Asynchronous Multi-stream Parallel Stochastic Gradient Descent for High-Dimensional Incomplete Data Latent Feature Analysis Supplementary File

Anonymous

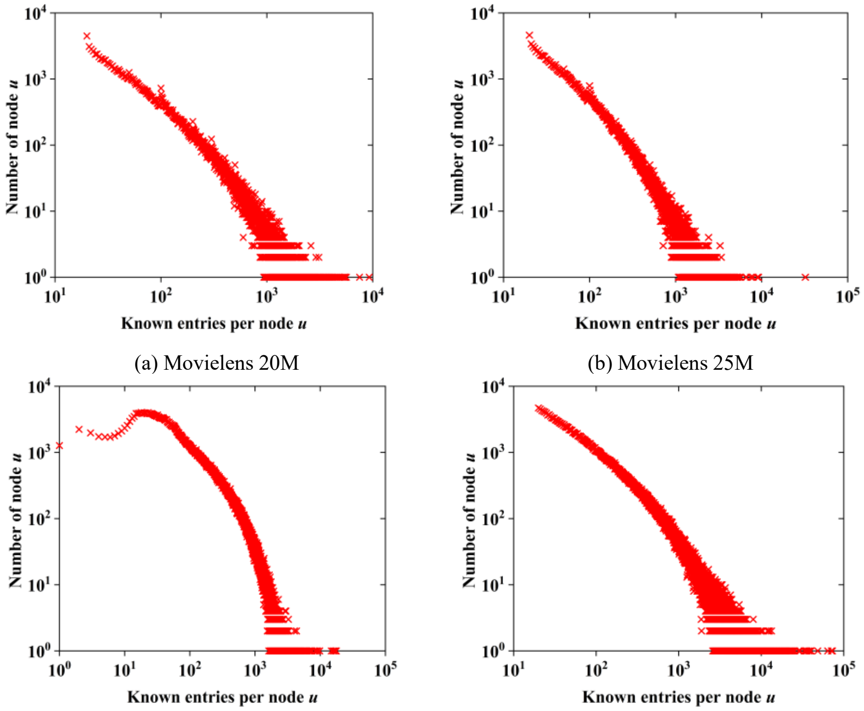
I. INTRODUCTION

This is the supplementary file for paper entitled "*Asynchronous Multi-stream Parallel Stochastic Gradient Descent for High-Dimensional Incomplete Data Latent Feature Analysis*". It provides the BAP algorithm's detailed design, experimental results, and the convergence proof of the AMP-SGD-based LFA model.

II. EXISTING PARALLEL SGDS AND THEIR LIMITATIONS

In this section, we analyze the challenges of existing parallel SGD algorithms for LFA model.

A. Load Imbalance



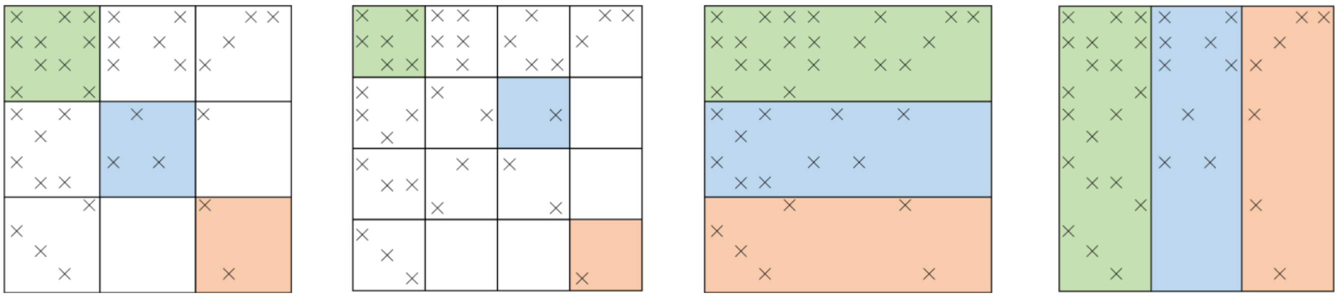
(a) Movielens 20M

(b) Movielens 25M

(c) Netflix

(d) YahooMusic

Fig. S1. The known entry distribution of four industrial HDI matrices.



(a) DSGD

(b) FPSGD

(c) ASGD

Fig. S2. Three algorithms' node-wise partitioning method at three threads.

HDI matrices are usually large-scale due to the huge number of nodes in industrial scenarios. Thus, constructing an LFA model via SGD is time-consuming. To speed up training, some parallel SGD algorithms have been proposed for LFA [6], [8]-[13]. The key to parallelizing SGD is to overcome the update overwriting problem, i.e., to avoid multiple threads selecting entries that share the same row or column indices, as this can cause update conflicts in $\mathbf{p}_u$ or $\mathbf{q}_i$. Hogwild! [10] directly uses multiple threads to asynchronously update LF matrices in parallel, tolerating the overwriting problem to some extent. However, it assumes that an HDI matrix is highly sparse to ensure model convergence. Instead, the mainstream algorithms partition an HDI matrix into multiple blocks and update the parameters (i.e., P and Q) of each block in parallel via batch synchronization or dynamic scheduling. Consider two entries $r_{u1,i1}$ and $r_{u2,i2}$, which can be updated independently without creating update conflicts, if they satisfy:

$$u1 \neq u2 \,\&\&\, i1 \neq i2 \tag{S1}$$

Distributed SGD (DSGD) [6] extent this concept to block granularity. It partitions an HDI matrix R into S×S blocks, where S is the number of threads. These blocks are further grouped into S stratums, each containing S independently updatable blocks. In a stratum, S threads update S blocks in parallel. An explicit batch synchronization of the previous stratum is necessary upon switching to the next stratum. Fast and parallel SGD (FPSGD) [11] eliminates batch synchronization by partitioning an HDI matrix into (S+1)×(S+1) blocks. It introduces a concept of free blocks, i.e., parallelizable blocks independent of the block currently being updated. Then, asynchronous parallel updating is realized by scheduling a free block with the fewest number of updates to the threads. Alternating SGD (ASGD) [8] partitions an HDI matrix into S row blocks and S column blocks for parameter parallelism, respectively. These two partitions correspond to two heterogeneous subtasks. In each subtask, S row or column blocks are updated by S threads in parallel. Subtask switching requires batch synchronization like DSGD. These algorithms are evenly partitioned in a node-wise manner, i.e., the node sets *U* and *I* are partitioned into equal-length subsets.

However, many existing industrial HDI matrices are characterized by long-tailed distributions [16], [17], i.e., a small portion of nodes in an HDI matrix have frequent interactions while most nodes have low interaction frequencies. For example, in recommender systems, a few popular items feature a large amount of interaction data (e.g., clicks, purchases, ratings, etc.), while the vast majority of cold items have only a few or even no interaction records. Fig. S1 shows the known entry distribution of four industrial HDI matrices.

The node-wise partitioning does not consider such the data skewness. Fig. S2 illustrates how the three SGD algorithms partition a HDI matrix at three threads. These algorithms suffer from different problems when the matrix is skewed. Both DSGD and ASGD suffer from the curse of the last reducer [18], i.e., in a batch synchronization phase, all threads must wait for the slowest one to finish its updates before switching to the next block. This causes thread underutilization. Moreover, in FPSGD, dense blocks update less frequently than sparse blocks, and this update bias leads to slow convergence of the model [19]. Although some methods use shuffle [6], [13] to alleviate this issue, its I/O and time complexity is high, and cannot effectively improve the balanced distribution of entries across blocks [9].

B. Lock Contention

```
Procedure S1: Single-Stream Scheduling Scheme
Class Scheduler:
    Lock lock;
    // Scheduling a free block
    Block Function getJob():
        acquire the lock;
        scheduling a free block with the fewest number of updates;
        mark the corresponding row and column block indices as 'non-free';
        return the free block
    // Pushing back a block
    Function pubJob(Block block):
        acquire the lock;
        add 1 to the number of updates for the block;
        push back the block to the pending update queue;
        mark the corresponding row and column block indices as 'free';
```

For parallel SGD algorithms based on dynamic scheduling, all threads need to issues a scheduling request to a global scheduler to obtain a free block with the fewest number of updates [11], [13]. However, there is an implicit lock inside the scheduler, each thread must acquire this lock before it can get a free block. If multiple threads issue scheduling requests to the scheduler simultaneously, it inevitably leads to a lock contention problem (one thread holds this lock and the rest of the threads are blocked, resulting in low thread utilization). We term such scheduling methods as single-stream scheduling schemes, Procedure S1 describes the scheme class.

In Procedure S1, the scheme has functions getJob() and putJob(). getJob() is used to schedule a free block with the fewest number of updates to a thread, and pubJob() is used to push that free block back to the pending update queue. Both functions share a data structure that maintains the current unprocessed blocks and their update counts. To prevent threads from writing to this data structure simultaneously, other threads must be blocked when either thread invokes getJob() or putJob(). More threads, the more prominent the lock contention problems, which limits the scalability of dynamic scheduling-based SGD. To address this problem, FPSGD++ [12] eliminates the lock in its scheme and allows multiple threads to randomly pick a free block simultaneously. However, it abandons the principle of selecting the block with the fewest number of updates, affecting its convergence.

III. Supplementary Algorithm

Algorithm S1: Balance-Aware Partitioning (BAP)

```
Input: Known entry set $\Lambda$, node sets $U$, $I$, and M, N
Output: Blocks R = $\{R^{mn}|1\leq m\leq M, 1\leq n\leq N\}$
rowBlockMap = {};
colBlockMap = {};
// Row-wise partitioning
$m$=1, $\Lambda(U^m)$=0;
for $u = 1, 2, \ldots, |U|$ do
    $\Lambda(U^m_{tmp}) = \Lambda(U^m) + nnz(u)$;
    if $||\Lambda|/M-\Lambda(U^m)|_{abs}<||\Lambda|/M-\Lambda(U^m_{tmp})|_{abs}$ then
        rowBlockMap[$m$] = $u$;
        $\Lambda(U^m) = nnz(u)$;
        $m$ += 1;
    else
        $\Lambda(U^m) = \Lambda(U^m_{tmp})$;
    end if
    if $m$ == M then
        rowBlockMap[$m$] = $|U|$;
    end if
end for
// Column-wise partitioning
$n$=1, $<I^n> = 0$;
for $i = 1, 2, \ldots, |I|$ do
    $\Lambda(I^n_{tmp}) =\Lambda(I^n) + nnz(i)$;
    if $||\Lambda|/N-\Lambda(I^n)|_{abs}<||\Lambda|/N-\Lambda(I^n_{tmp})|_{abs}$ then
        colBlockMap[$n$] = $i$;
        $\Lambda(I^n) = nnz(i)$;
        $n$ += 1;
    else
        $\Lambda(I^n) = \Lambda(I^n_{tmp})$;
    end if
    if $n$ == N then
        colBlockMap[$n$] = $|I|$;
// Block partitioning
for $r_{ui}$ in $\Lambda$ do
    for pair in rowBlockMap do
        if $u$ <= pair.second then
            $m$ = pair.first;
            break;
        end if
    end for
    for pair in colBlockMap do
        if $i$ <= pair.second then
            $n$ = pair.first;
            break;
        end if
    end for
    Add $r_{ui}$ to $R^{mn}$
return Blocks R = $\{R^{mn}|1\leq i\leq M, 1\leq j\leq N\}$;
```

*Note that $|\cdot|_{abs}$ denotes the absolute value of a number.

IV. Proofs for Lemmas 1, 2, and Theorem 1

A. Proof for Lemma 1

Given two arbitrary vectors $\mathbf{p}_{u1}$, $\mathbf{p}_{u2}\in\mathbb{R}^d$ and perform the second-order Taylor expansion of $\varepsilon_{ui}(\mathbf{p}_{u1})$ at $\mathbf{p}_{u2}$, we get

$$\begin{aligned}&\varepsilon_{ui}(\mathbf{p}_{u1})=\varepsilon_{ui}(\mathbf{p}_{u2})+\nabla\varepsilon_{ui}(\mathbf{p}_{u2})(\mathbf{p}_{u1}-\mathbf{p}_{u2})^T+\frac{1}{2}(\mathbf{p}_{u1}-\mathbf{p}_{u2})\nabla^2\varepsilon_{ui}(\mathbf{p}_{u2})(\mathbf{p}_{u1}-\mathbf{p}_{u2})^T\\ &\Rightarrow\varepsilon_{ui}(\mathbf{p}_{u1})-\varepsilon_{ui}(\mathbf{p}_{u2})=\nabla\varepsilon_{ui}(\mathbf{p}_{u2})(\mathbf{p}_{u1}-\mathbf{p}_{u2})^T+\frac{1}{2}(\mathbf{p}_{u1}-\mathbf{p}_{u2})\nabla^2\varepsilon_{ui}(\mathbf{p}_{u2})(\mathbf{p}_{u1}-\mathbf{p}_{u2})^T.\end{aligned}\tag{S2}$$

According to **Definition 3**, the following inequality holds if ε_{ui} is strongly convex:

$$\varepsilon_{ui}(\mathbf{p}_{u1})-\varepsilon_{ui}(\mathbf{p}_{u2})\geq\nabla\varepsilon_{ui}(\mathbf{p}_{u2})(\mathbf{p}_{u1}-\mathbf{p}_{u2})^T+\frac{\mu}{2}\|\mathbf{p}_{u1}-\mathbf{p}_{u2}\|_2^2.\tag{S3}$$

Combining (S2) and (S3), we get

$$(\mathbf{p}_{u1}-\mathbf{p}_{u2})\nabla^2\varepsilon_{ui}(\mathbf{p}_{u2})(\mathbf{p}_{u1}-\mathbf{p}_{u2})^T\geq\mu\|\mathbf{p}_{u1}-\mathbf{p}_{u2}\|_2^2.\tag{S4}$$

Since $\nabla^2\varepsilon_{ui}(\mathbf{p}_{u2})=\mathbf{q}_i^T\mathbf{q}_i+\lambda\mathrm{I}_d$, we get

$$\begin{aligned}&(\mathbf{p}_{u1}-\mathbf{p}_{u2})(\mathbf{q}_i^T\mathbf{q}_i+\lambda I_d)(\mathbf{p}_{u1}-\mathbf{p}_{u2})^T\geq\mu\|\mathbf{p}_{u1}-\mathbf{p}_{u2}\|_2^2\\ &\Rightarrow(\mathbf{p}_{u1}-\mathbf{p}_{u2})(\mathbf{q}_i^T\mathbf{q}_i+\lambda I_d-\mu I_d)(\mathbf{p}_{u1}-\mathbf{p}_{u2})^T\geq 0.\end{aligned}\tag{S5}$$

If matrix $(\mathbf{q}_i^T\mathbf{q}_i+\lambda\mathrm{I}_d\text{-}\mu\mathrm{I}_d)$ is positive semi-definite, then (S5) holds. Further, the matrix is positive semi-definite if μ is the minimum singular value of matrix $(\mathbf{q}_i^T\mathbf{q}_i+\lambda\mathrm{I}_d)$. Hence, **Lemma 1** stands. □

B. Proof for Lemma 2

Given vectors $\mathbf{p}_{u1}$, $\mathbf{p}_{u2}$, we get

$$\begin{aligned}&\nabla\varepsilon_{ui}(\mathbf{p}_{u1})-\nabla\varepsilon_{ui}(\mathbf{p}_{u2})=(\mathbf{p}_{u1}-\mathbf{p}_{u2})(\mathbf{q}_i^T\mathbf{q}_i+\lambda I_d)\\ &\Rightarrow\|\nabla\varepsilon_{ui}(\mathbf{p}_{u1})-\nabla\varepsilon_{ui}(\mathbf{p}_{u2})\|_2^2=\|(\mathbf{p}_{u1}-\mathbf{p}_{u2})(\mathbf{q}_i^T\mathbf{q}_i+\lambda I_d)\|_2^2\leq\|(\mathbf{p}_{u1}-\mathbf{p}_{u2})\|_2^2\|(\mathbf{q}_i^T\mathbf{q}_i+\lambda I_d)\|_2^2,\end{aligned}\tag{S6}$$

where $\|\mathbf{q}_i^T\mathbf{q}_i+\lambda\mathrm{I}_d\|_2^2$ is the maximum singular value of $(\mathbf{q}_i^T\mathbf{q}_i+\lambda\mathrm{I}_d)$. Let $L=\|\mathbf{q}_i^T\mathbf{q}+\lambda\mathrm{I}_d\|^2$, **Lemma 2** stands. □

C. Proof for Theorem 1

Based on the learning schemes (4), we get

$$\begin{aligned}\|\mathbf{p}_u^{(k)}-\mathbf{p}_u^*\|_2^2&=\|\mathbf{p}_u^{(k-1)}-\eta\nabla\varepsilon_{ui}(\mathbf{p}_u^{(k-1)})-\mathbf{p}_u^*\|_2^2\\ &=\|\mathbf{p}_u^{(k-1)}-\mathbf{p}_u^*\|_2^2-2\eta\nabla\varepsilon_{ui}(\mathbf{p}_u^{(k-1)})(\mathbf{p}_u^{(k-1)}-\mathbf{p}_u^*)^T+\eta^2\|\nabla\varepsilon_{ui}(\mathbf{p}_u^{(k-1)})\|_2^2.\end{aligned}\tag{S7}$$

Taking the expectation on both sides of (S7), we get

$$\mathbb{E}\left[\|\mathbf{p}_u^{(k)}-\mathbf{p}_u^*\|_2^2\right]=\mathbb{E}\left[\|\mathbf{p}_u^{(k-1)}-\mathbf{p}_u^*\|_2^2\right]-2\eta\mathbb{E}\left[\nabla\varepsilon_{ui}(\mathbf{p}_u^{(k-1)})(\mathbf{p}_u^{(k-1)}-\mathbf{p}_u^*)^T\right]+\eta^2\mathbb{E}\left[\|\nabla\varepsilon_{ui}(\mathbf{p}_u^{(k-1)})\|_2^2\right].\tag{S8}$$

Based on **Definition 3** and **Lemma 1**, we get

$$\begin{cases}\varepsilon_{ui}(\mathbf{p}_u^*)\geq\varepsilon_{ui}(\mathbf{p}_u^{k-1})+\nabla\varepsilon_{ui}(\mathbf{p}_u^{k-1})(\mathbf{p}_u^*-\mathbf{p}_u^{k-1})^T+\dfrac{\mu}{2}\|\mathbf{p}_u^{k-1}-\mathbf{p}_u^*\|_2^2;\\ \varepsilon_{ui}(\mathbf{p}_u^{k-1})\geq\varepsilon_{ui}(\mathbf{p}_u^*)+\nabla\varepsilon_{ui}(\mathbf{p}_u^*)(\mathbf{p}_u^{k-1}-\mathbf{p}_u^*)^T+\dfrac{\mu}{2}\|\mathbf{p}_u^{k-1}-\mathbf{p}_u^*\|_2^2.\end{cases}\tag{S9}$$

Combining the above inequalities yields

$$\left[\nabla\varepsilon_{ui}(\mathbf{p}_u^{k-1})-\nabla\varepsilon_{ui}(\mathbf{p}_u^*)\right](\mathbf{p}_u^{k-1}-\mathbf{p}_u^*)^T\geq\mu\|\mathbf{p}_u^{k-1}-\mathbf{p}_u^*\|_2^2.\tag{S10}$$

Since $\mathbf{p}_u^*$ is the optimal state of $\mathbf{p}_u$, we get

$$\varepsilon_{ui}(\mathbf{p}_u^*)=0.\tag{S11}$$

Hence, (S10) is reformulated as

$$\nabla\varepsilon_{ui}(\mathbf{p}_u^{k-1})(\mathbf{p}_u^{k-1}-\mathbf{p}_u^*)^T\geq\mu\|\mathbf{p}_u^{k-1}-\mathbf{p}_u^*\|_2^2.\tag{S12}$$

Substituting (S12) into (S8) yields

$$\mathbb{E}\left[\|\mathbf{p}_u^{(k)}-\mathbf{p}_u^*\|_2^2\right]\leq\mathbb{E}\left[\|\mathbf{p}_u^{(k-1)}-\mathbf{p}_u^*\|_2^2\right]-2\eta\mu\mathbb{E}\left[\|\mathbf{p}_u^{k-1}-\mathbf{p}_u^*\|_2^2\right]+\eta^2\mathbb{E}\left[\|\nabla\varepsilon_{ui}(\mathbf{p}_u^{(k-1)})\|_2^2\right].\tag{S13}$$

Under **Lemmas 1** and **2**, we have the following corollary [23]:

$$\mathbb{E}\left[\left\|\nabla\varepsilon_{ui}\left(\mathbf{p}_u^{(k-1)}\right)\right\|_2^2\right]\le G^2. \tag{S14}$$

Substituting (S14) into (S13) yields

$$\mathbb{E}\left[\left\|\mathbf{p}_u^{(k)}-\mathbf{p}_u^{*}\right\|_2^2\right]\le\left(1-2\eta\mu\right)\mathbb{E}\left[\left\|\mathbf{p}_u^{k-1}-\mathbf{p}_u^{*}\right\|_2^2\right]+\eta^2G^2. \tag{S15}$$

By induction on *k*, we get

$$\mathbb{E}\left[\left\|\mathbf{p}_u^{(k)}-\mathbf{p}_u^{*}\right\|_2^2\right]\le\left(1-2\eta\mu\right)^{k-1}\mathbb{E}\left[\left\|\mathbf{p}_u^{1}-\mathbf{p}_u^{*}\right\|_2^2\right]+\sum_{j=0}^{k-2}\left(1-2\eta\mu\right)^{j}\left(\eta G\right)^2. \tag{S16}$$

Based on **Theorem 1**’s condition 0<η<1/2μ, we know that

$$\sum_{j=0}^{k-2}\left(1-2\eta\mu\right)^{j}<\sum_{j=0}^{\infty}\left(1-2\eta\mu\right)^{j}=\frac{1}{2\eta\mu}. \tag{S17}$$

Substituting (S17) into (S16) yields

$$\mathbb{E}\left[\left\|\mathbf{p}_u^{(k)}-\mathbf{p}_u^{*}\right\|_2^2\right]\le\left(1-2\eta\mu\right)^{k-1}\mathbb{E}\left[\left\|\mathbf{p}_u^{1}-\mathbf{p}_u^{*}\right\|_2^2\right]+\frac{\eta G^2}{2\mu}. \tag{S18}$$

According to the quadratic upper bound of the *L*-smooth function [23], we get

$$\begin{aligned}&\varepsilon_{ui}\left(\mathbf{p}_u^{k}\right)-\varepsilon_{ui}\left(\mathbf{p}_u^{*}\right)\le\nabla\varepsilon_{ui}\left(\mathbf{p}_u^{*}\right)\left(\mathbf{p}_u^{k}-\mathbf{p}_u^{*}\right)^T+\frac{L}{2}\left\|\mathbf{p}_u^{k}-\mathbf{p}_u^{*}\right\|_2^2\\&\Rightarrow\varepsilon_{ui}\left(\mathbf{p}_u^{k}\right)-\varepsilon_{ui}\left(\mathbf{p}_u^{*}\right)\le\frac{L}{2}\left\|\mathbf{p}_u^{k}-\mathbf{p}_u^{*}\right\|_2^2.\end{aligned} \tag{S19}$$

Combining (S18) and (S19), we get

$$\mathbb{E}\left[\varepsilon_{ui}\left(\mathbf{p}_u^{k}\right)-\varepsilon_{ui}\left(\mathbf{p}_u^{*}\right)\right]\le\frac{L}{2}\mathbb{E}\left[\left\|\mathbf{p}_u^{k}-\mathbf{p}_u^{*}\right\|_2^2\right]\le\frac{L}{2}\left[\left(1-2\eta\mu\right)^{k-1}\mathbb{E}\left[\left\|\mathbf{p}_u^{1}-\mathbf{p}_u^{*}\right\|_2^2\right]+\frac{\eta G^2}{2\mu}\right]. \tag{S20}$$

When the update count *k* is sufficiently large, i.e., *k*⟶∞,

$$\mathbb{E}\left[\varepsilon_{ui}\left(\mathbf{p}_u^{k}\right)-\varepsilon_{ui}\left(\mathbf{p}_u^{*}\right)\right]\le\frac{L\eta G^2}{4\mu}. \tag{S21}$$

Hence, **Theorem 1** stands. Further, in our learning rate adaption scheme, $\eta\rightarrow\eta_{lb}$ when *k*⟶∞. Therefore,

$$\mathbb{E}\left[\varepsilon_{ui}\left(\mathbf{p}_u^{k}\right)-\varepsilon_{ui}\left(\mathbf{p}_u^{*}\right)\right]\le\frac{L\eta_{lb}G^2}{4\mu}. \tag{S22}$$

Let $\Lambda(u)$ denotes entries related to node *u*, (S22) is reformulated as

$$\mathbb{E}\left[\sum_{r_{ui}\in\Lambda(u)}\varepsilon_{ui}\left(\mathbf{p}_u^{k}\right)-\varepsilon_{ui}\left(\mathbf{p}_u^{*}\right)\right]\le\frac{\left|\Lambda(u)\right|L\eta_{lb}G^2}{4\mu} \tag{S23}$$

For ∀*u*∈U, (S23) can be extended as

$$\mathbb{E}\left[\sum_{u\in U}\sum_{r_{ui}\in\Lambda(u)}\varepsilon_{ui}\left(\mathbf{p}_u^{k}\right)-\varepsilon_{ui}\left(\mathbf{p}_u^{*}\right)\right]\le\frac{\left|\Lambda\right|L\eta_{lb}G^2}{4\mu} \tag{S24}$$

V. Supplementary Empirical Studies

A. Implementation Details

1) The known entries of each HDI matrix are stored in coordinate format, i.e., a (u, i, r_{ui}) triple format;
2) The known entries of each HDI matrix are randomly partitioned into training, test, and validation sets, denoted as Ω, Ψ, and Φ, with a 7:1:2 ratio, respectively;
3) Considering the tradeoff between accuracy and computational efficiency, the LF dimension d of each LFA model is set to 20, and LF matrices are initialized to follow a normal distribution with mean 0 and variance 1/*d* [6], [13];
4) The regularization coefficient λ of each model is set at 0.03, the learning rate η of each competitor is set at 0.002 [13];
5) M1 requires set the initial value η_0, upper bound η_{ub} and lower bound η_{lb} of learning rate. We empirically set them as 0.002, 0.003 and 0.001, respectively [24];
6) M2-M8 all shuffle the HDI matrices to mitigate the load imbalance [13].
7) All experiments are performed on a machine equipped with two 2.1GHz Montage Jintide(R) C6230R CPUs (each with 26 cores, totaling 52 cores) and 1 TB RAM;
8) All experiments are repeated ten times to obtain relatively unbiased results;
9) All models are implemented in C++.

B. Ablation Study

This section conducts an ablation study on M1. We observe the effect on prediction accuracy and computational efficiency by gradually removing M1's one-cycle learning rate policy and the BAP. The two ablation models are designed as follows:

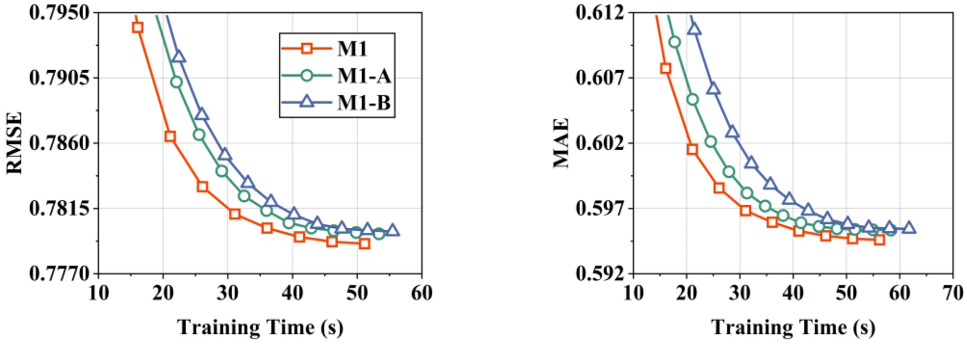
1) M1-A: One-cycle learning rate policy is removed, but the BAP is retained;

2) M1-B: Both the one-cycle learning rate policy and the BAP are removed, but shuffle is adopted.

TABLE S1

COMPARISON OF BAP AND SHUFFLE METHODS ACROSS DIFFERENT BLOCK PARTITIONS

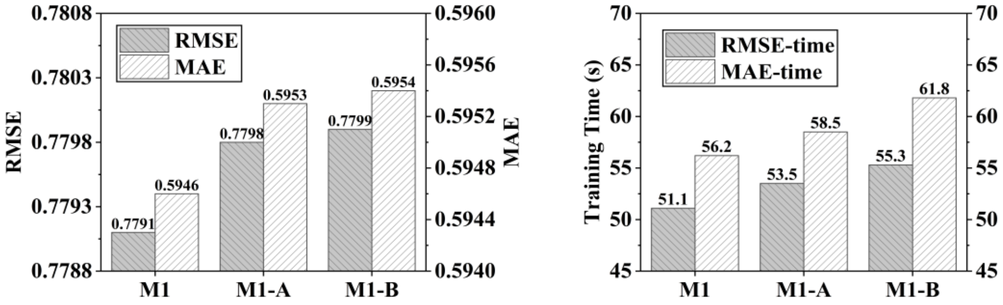
Dataset	Case	2×4		4×8		8×16		16×32		32×64	
		Shuffle	BAP	Shuffle	BAP	Shuffle	BAP	Shuffle	BAP	Shuffle	BAP
D1	Ratio	1.1136	**1.0028**	1.2267	**1.0242**	1.5265	**1.0256**	1.9693	**1.0698**	3.3244	**1.1890**
	Variance	4.69E+09	**2.32E+06**	6.22E+08	**6.36E+06**	1.00E+08	**4.10E+05**	1.16E+07	**7.45E+04**	1.82E+06	**2.07E+04**
D2	Ratio	1.0793	**1.0030**	1.1644	**1.0070**	1.4865	**1.0194**	1.8168	**1.0563**	2.5363	**1.1336**
	Variance	2.80E+09	**3.99E+06**	6.10E+08	**1.02E+06**	1.62E+08	**2.43E+05**	1.67E+07	**8.20E+04**	2.61E+06	**2.14E+04**
D3	Ratio	1.1224	**1.0015**	1.2301	**1.0065**	1.5056	**1.0247**	1.8136	**1.0651**	2.4915	**1.1500**
	Variance	1.33E+11	**1.17E+07**	2.24E+10	**1.39E+07**	3.33E+09	**5.70E+06**	3.65E+08	**1.59E+06**	4.66E+07	**2.97E+05**
D4	Ratio	1.0319	**1.0010**	1.0677	**1.0021**	1.1287	**1.0049**	1.1999	**1.0151**	1.3694	**1.0356**
	Variance	1.82E+10	**1.51E+07**	3.88E+09	**2.29E+06**	4.68E+08	**7.87E+05**	6.38E+07	**2.25E+05**	8.37E+06	**6.01E+04**



(a) RMSE versus training time

(b) MAE versus training time

Fig. S3. RMSE and MAE of M1 and its ablation models on D1 at 32 threads versus training time.



(a) Prediction accuracy

(b) Computational efficiency

Fig. S4. Comparison of M1 and its ablation models in prediction accuracy and computational efficiency on D1 at 32 Threads.

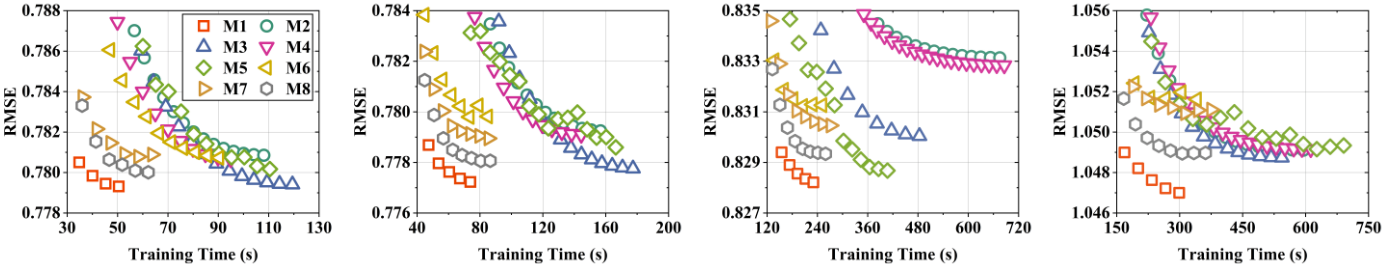
Fig. S3 depicts the RMSE and MAE of M1 and its ablation models on D1 at 32 threads versus training time. Fig. S4 lists the prediction accuracy and computational efficiency of M1 and its ablation models on D1 at 32 threads. Moreover, Table S1 compares the BAP and shuffle methods across different block partitions, computing the ratio of the block with the most entries to the block with the fewest entries, and the variance in entry counts among all blocks. From these results, we find that:

1) **BAP alleviates the impact of load imbalance, achieving favorable convergence results.** As shown in Table S1, compared to shuffle, BAP reduces the ratio and variance on D1 at 32×64 blocks from 3.3244 and 1.82E+06 to 1.1890 and 2.07E+04, respectively. The other cases present similar results. This indicates that BAP achieves better balance in the number of entries across blocks. Moreover, as shown in Fig. S3 and Fig. S4, M1-A, which uses BAP, demonstrates better convergence results compared to M1-B, which employs shuffle. This suggests that BAP effectively reduces M1's convergence time by balancing the number of entries across blocks. For example, compared to the 55.3 seconds convergence time of M1-B on RMSE in Fig. S4, M1-A's convergence time drops to 53.5 seconds, improving computational efficiency by 3.3%. Similar results can be observed in MAE.

2) **One-cycle learning rate policy demonstrates improvements in convergence speed, prediction accuracy, and computational efficiency**. As illustrated in Fig. S3, M1 incorporating the one-cycle learning rate policy converges more rapidly on both RMSE and MAE compared to M1-A. This is further supported by Fig. S4, which shows that the training times for RMSE and MAE are both reduced when using the policy. For example, M1 reduces RMSE training time from 53.5

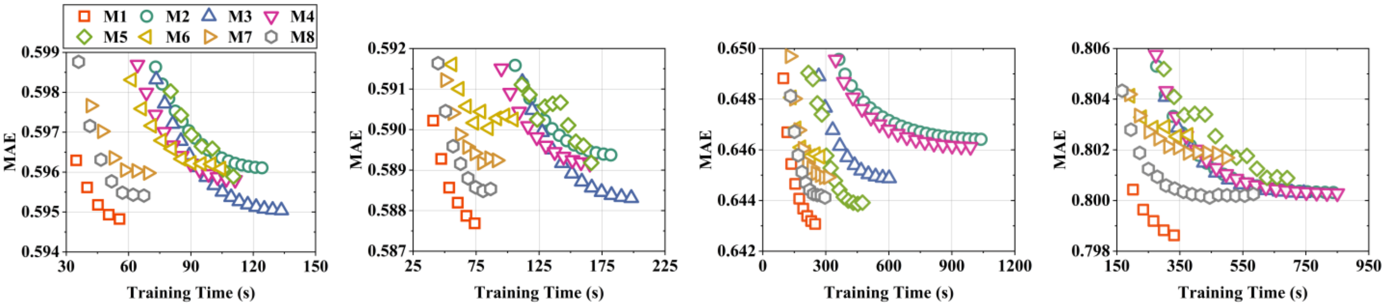
seconds to 51.1 seconds, obtaining a 4.5% gain in computational efficiency. This improvement is attributed to the policy's ability to rapidly explore the para meter space in the early training stage through a linearly increasing learning rate. Furthermore, the use of a cosine annealing learning rate in the later training stage allows for a more precise optimization of the parameters, resulting in an accuracy gain. Specifically, M1-A's RMSE decreases from 0.7798 to M1's 0.7791 after incorporating the one-cycle learning rate policy.

C. Comparison Results and Analysis



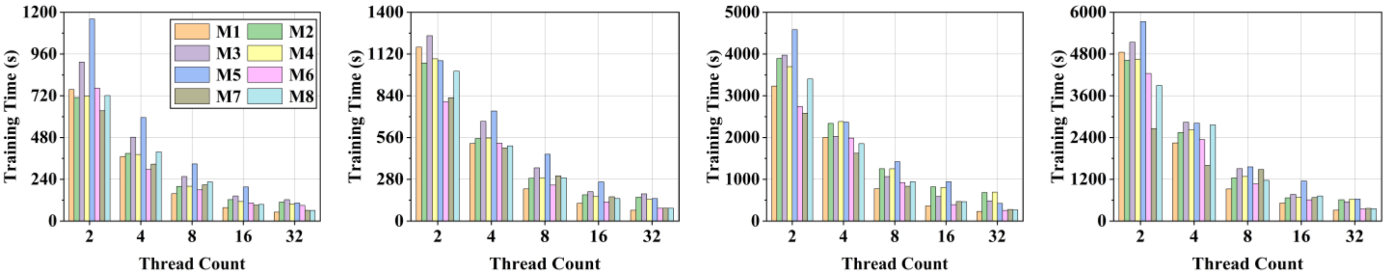
(a) D1-RMSE (b) D2-RMSE (c) D3-RMSE (d) D4-RMSE

Fig. S5. M1-8's RMSE on D1-4 at 32 threads versus training time. All panels share the legend of panel Fig. S5(a).



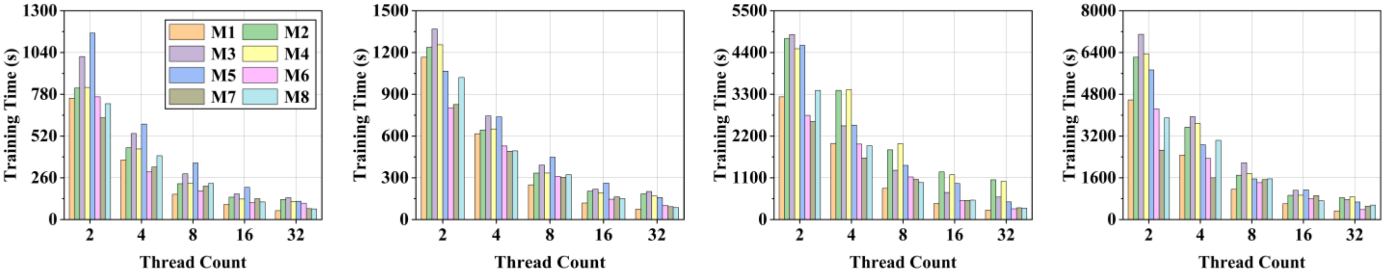
(a) D1-MAE (b) D2- MAE (c) D3- MAE (d) D4- MAE

Fig. S6. M1-8's MAE on D1-4 at 32 threads versus training time. All panels share the legend of panel Fig. S6(a).



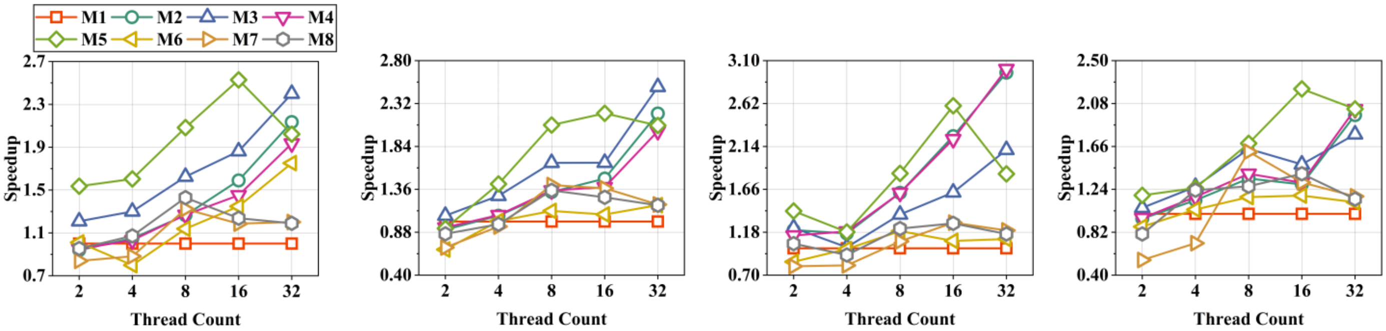
(a) D1-RMSE (b) D2-RMSE (c) D3-RMSE (d) D4-RMSE

Fig. S7. M1-8's RMSE training time on D1-4 versus thread count. All panels share the legend of panel Fig. S7(a).



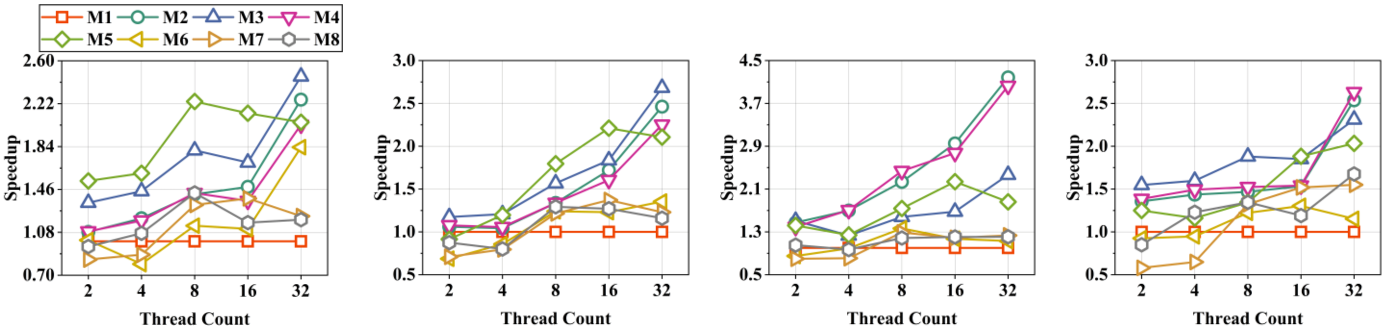
(a) D1-MAE (b) D2- MAE (c) D3- MAE (d) D4- MAE

Fig. S8. M1-8's MAE training time on D1-4 versus thread count. All panels share the legend of panel Fig. S8(a).



(a) D1-RMSE (b) D2-RMSE (c) D3-RMSE (d) D4-RMSE

Fig. S9. M1-8's RMSE speedup on D1-4 versus thread count. All panels share the legend of panel Fig. S9(a).



(a) D1-MAE (b) D2- MAE (c) D3- MAE (d) D4- MAE

Fig. S10. M1-8's MAE speedup on D1-4 versus thread count. All panels share the legend of panel Fig. S10(a).

TABLE S2

M1-8's Speedup Comparison at 32 Threads

Dataset	CASE	M1	M2	M3	M4	M5	M6	M7	M8
D1	**RMSE-speedup**	1×	2.13×	2.40×	1.93×	2.02×	1.75×	1.20×	1.19×
	MAE-speedup	1×	2.26×	2.46×	2.03×	2.06×	1.83×	1.22×	1.19×
D2	**RMSE-speedup**	1×	2.21×	2.51×	2.01×	2.07×	1.18×	1.19×	1.18×
	MAE-speedup	1×	2.46×	2.68×	2.25×	2.11×	1.35×	1.23×	1.16×
D3	**RMSE-speedup**	1×	2.97×	2.10×	3.01×	1.83×	1.10×	1.20×	1.16×
	MAE-speedup	1×	4.19×	2.37×	4.03×	1.86×	1.13×	1.24×	1.21×
D4	**RMSE-speedup**	1×	1.97×	1.78×	2.02×	2.03×	1.11×	1.17×	1.14×
	MAE-speedup	1×	2.54×	2.31×	2.63×	2.03×	1.15×	1.55×	1.68×
	Win/Loss	-	8/0	8/0	8/0	8/0	8/0	8/0	8/0
Friedman Rank		**1.00**	7.00	6.88	6.38	5.75	2.81	3.50	2.69

TABLE S3

The Wilcoxon's Signed-Rank Test Results based on TABLE III-V

Competitor	Prediction Accuracy			Computational Efficiency			Scalability		
	R^+	R^-	P-value	R^+	R^-	P-value	R^+	R^-	P-value
M1 vs. M2	36	0	0.012	36	0	0.012	36	0	0.012
M1 vs. M3	36	0	0.012	36	0	0.012	36	0	0.012
M1 vs. M4	36	0	0.012	36	0	0.012	36	0	0.012
M1 vs. M5	36	0	0.012	36	0	0.012	36	0	0.012
M1 vs. M6	36	0	0.012	36	0	0.012	36	0	0.012
M1 vs. M7	36	0	0.012	36	0	0.012	36	0	0.012
M1 vs. M8	36	0	0.012	36	0	0.012	36	0	0.012