olama et al. [4].

In both papers provided by Olama et al. [5, 4], the ℓ_0 norm constraint is addressed by reformulating the original problem into a constraint-coupled Mixed Integer Programming (MIP) equivalent. Although this strategy can solve SML problems to global optimality and satisfy the data privacy constraints, the MIP-based methods face challenges when dealing with large-scale datasets, especially in high-dimensional regimes and distributed settings. In Olama et al. [19], the authors propose a distributed algorithm formulated upon a distributed gradient tracking Carnevale et al. [20] and Block Coordinated Descent (BCD) methods. This paper addresses sparsity constraints by iteratively projecting dense solutions onto a set defined by the ℓ_0 norm. This method is scalable as it avoids adding binary variables to the optimization problem. However, when processing large datasets many iterations are usually needed leading to high communication costs Fang et al. [21].

In this paper, we propose a scalable algorithm to solve the SML problems distributedly while satisfying data privacy among network nodes. Our contribution lies in developing a distributed algorithm called Bi-Linear consensus ADMM (Bi-cADMM), specifically designed for SML problems featuring ℓ_0 norm constraints. The Bi-cADMM algorithm adopts a bi-linear approach to reformulate the problem where the ℓ_0 norm constraint is exactly reformulated as a bi-linear equality constraint and three convex constraints. Additionally, Bi-cADMM employs a hierarchical decomposition strategy to split the reformulated problem into smaller sub-problems, each of which is private to each computational node. Initially, it conducts a sample decomposition of the original dataset and distributes the local datasets across computational nodes. Following this decomposition, Bi-cADMM performs a feature decomposition, of each local dataset on the GPUs available to each node. This strategy enables Bi-cADMM to execute computationally intensive data-centric computations on GPUs, while CPUs handle more cost-effective computations. The main contributions of this paper are summarized as follows:

1. We introduce the Bi-cADMM algorithm that distributedly solves SML problems induced by the ℓ_0 norm.
2. We propose a hierarchical data decomposition approach to make the Bi-cADMM algorithm adaptable across hardware platforms, from single CPU-GPU setups to large distributed networks.
3. We introduce the `PsFiT` Python package to train SML models with the Bi-cADMM optimizer.
4. We validate the algorithm's effectiveness through various numerical scenarios.

2. Distributed Data Parallel Sparse Training

The SML problem is defined over a network of N computational nodes as a minimization fitting problem incorporating a ℓ_2 regularization term [1] Tikhonov et al. [22] and an explicit sparsity constraint defined by the ℓ_0 norm. Mathematically, it can be represented as:

$$\begin{aligned} &\min_{\mathrm{x}\in\mathbb{R}^n} \sum_{i=1}^{N} \ell_i(\mathbf{A}_i\mathrm{x} - \mathbf{b}_i) + \frac{1}{2\gamma}\|\mathrm{x}\|_2^2 \\ &\text{subject to} \quad \|\mathrm{x}\|_0 \leq \kappa, \end{aligned} \tag{1}$$

[1] also known as ridge or Tikhonov regularization

where $x \in \mathbb{R}^n$ is a global decision vector, $\mathbf{A}_i \in \mathbb{R}^{m_i \times n}$ is the local feature matrix, $\mathbf{b}_i \in \mathbb{R}^{m_i}$ is the local output vector, $\gamma > 0$ is the regularization weight that controls the importance of the regularization, and $\ell_i : \mathbb{R}^n \to \mathbb{R}$ is a local convex loss function. The ℓ_2 regularization helps to reduce the effect of noise in the input data and improves the model's prediction performance. From a numerical perspective, the introduction of the ℓ_2 regularization promotes the strong convexity of the objective function, thereby improving the condition number of the objective function's Hessian matrix. By choosing different loss function ℓ_i, different sparse regression and classification models such as Sparse Linear Regression (SLinR), Sparse Logistic Regression (SLogR), Sparse Softmax Regression (SSR), and Sparse Support Vector Machines (SSVMs) can be obtained.

2.1. Bi-linear Consensus Reformulation

In this section, we present a bi-linear consensus reformulation of problem (1), enabling the development of a distributed algorithm to solve (1). Initially, we reformulate the sparsity constraint $\|\mathbf{x}\|_0 = \kappa$ as a set of norm inequality constraints and a single bi-linear equality constraint as presented in the following theorem.

Theorem 2.1 (Hempel and Goulart [23]). *For any vector* $x \in \mathbb{R}^n$*, the condition* $\|x\|_0 \leq \kappa$ *holds if, and only if, a vector* $s \in \mathbb{R}^n$ *and a scalar* $t \in \mathbb{R}$ *exist such that,*

$$x^T s = t, \quad \|x\|_1 \leq t, \quad \|s\|_1 \leq \kappa, \quad \|s\|_\infty \leq 1 \tag{2}$$

According to the theorem cited, the ℓ_0 norm constraint can be transformed into a combination of one bi-linear constraint and three linear constraints. Furthermore, we generate a local estimate of the global decision vector x, denoting it as x_i, and distribute it to each node. Subsequently, a consensus constraint is enforced on the problem, ensuring consensus among all local estimates x_i. Hence, problem (1) can be expressed equivalently as a consensus bi-linear optimization problem defined as,

$$\begin{aligned} \min_{x,z,t,s \in \mathcal{S}^\kappa} \; & \sum_{i=1}^{N} \ell_i(\mathbf{A}_i x_i - \mathbf{b}_i) + \frac{1}{2N\gamma} \|x_i\|_2^2 \\ \text{subject to } & x_i = z, \;\; \forall i = 1 \dots N \\ & g(z, s, t) = 0 \\ & \|z\|_1 \leq t. \end{aligned} \tag{3}$$

Here, z is the consensus variable, $\mathcal{S}^\kappa = \{s \in \mathbb{R}^n : \|s\|_\infty \leq 1, \|s\|_1 \leq \kappa\}$, and $g(z, s, t) = z^T s - t$. It can be readily observed that $g(z, s, t)$ is linear if either z or s is fixed.

3. Distributed Bi-Linear ADMM Algorithm

In this section, we develop the Bi-linear consensus ADMM (Bi-cADMM) algorithm to solve problem (3) in a distributed fashion. We first construct the augmented Lagrangian for problem (3), incorporating two penalty terms, one for consensus and another for the bi-linear constraint, as follows:

$$\begin{aligned} L(x, z, y, \lambda, s, t) = & \sum_{i=1}^{N} \ell_i(\mathbf{A}_i x_i - \mathbf{b}_i) + \frac{1}{2N\gamma} \|x_i\|_2^2 + y_i^T(z - x_i) + \frac{\rho_c}{2} \|z - x_i\|_2^2 \\ & + \lambda g(z, s, t) + \frac{\rho_b}{2} g(z, s, t)^2 = L_{\rho_c}(x, z, y) + L_{\rho_b}(z, \lambda, s, t). \end{aligned} \tag{4}$$

Here, y_i and λ are Lagrange multipliers, $\rho_c > 0$ and $\rho_b > 0$ are penalties, and L_{ρ_c} and L_{ρ_b} represent augmented Lagrangian functions for consensus and bi-linear constraints, defined as:

$$L_{\rho_c}(x, z, y) = \sum_{i=1}^{N} L^i_{\rho_c}(x_i, z, y_i) \qquad \text{and} \qquad L_{\rho_b}(z, \lambda, s, t) = \lambda g(z, s, t) + \frac{\rho_b}{2} g(z, s, t)^2, \tag{5}$$

where the local augmented Lagrangian function $L^i_{\rho_c}(\mathrm{x}_i, \mathrm{z}, \mathrm{y}_i)$ is expressed as:

$$L^i_{\rho_c}(\mathrm{x}_i, \mathrm{z}, \mathrm{y}_i) = \ell_i(\mathbf{A}_i\mathrm{x}_i - \mathbf{b}_i) + \frac{1}{2N\gamma}\|\mathrm{x}_i\|_2^2 + \mathrm{y}_i^T(\mathrm{z} - \mathrm{x}_i) + \frac{\rho_c}{2}\|\mathrm{z} - \mathrm{x}_i\|_2^2. \tag{6}$$

Therefore, the ADMM steps at iteration $k+1$ can be written as

$$\mathrm{x}_i^{k+1} = \arg\min_{\mathrm{x}_i} L^i_{\rho_c}(\mathrm{x}_i, \mathrm{z}^k, \mathrm{y}_i^k) \tag{7a}$$

$$(\mathrm{z}^{k+1}, t^{k+1}) = \arg\min_{\|\mathrm{z}\|_1 \le t} \sum_{i=1}^{N} L^i_{\rho_c}(\mathrm{x}_i^{k+1}, \mathrm{z}, \mathrm{y}_i^k) + L_{\rho_b}(\mathrm{z}, \lambda^k, \mathrm{s}^k, t) \tag{7b}$$

$$\mathrm{s}^{k+1} = \arg\min_{\mathrm{s}\in\mathcal{S}^\kappa} L_{\rho_b}(\mathrm{z}^{k+1}, \lambda^k, \mathrm{s}, t^{k+1}) \tag{7c}$$

$$\mathrm{y}_i^{k+1} = \mathrm{y}_i^k + \rho_c(\mathrm{x}_i^{k+1} - \mathrm{z}^{k+1}) \tag{7d}$$

$$\lambda^{k+1} = \lambda^k + \rho_b g(\mathrm{z}^{k+1}, \mathrm{s}^{k+1}, t^{k+1}). \tag{7e}$$

The minimization (7a) can be written as,

$$\mathrm{x}_i^{k+1} = \arg\min_{\mathrm{x}_i} \ell_i(\mathbf{A}_i\mathrm{x}_i - \mathbf{b}_i) + \frac{1}{2N\gamma}\|\mathrm{x}_i\|_2^2 + \frac{\rho_c}{2}\left\|\mathrm{x}_i - \mathrm{z}^k + \mathbf{u}_i^k\right\|_2^2, \tag{8}$$

where $\mathbf{u}_i^k = \frac{1}{\rho_c}\mathrm{y}_i^k$ and is updated according to

$$\mathbf{u}_i^{k+1} = \mathbf{u}_i^k + \mathrm{x}_i^{k+1} - \mathrm{z}^{k+1}. \tag{9}$$

where (7d) is used. The minimization step (7a) is reduced to the computation of the proximal operator Parikh et al. [24] of the loss function stated as,

$$\mathrm{x}_i^{k+1} = \mathrm{prox}_{\eta f_i}(\mathrm{z}^k - \mathbf{u}_i^k) \tag{10}$$

where $\mathrm{prox}_{\eta f}(\mathbf{v}) = \arg\min_{\mathrm{x}} f(\mathrm{x}) + \frac{1}{2\eta}\|\mathrm{x} - \mathbf{v}\|_2^2$ with $\eta = \frac{1}{\rho_c}$ and $f_i(\mathrm{x}_i) = \ell_i(\mathbf{A}_i\mathrm{x}_i - \mathbf{b}_i) + \frac{1}{2N\gamma}\|\mathrm{x}_i\|_2^2$. Therefore, the minimization step (10) involves solving an unconstrained and possibly high-dimensional regularized strongly convex optimization problem. To utilize parallelism, we split problem (10) across features and process each split on a GPU available in each node as discussed in the next section. The minimization steps (7b) and (7c) are convex quadratic optimization problems performed on a coordinator node. These optimization problems do not depend on the raw data and can be constructed by collecting local parameter estimates x_i^{k+1} and dual estimates y_i^k from other computational nodes. The augmented Lagrangian of the bi-linear term, $L_{\rho_b}(\mathrm{z}, \lambda, \mathrm{s}, t)$, can be written as,

$$L_{\rho_b}(\mathrm{z}, \lambda, \mathrm{s}, t) = \frac{\rho_b}{2}\left(g(\mathrm{z}, \mathrm{s}, t) + \nu\right)^2 - \frac{\rho_b}{2}\nu^2 \tag{11}$$

where $\nu = \frac{1}{\rho_b}\lambda$. Hence, the minimization step (7c) is written as,

$$\mathrm{s}^{k+1} = \arg\min_{\mathrm{s}\in\mathcal{S}^\kappa}\left(g(\mathrm{z}^{k+1}, \mathrm{s}, t^{k+1}) + \nu^k\right)^2, \tag{12}$$

where ν^k is updated by the following iteration,

$$\nu^{k+1} = \nu^k + g(\mathrm{z}^{k+1}, \mathrm{s}, t^{k+1}) \tag{13}$$

which is obtained by multiplying both sides of (7e) to $\frac{1}{\rho_b}$.

Termination. Similar to the standard distributed ADMM described by Boyd et al. [15], the convergence of iterations in (7) can be monitored using primal and dual residuals denoted by p_r^k and d_r^k, respectively. Additionally, we also consider the residual related to the bi-linear equality constraint denoted by b_r^k. These residuals are defined as

$$p_r^k = \sum_{i=1}^{N} \left\| x_i^k - z^k \right\|_2, \quad d_r^k = \sqrt{N}\rho_c \left\| z^k - z^{k-1} \right\|, \quad b_r^k = \left\| g(\mathrm{z}^k, \mathrm{s}^k, t^k) \right\|_2. \tag{14}$$

Thus, the iterations in (7) can be terminated when p_r^k, d_r^k, and b_r^k fall within a specified small tolerance.

3.1. GPU Accelerated Data-Parallel Sub-Solver

In this section, we develop an augmented Lagrangian-based algorithm to evaluate the proximal operator (10) across the GPUs available within each computing node. Our method involves breaking down the local dataset into partitions based on features and assigning each partition to a separate GPU where a portion of the model fitting problem is solved. From perspective of node i, problem (10) can be seen as:

$$\min_{\mathrm{x}_i} \ell_i(\mathbf{A}_i\mathrm{x}_i - \mathbf{b}_i) + \frac{1}{2N\gamma}\left\|\mathrm{x}_i\right\|_2^2 + \frac{\rho_c}{2}\left\|\mathrm{x}_i - \mathrm{z}^k + \mathbf{u}_i^k\right\|_2^2. \tag{15}$$

We partition the local parameter vector x_i into M blocks as $\mathrm{x}_i = [\mathrm{x}_{i1}, \ldots, \mathrm{x}_{iM}]$, with $\mathrm{x}_{ij} \in \mathbb{R}^{n_j}$, where $\sum_{j=1}^{M} n_j = n_i = n$. Hence, the local feature matrix $\mathbf{A}_i$ can be partitioned as, $\mathbf{A}_i = [\mathbf{A}_{i1}, \ldots, \mathbf{A}_{iM}]$ with $\mathbf{A}_{ij} \in \mathbb{R}^{m_i \times n_j}$. Therefore, the local subproblem (15) can be written as,

$$\min_{\mathrm{x}_i} \ \ell_i(\sum_{j=1}^{M} \mathbf{A}_{ij}\mathrm{x}_{ij} - \mathbf{b}_i) + \sum_{j=1}^{M} \frac{1}{2N\gamma}\left\|\mathrm{x}_{ij}\right\|_2^2 + \frac{\rho_c}{2}\left\|\mathrm{x}_{ij} - \mathrm{z}_j^k + \mathbf{u}_{ij}^k\right\|_2^2, \tag{16}$$

where $\mathrm{z}^k = \left[\mathrm{z}_1^k, \ldots, \mathrm{z}_M^k\right]$ and $\mathbf{u}_i^k = \left[\mathbf{u}_{i1}^k, \ldots, \mathbf{u}_{iM}^k\right]$ are also split in M blocks. By defining $r_j(\mathrm{x}_{ij})$ as

$$r_j(\mathrm{x}_{ij}) = \frac{1}{2N\gamma}\left\|\mathrm{x}_{ij}\right\|_2^2 + \frac{\rho_c}{2}\left\|\mathrm{x}_{ij} - \mathrm{z}_j^k + \mathbf{u}_{ij}^k\right\|_2^2, \tag{17}$$

problem (16) becomes

$$\begin{aligned} &\min_{\mathrm{x}_i} \ \ell_i(\sum_{j=1}^{M} \omega_j - \mathbf{b}_i) + \sum_{j=1}^{M} r_j(\mathrm{x}_{ij}) \\ &\text{subject to} \ \ \mathbf{A}_{ij}\mathrm{x}_{ij} = \omega_j, \end{aligned} \tag{18}$$

where ω_j is an auxiliary variable. The Augmented Lagrangian function for this problem now becomes:

$$L_{\rho_l} = \ell_i(\sum_{j=1}^{M} \omega_j - \mathbf{b}_i) + \sum_{j=1}^{M}\Big\{r_j(\mathrm{x}_{ij}) + \mu_j^T(\mathbf{A}_{ij}\mathrm{x}_{ij} - \omega_j) + \frac{\rho_l}{2}\left\|\mathbf{A}_{ij}\mathrm{x}_{ij} - \omega_j\right\|_2^2\Big\}. \tag{19}$$

The ADMM steps in the scaled form are written as,

$$\begin{aligned} \mathrm{x}_{ij}^{k+1} &= \arg\min_{\mathrm{x}_{ij}} r_j(\mathrm{x}_{ij}) + \frac{\rho_l}{2}\left\|\mathbf{A}_{ij}\mathrm{x}_{ij} - \omega_j^k + v_j^k\right\|_2^2 \\ \omega^{k+1} &= \arg\min_{\omega} \ \ell_i(\sum_{j=1}^{M} \omega_j - \mathbf{b}_i) + \sum_{j=1}^{M} \frac{\rho_l}{2}\left\|\mathbf{A}_{ij}\mathrm{x}_{ij}^{k+1} - \omega_j + v_j^k\right\|_2^2 \\ v_j^{k+1} &= v_j^k + A_{ij}\mathrm{x}_{ij}^{k+1} - \omega_j^{k+1}, \end{aligned} \tag{20}$$

where $v_j^k = \frac{1}{\rho_l}\mu_j^k$. Each GPU within a node is responsible for solving the first and last steps independently in parallel for each $j = 1, \ldots, M$. The ω-update requires solving a problem in Mn_j variables, however, it is shown in Boyd et al. [15] that this step can be carried out by solving a problem only in n_j variables by

Algorithm 1 Bi-cADMM Algorithm

Initialization:
Set $\mathbf{u}_i^0$, x_i^0 for $i = 1, \ldots, N$, and v^0, s^0, z^0
Main Task Scheduler:
for $k = 0, \ldots, K$ **do**
    GlobalNodeUpdate() ▷ Update consensus and sparsity variables
    Broadcast(z^{k+1}) ▷ Send global estimate to all nodes
    WorkerNodeUpdate() ▷ Parallel execution across N nodes
end for
Global Node Update:
function GlobalNodeUpdate
    Gather $\{\mathrm{x}_i^k, \mathbf{u}_i^k\}$ from all nodes $i = 1, \ldots, N$
    Solve (7b) to compute z^{k+1} and t^{k+1}
    Update s^{k+1} via (12) and dual variable v^{k+1} via (13)
end function
Worker Node Update ($i = 1, \ldots, N$):
function WorkerNodeUpdate
    $\mathbf{u}_i^{k+1} \leftarrow \mathbf{u}_i^k + \mathrm{x}_i^k - \mathrm{z}^{k+1}$ ▷ Consensus dual update (9)
    Partition $\{\mathrm{z}^{k+1}, \mathbf{u}_i^{k+1}\}$ into M blocks for GPU offloading
    $\{\mathrm{x}_{ij}^{k+1}\}_{j=1}^M \leftarrow$ GPUAcceleratorUpdate()
    Push local estimates x_i^{k+1} to Global Node
end function
GPU Accelerator Update ($j = 1, \ldots, M$):
function GPUAcceleratorUpdate
    Parallel Step: Compute local x_{ij}^{k+1} via (23) and product $\mathbf{w} = \mathbf{A}_{ij}\mathrm{x}_{ij}$
    $\overline{\mathbf{A}\mathrm{x}} \leftarrow$ IntraNodeAllReduce($\mathbf{w}$) ▷ Combine partial predictors
    Solve (21) for $\bar{\omega}$ and update local dual ν via (22)
    return x_{ij}^{k+1}
end function

optimizing over $\bar{\omega}_j = \frac{1}{M}\omega_j$, *i.e.,* the average of all local ω_j variables. The $\bar{\omega}$-minimization step is written as,

$$\bar{\omega}^{k+1} = \arg\min_{\bar{\omega}} \ell_i(M\bar{\omega} - \mathbf{b}_i) + \frac{M\rho_l}{2}\left\|\bar{\omega} - \overline{\mathbf{A}_{ij}\mathrm{x}_{ij}}^{k+1} - \nu^k\right\|_2^2, \tag{21}$$

where $\overline{\mathbf{A}_{ij}\mathrm{x}_{ij}}^{k+1} = \frac{1}{M}\sum_{j=1}^M \mathbf{A}_{ij}\mathrm{x}_{ij}^{k+1}$. Similarly, the ν_j-update becomes

$$\nu^{k+1} = \nu^k + \overline{\mathbf{A}_{ij}\mathrm{x}_{ij}}^{k+1} - \bar{\omega}^{k+1}. \tag{22}$$

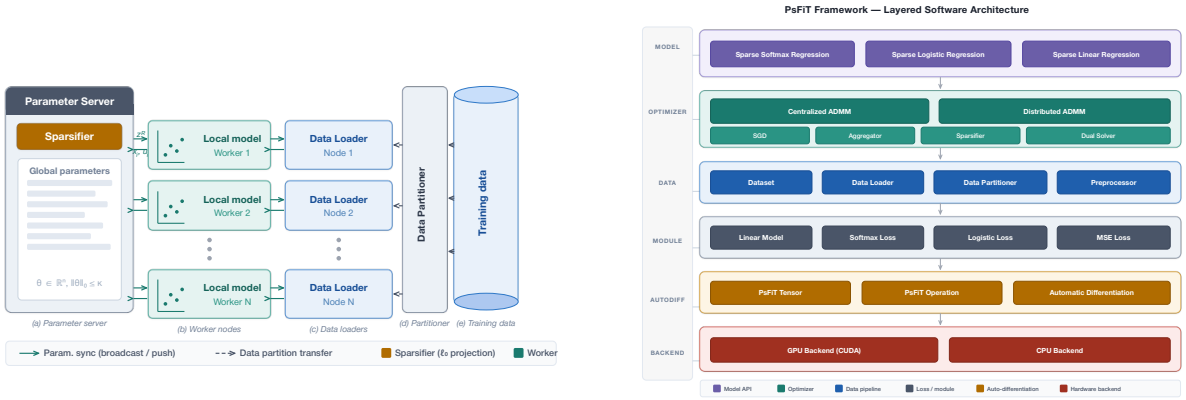
Finally, the x_{ij}-minimization step is written as

$$\mathrm{x}_{ij}^{k+1} = \arg\min_{\mathrm{x}_{ij}} r_j(\mathrm{x}_{ij}) + \frac{\rho_l}{2}\left\|\mathbf{A}_{ij}\mathrm{x}_{ij} - \mathbf{A}_{ij}\mathrm{x}_{ij}^k - \bar{\omega}^k + \overline{\mathbf{A}_{ij}\mathrm{x}_{ij}}^k + \nu^k\right\|_2^2. \tag{23}$$

The x_{ij}-minimization step involves solving M parallel regularized least-squares problems, each with n_j variables. Following this, we aggregate and sum the partial predictors $\mathbf{A}_{ij}\mathrm{x}_{ij}^{k+1}$ between the first and second steps to construct $\overline{\mathbf{A}_{ij}\mathrm{x}_{ij}}^{k+1}$. The second stage involves a single minimization problem in m_i variables, focusing on quadratically regularized loss minimization. Subsequently, the third stage comprises a straightforward update in m_i variables. Since the loss function ℓ_i is separable, the $\bar{\omega}$-update splits entirely into m_i scalar optimization problems.

3.2. Distributed Implementation and Bi-cADMM Pseudo-Code

This section describes the algorithm's pseudo-code corresponding to the main steps discussed earlier. We categorize the algorithm into three primary updates: *global-level*, *node-level*, and *device-level*. The global-level update encompasses steps executed on the CPU of the global node with network-level communications. These computations are independent of problem-specific data such as $\mathbf{A}_i$ and $\mathbf{b}_i$. The



(a) (b)

Figure 1: Architectural overview of the PsFiT framework. Panel (a) illustrates the distributed parameter-server system used for large-scale data processing, while panel (b) details the internal modular software stack, ranging from the high-level model API down to the hardware abstraction backend.

node-level computations are primarily data-driven and are carried out on the GPUs available to each node with possible inter-GPU communications. Consequently, data partitions like $\mathbf{A}_{ij}$ reside on the j-th GPU of the i-th node during algorithm execution. Lastly, the device-level computations involve solving the sub-problem (23) on each GPU (or CPU in case GPUs are not available) within a node, primarily focusing on thread-level parallel computations. The pseudo-code of Bi-cADMM algorithm is presented in Algorithm 1.

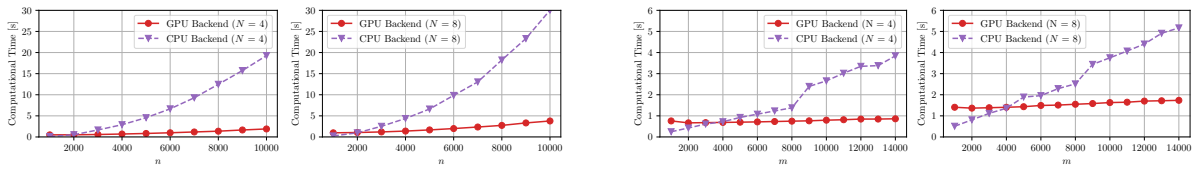
In the node-level algorithm, each GPU unit gathers z_j^k and $\mathbf{u}_{ij}^k$ from the corresponding node. Then, it computes x_{ij} using an equation (23). Following this, it computes $\mathbf{A}_{ij}x_{ij}$, denoted as $\mathbf{w}$, representing a matrix-vector product, independently across all GPUs in parallel. Subsequently, the computed $\mathbf{w}$ values are aggregated across all GPUs using an `AllReduce` operation to compute $\overline{\mathbf{A}_{ij}x_{ij}}$.

4. PsFiT System Architecture

This section provides an overview of the technical aspects of PsFiT, including its software architecture, core components, and implementation details. PsFiT is developed using `Python 3.11` and leverages the `Ray` framework [25] to enable distributed computing and implement the parameter-server model [26]. The package is open-sourced and available on https://github.com/Alirezalm/psfit.

4.1. Numerical Hierarchy

PsFiT adopts a modular, layered architecture to facilitate scalability, flexibility, and ease of maintenance. The architecture, as illustrated in Figure 1b, is organized into several layers, each handling specific tasks and interacting through well-defined APIs. At the top, the *Model Layer* supports predefined sparse models, including softmax regression, logistic regression, and linear regression. This layer provides a *high-level* API, allowing users to configure and train these models with minimal effort. For advanced use cases, users must interact with the APIs provided by other layers and build their sparse models using either PsFiT's built-in functionality or by extending classes across different layers. The *Optimizer Layer* implements Algorithm 1 in both centralized and distributed versions, alongside other components like SGD, Aggregator, Sparsifier, and Dual Solver which are used to create an ADMM-based algorithm. The *Data Layer* handles data input and preprocessing, including dataset loading, partitioning for distributed computing, and initial data transformations. This allows seamless integration with a variety of datasets while ensuring they are ready for model training. The *Module Layer* comprises the core machine learning operations, including modules for linear models, softmax, logistic, and MSE loss functions. These modules define the objective functions needed for sparse optimization. The



(a) (b)

Figure 2: Scalability analysis of the Bi-cADMM algorithm. The plots compare the computational execution times of CPU and GPU backends across different node cluster sizes ($N = 4$ and $N = 8$) as (a) the number of model features increases and (b) the number of data samples scales.

Autodiff Layer provides automatic differentiation via custom tensors and operations, which is crucial for gradient-based optimization. The *Backend Layer* abstracts hardware details and supports both GPU and CPU backends.

4.2. Parameter-Server Architecture

The `PsFiT` framework employs a parameter-server architecture for distributed ℓ_0 sparse model training (see Figure 1a). This architecture comprises a central *Parameter Server* and several worker nodes. The parameter server manages the global model parameters and sparsification, which is achieved through the *sparsifier* component. Each worker node consists of a *Data Loader* that retrieves partitioned data for training and sends the relevant local updates to the parameter server. The *Data Partitioner* module is responsible for splitting the training dataset across nodes, allowing the system to handle large-scale data efficiently. During training, the parameter server synchronizes the global model parameters across nodes, while the worker nodes update their local models using the data they load.

5. Experimental Results

In this section, we empirically evaluate `PsFiT` using synthetic data (to assess scalability for sparse linear regression) and real-world benchmarks: MNIST (sparse softmax regression for image classification) and IMDB (sparse logistic regression for sentiment analysis). Table 1 summarizes the dataset statistics.

Table 1
Dataset statistics for MNIST and IMDB used in the empirical evaluation.

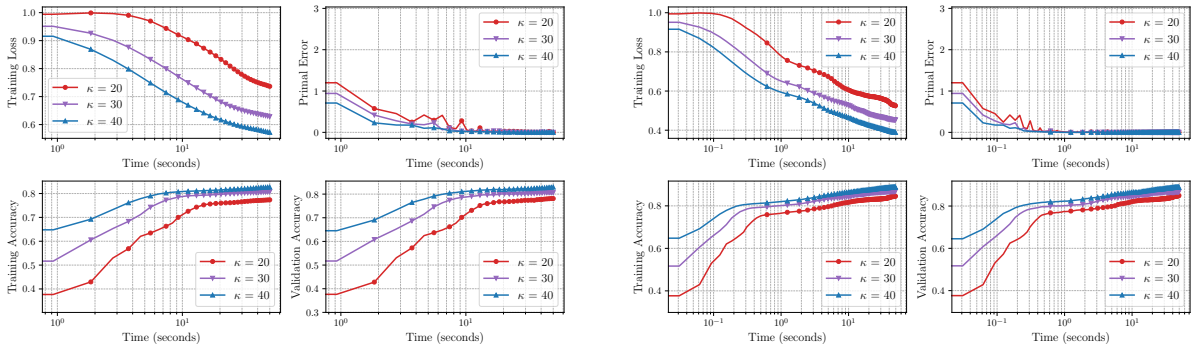
Dataset	Train	Valid.	Test	Dim.	N	m_i
MNIST	60k	10k	10k	784	10	6k
IMDB	20k	5k	25k	10k	5	4k

5.1. Experimental Setup

All the experiments were performed on a machine running Ubuntu 24.04 LTS equipped with an Intel Core i9-14900K processor clocked at 3.20 GHz and 64 GB of DDR5 memory. The GPU backend uses dual NVIDIA GeForce RTX 5070 GPUs, each featuring 12 GB of GDDR7 memory, connected through PCIe 5.0. To evaluate the distributed nature of the algorithms without a physical cluster, multi-node training environments are simulated locally utilizing the `ray` framework. The proposed algorithm is implemented within the `PsFiT` package, which is entirely written in Python 3.11.

5.2. Synthetic Datasets

Local feature matrices $\mathbf{A}_i$ are generated from a standard normal distribution with ℓ_2-normalized columns. A ground truth vector $x_{\text{true}} \in \mathbb{R}^n$ is generated with a specified sparsity level, and local labels are computed



(a) (b)

Figure 3: Sparse softmax regression performance on the MNIST dataset evaluated over a fixed 50-second training period. Convergence metrics, including training loss, primal error, and accuracy, are compared between (a) CPU and (b) GPU backends across varying sparsity levels (κ).

as $\mathbf{b}_i = \mathbf{A}_i \mathrm{x}_{\text{true}} + \mathbf{e}$, with noise $\mathbf{e} \sim \mathcal{N}(0, \Sigma)$. To evaluate the feature scalability of Algorithm 1, we fix the node-level sample size to 800 ($\mathbf{A}_i \in \mathbb{R}^{800 \times n}$) and vary the number of features n from 1,000 to 10,000. We test configurations with $N \in \{4, 8\}$ nodes, yielding total dataset sizes of 3,200 and 6,400 points, respectively. For this scenario, the ground truth sparsity is fixed at 80% (0.8).

Scalability across features Figure 2a presents the performance comparison of Algorithm 1 using CPU and GPU backends as the number of features increases. The line graphs depict computational times for feature scaling on CPUs and GPUs across different numbers of nodes. It is evident that GPUs consistently outperform CPUs, maintaining lower computational times even as the number of features rises to 10000.

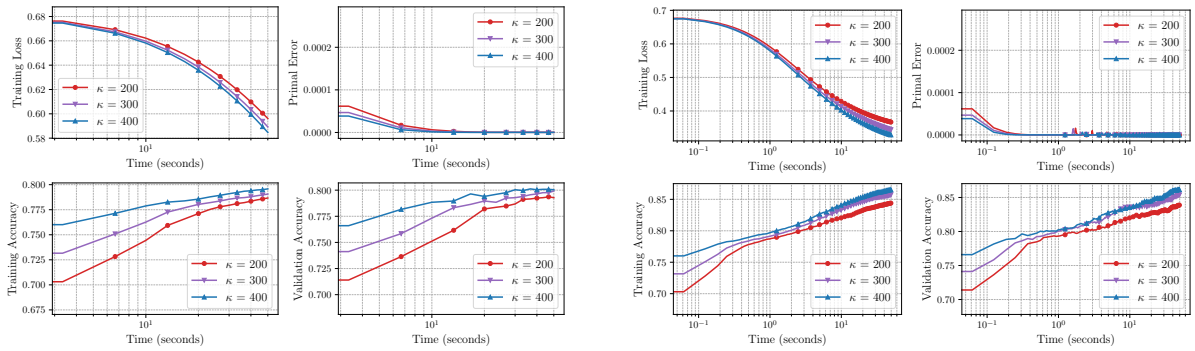
Scalability across datapoints Figure 2b illustrates the sample scalability of Algorithm 1. Fixing features at $n = 4000$, we scale the local data points per node from 25×10^3 to 300×10^3 (totaling up to 1.2×10^6 and 2.4×10^6 samples for $N = 4$ and $N = 8$, respectively). Consistent with the feature scaling trends, the GPU backend handles large datasets efficiently, exhibiting only a gradual rise in computational time compared to the steep increase observed on the CPU.

5.3. Real-world datasets

In this section, we present the evaluation of Algorithm 1 for training distributed sparse softmax and logistic regression models, focusing on image classification and sentiment analysis tasks. To ensure consistency in benchmarking, the maximum training time is fixed at 50 seconds across all configurations. Moreover, we randomly partitioned the datasets into $N = 16$ splits to simulate a distributed training environment.

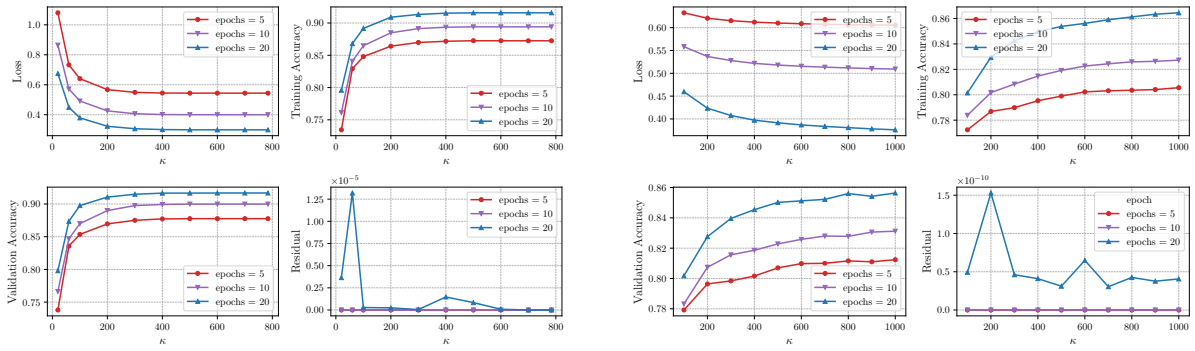
5.3.1. Image Classification with Sparse Softmax Regression

Figures 3a and 3b present the sparse softmax regression results for sparsity levels $\kappa \in \{20, 30, 40\}$. On both backends, relaxing the sparsity constraint (increasing κ) consistently improves training loss, primal error convergence, and overall accuracy. Notably, the GPU backend achieves significantly faster convergence across all κ values, yielding lower loss and higher accuracy than the CPU within the fixed 50-second training window.



(a) (b)

Figure 4: Sparse logistic regression performance on the IMDB dataset evaluated over a fixed 50-second training window. The panels contrast the convergence and efficiency of (a) CPU versus (b) GPU implementations at different sparsity levels (κ).



(a) (b)

Figure 5: Impact of network density and local SGD epochs on model performance. The charts depict training/-validation accuracy, loss, and residuals for (a) sparse softmax regression applied to MNIST and (b) sparse logistic regression applied to IMDB.

5.3.2. Sentiment Analysis with Sparse Logistic Regression

For sentiment analysis on the IMDB dataset, sparse logistic regression results ($\kappa \in \{200, 300, 400\}$) in Figures 4a and 4b mirror the image classification trends, with the GPU backend consistently outperforming the CPU. Furthermore, Figures 5a and 5b analyze the impact of network density and local SGD epochs. Across both datasets, higher densities and increased local epochs consistently improve accuracy and reduce residuals. However, highly competitive performance (e.g., $\sim$ 0.9 accuracy on MNIST) is achieved even at lower densities and fewer epochs (e.g., 20 epochs), demonstrating the robustness of the sparse models without requiring dense feature sets or excessive local training.

6. Conclusions

In this paper, we introduced the Bi-cADMM algorithm for solving regularized SML problems over a network of computational nodes. Bi-cADMM leverages a bi-linear consensus reformulation of the original SML problem and hierarchical decomposition to enable parallel computing on both CPUs and GPUs. Implemented within an open-source Python package, our algorithm demonstrates efficiency and scalability through numerical benchmarks across various SML problems with distributed datasets. However, the main limitations lie in the necessity of a global node and the lack of convergence proof. Future work will address these issues by focusing on convergence analysis and optimizing Bi-cADMM for efficient use in multi-GPU and multi-CPU environments.

7. Acknowledgment

This research was funded by Högskolestiftelsen i Österbotten. This work was also conducted within the Data Analytics for Zero Emission Marine (DAZE) project, with financial support from Business Finland (ref. 6525/31/2022). The authors gratefully acknowledge additional support from Digital Futures and the C3.ai Digital Transformation Institute.

Declaration on Generative AI

During the preparation of this work, the authors used ChatGPT solely for grammar and language checking.

References

[1] J. Verbraeken, M. Wolting, J. Katzy, J. Kloppenburg, T. Verbelen, J. S. Rellermeyer, A survey on distributed machine learning, ACM Computing Surveys 53 (2020) 1–33. URL: https://doi.org/10.1145/3377454.

[2] D. Bertsimas, J. Pauphilet, B. Van Parys, Sparse classification: a scalable discrete optimization perspective, Machine Learning 110 (2021) 3177–3209. URL: https://doi.org/10.1007/s10994-021-06085-5.

[3] D. Bertsimas, J. Dunn, L. Kapelevich, R. Zhang, Sparse regression over clusters: SparClur, Optimization Letters 16 (2022) 433–448. URL: https://doi.org/10.1007/s11590-021-01770-9.

[4] A. Olama, E. Camponogara, J. Kronqvist, Sparse convex optimization toolkit: a mixed-integer framework, Optimization Methods and Software 38 (2023) 1269–1295. URL: https://doi.org/10.1080/10556788.2023.2222429.

[5] A. Olama, E. Camponogara, P. R. Mendes, Distributed primal outer approximation algorithm for sparse convex programming with separable structures, Journal of Global Optimization 86 (2023) 637–670. URL: https://doi.org/10.1007/s10898-022-01266-5.

[6] D. Bertsimas, A. King, R. Mazumder, Best subset selection via a modern optimization lens, The Annals of Statistics 44 (2016) 813–852. URL: https://doi.org/10.1214/15-AOS1388.

[7] D. Bertsimas, N. Mundru, Sparse convex regression, INFORMS Journal on Computing 33 (2021) 262–279. URL: https://doi.org/10.1287/ijoc.2020.0954.

[8] D. Bertsimas, B. Van Parys, Sparse high-dimensional regression: Exact scalable algorithms and phase transitions, The Annals of Statistics 48 (2020) 300–323. URL: https://doi.org/10.1214/18-AOS1804.

[9] D. Bertsimas, A. King, Logistic regression: From art to science, Statistical Science 32 (2017) 367–384. URL: https://www.jstor.org/stable/26408297.

[10] Y. Bai, R. Liang, Z. Yang, Splitting augmented Lagrangian method for optimization problems with a cardinality constraint and semicontinuous variables, Optimization Methods and Software 31 (2016) 1089–1109. URL: https://doi.org/10.1080/10556788.2016.1196206.

[11] J. Liu, J. Huang, Y. Zhou, X. Li, S. Ji, H. Xiong, D. Dou, From distributed machine learning to federated learning: A survey, Knowledge and Information Systems 64 (2022) 885–917. URL: https://doi.org/10.1007/s10115-022-01664-x.

[12] B. McMahan, E. Moore, D. Ramage, S. Hampson, B. A. y Arcas, Communication-efficient learning of deep networks from decentralized data, in: Artificial intelligence and statistics, PMLR, 2017, pp. 1273–1282.

[13] A. M. Tillmann, D. Bienstock, A. Lodi, A. Schwartz, Cardinality minimization, constraints, and regularization: a survey, arXiv preprint arXiv:2106.09606 (2021).

[14] X. Sun, X. Zheng, D. Li, Recent advances in mathematical programming with semi-continuous variables and cardinality constraint, Journal of the Operations Research Society of China 1 (2013) 55–77. URL: https://doi.org/10.1007/s40305-013-0004-0.

[15] S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein, et al., Distributed optimization and statistical learning via the alternating direction method of multipliers, Foundations and Trends® in Machine learning 3 (2011) 1–122. URL: http://dx.doi.org/10.1561/2200000016.

[16] Q. Tong, G. Liang, J. Ding, T. Zhu, M. Pan, J. Bi, Federated optimization of ℓ_0-norm regularized sparse learning, Algorithms 15 (2022) 319. URL: https://doi.org/10.3390/a15090319.

[17] M. A. Duran, I. E. Grossmann, An outer-approximation algorithm for a class of mixed-integer nonlinear programs, Mathematical Programming 36 (1986) 307–339. URL: https://doi.org/10.1007/BF02592064.

[18] A. Olama, N. Bastianello, P. R. Mendes, E. Camponogara, Relaxed hybrid consensus ADMM for distributed convex optimisation with coupling constraints, IET Control Theory & Applications 13 (2019) 2828–2837. URL: https://doi.org/10.1049/iet-cta.2018.6260.

[19] A. Olama, G. Carnevale, G. Notarstefano, E. Camponogara, A tracking augmented lagrangian method for ℓ_0 sparse consensus optimization, in: 9th International Conference on Control, Decision and Information Technologies (CoDIT), 2023, pp. 01–06. URL: https://ieeexplore.ieee.org/document/10284260.

[20] G. Carnevale, I. Notarnicola, L. Marconi, G. Notarstefano, Triggered gradient tracking for asynchronous distributed optimization, Automatica 147 (2023) 110726. URL: https://doi.org/10.1016/j.automatica.2022.110726.

[21] C.-H. Fang, S. B. Kylasa, F. Roosta, M. W. Mahoney, A. Grama, Newton-ADMM: A distributed gpu-accelerated optimizer for multiclass classification problems, in: SC20: International Conference for High Performance Computing, Networking, Storage and Analysis, IEEE, 2020, pp. 1–12. URL: https://ieeexplore.ieee.org/document/9355255.

[22] A. N. Tikhonov, et al., On the stability of inverse problems, in: Dokl. Akad. Nauk SSSR, volume 39, 1943, pp. 195–198.

[23] A. B. Hempel, P. J. Goulart, A novel method for modelling cardinality and rank constraints, in: 53rd IEEE Conference on Decision and Control, IEEE, 2014, pp. 4322–4327. URL: https://ieeexplore.ieee.org/document/7040063.

[24] N. Parikh, S. Boyd, et al., Proximal algorithms, Foundations and trends® in Optimization 1 (2014) 127–239.

[25] P. Moritz, et al., Ray: A distributed framework for emerging AI applications, in: 13th USENIX symposium on operating systems design and implementation (OSDI 18), 2018, pp. 561–577.

[26] M. Li, D. G. Andersen, A. Smola, K. Yu, Communication efficient distributed machine learning with the parameter server, Advances in neural information processing systems 27 (2014).