# Accelerating Multi-Output GBDTs with GPUs

Hanfeng Liu[1], Xuemei Peng[1], Zeyi Wen[1,2*]
[1]The Hong Kong University of Science and Technology (Guangzhou)
[2]The Hong Kong University of Science and Technology
{hliu174,xpeng558}@connect.hkust-gz.edu.cn,wenzeyi@ust.hk

## ABSTRACT

Gradient Boosted Decision Trees (GBDTs) have demonstrated good performance in many data science competitions. This paper studies multidimensional output GBDT (GBDT-MO) training which can handle complex dependencies between input features and multidimensional outputs. Due to the multiple output dimensionality, the training time and memory consumption of GBDT-MO are significantly higher than those of the single-dimensional output GBDT training, which makes training GBDT-MO challenging, especially for large-scale datasets. In this paper, we propose a novel GPU-accelerated GBDT-MO training system to speed up the training while maintaining competitive model quality. Our system leverages the GPU to efficiently construct decision trees, especially by dynamically choosing efficient histogram building methods at different stages of the training, enabling scalable and efficient GBDT-MO training. Furthermore, our system supports multi-GPU training on a single machine by partitioning features across GPUs and using communication-efficient synchronization strategies, allowing further scalability for high-dimensional datasets. We evaluate the performance of our proposed GBDT-MO system on several datasets. Compared with CPU-based implementations, our system achieves a speedup ranging from 30× to 190×. Moreover, our system outperforms the state-of-the-art GPU-based baselines by a speedup ranging from 1.7× to 170×, while maintaining strong predictive performance compared with the state-of-the-art methods.

**ACM Reference Format:**
Hanfeng Liu[1], Xuemei Peng[1], Zeyi Wen[1,2]. 2025. Accelerating Multi-Output GBDTs with GPUs. In . ACM, San Diego, CA, 10 pages. https://doi.org/10.1145/3754598.3754638

## 1 INTRODUCTION

Gradient Boosted Decision Trees (GBDTs) are a powerful ensemble method in machine learning, widely used in computer vision, natural language processing, and recommendation systems [29]. GBDTs excel at capturing complex feature interactions and have a proven track record of delivering top-notch performance in both data science competitions [12] and real-world applications [4, 8, 20]. Multidimensional output learning aims to simultaneously predict multiple target variables, which has been employed in various tasks such as multi-task learning [1], multi-label classification [18], and multi-output regression [25]. Among the various multi-output models, GBDTs with multiple outputs (GBDT-MO) stand out due to their ability to handle complex dependencies between input features and multiple outputs, leading to improved predictive performance and interpretability [23].

GBDT-MO offers several significant advantages over training separate single-output GBDT (GBDT-SO) models for each output. GBDT-MO can model the correlations and dependencies between different output variables, which is particularly valuable in tasks where outputs are related (e.g., predicting multiple disease outcomes that share common risk factors). Notably, instead of training $d$ separate tree ensembles for $d$ output dimensions, GBDT-MO efficiently constructs a single ensemble of trees with multi-dimensional leaf outputs, reducing both memory footprint and computational training overhead. These advantages have enabled GBDT-MO to achieve success in diverse real-world applications, including simultaneous prediction of multiple disease outcomes in healthcare [30, 35], click prediction in recommendation systems [29], and multi-step traffic forecasting [31].

Efficient GPU implementations for GBDT-MO algorithms remain underexplored [13], despite the significant advancements in accelerating single-output GBDT training using GPUs [5, 16, 28]. This gap persists primarily due to several key challenges associated with GBDT-MO. First, GBDT-MO requires more complex data structures and algorithms to handle multiple outputs simultaneously. Second, its memory requirements are significantly higher due to the need to store gradient statistics for multiple outputs. Lastly, the computational workload per tree in GBDT-MO is greater due to the multi-dimensional nature of the outputs.

This paper addresses the gap by introducing an efficient GPU-accelerated GBDT-MO training system. Unlike existing GPU-based GBDT implementations such as LightGBM and XGBoost, our system stands out in four key aspects: *i) Multi-output focus*: Unlike existing systems that primarily target single-output problems, our solution is specifically designed for multi-output scenarios, addressing the unique challenges they present. *ii) Adaptive histogram building*: We identify that histogram building is the primary bottleneck in GBDT-MO training, accounting for approximately 70% of the total training time. Our solution dynamically selects the most appropriate histogram building method from multiple optimized approaches based on the dataset characteristics and training stage. *iii) Memory efficiency*: We redesign the complete training pipeline (loss calculation, gradient computation, split evaluation) with GPU-native operations. To handle the significantly higher memory requirements of GBDT-MO, we adopt tiling to utilize shared memory and bin packing to optimize memory access. *iv) Multi-GPU training*: Our system supports multi-GPU training within a single node. By

*Corresponding Author


*ICPP'25, September 2025, San Diego, CA*

ACM ISBN 979-8-4007-2074-1
https://doi.org/10.1145/3754598.3754638

distributing feature columns across GPUs and synchronizing partial histograms via efficient collective communication, we enable high-throughput training even for high-dimensional or large-scale datasets.

The major contributions can be formally summarized as:

- We design a comprehensive GPU-accelerated system for GBDT-MO, where all major components—including loss computation, gradient calculation, gain evaluation, and prediction—are implemented on the GPU using efficient parallel primitives.
- We identify the histogram building process as a key bottleneck in GBDT-MO training and propose a parallel method to construct histograms efficiently using both global and shared memory on GPUs. We further optimize this process with warp-level techniques, including data compression to reduce bank conflicts.
- We benchmark our proposed GBDT-MO system across a range of real-world datasets and observe acceleration from 30× to 190× over CPU-based counterparts, and from 1.7× to 170× over the leading GPU-based systems. This demonstrates that our system achieves superior training efficiency while delivering strong predictive performance. The experimental results under different parameter settings also demonstrate the good scalability of our proposed system.

## 2 PRELIMINARY

In this section, we present the background knowledge of multi-output GBDTs. We first formulate the learning objective of multi-output GBDTs and then describe the process of finding split points.

### 2.1 GBDTs with multiple outputs

As illustrated in Figure 1, the learned trees of GBDTs with a single output (GBDT-SO) and GBDTs with multiple outputs (GBDT-MO) are notably different in their leaves. In regression tasks with a single output, GBDT-SO is indeed a special case of GBDT-MO, where they both produce identical tree structures. However, in multi-class classification and multi-label classification tasks, they become much more different, and the models trained by GBDT-MO are much simpler. We take a multi-class classification task with $d$ classes as an example. GBDT-SO needs to train $d \times |T|$ trees, where $|T|$ is the number of trees trained for each class. In comparison, GBDT-MO only requires training $|T|$ trees with the leaf nodes storing multiple class information (cf. Figure 1). The model complexity of GBDT-SO is $d$ times higher than GBDT-MO, and its cost increases significantly for high-dimensional problems.

While GBDT-MO can keep the number of trees independent from the number of classes, it is important to recognize that designing an efficient GBDT-MO training algorithm is still challenging. For instance, memory usage substantially escalates during the histogram building phase because of the inclusion of the output dimension (i.e., $d$). Consequently, the adaptation of GBDT-MO for GPU platforms requires careful design to tackle the challenges related to memory and computational efficiency effectively.

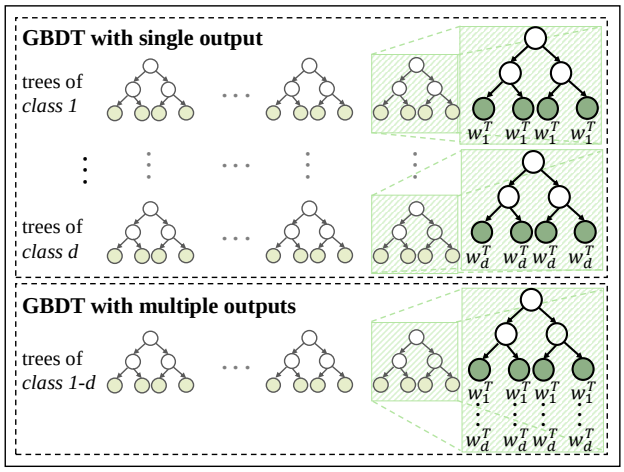



**Figure 1: GBDTs with single output and multiple outputs.**

### 2.2 Loss and objective function

A multi-output dataset with $n$ training instances can be denoted as $\{(\mathbf{x}_i, \mathbf{y}_i)_{i=1}^n\}$, where $\mathbf{x}_i \in \mathbb{R}^m$ is an $m$ dimensional input and $\mathbf{y}_i \in \mathbb{R}^d$ is a $d$ dimensional output. The GBDT is a type of ensemble learning method that trains a series of decision trees sequentially to make predictions on unseen instances. Let $f_t(\cdot) : \mathbb{R}^m \to \mathbb{R}^d$ be the function of the $t$-th decision tree. According to the construction of decision trees, $f_t(\mathbf{x})$ can be expressed by the following formula:

$$f_t(\mathbf{x}) = \sum_{j=1}^{L} \mathbf{v}_j \cdot \mathbf{1}(q(\mathbf{x}) = j)$$

where $\mathbf{v}_j$ is a $d$ dimensional vector that belongs to the $j$-th leaf and $\mathbf{1}(q(\mathbf{x}) = j)$ is an indicator function that takes the value 1 if $q(\mathbf{x}) = j$ and 0 otherwise. The function $q(\cdot) : \mathbb{R}^m \to j$, where $L$ is the number of leaves of the $t$-th decision tree and $j \in [1, L]$. The role of $q(\cdot)$ is to identify the leaf node corresponding to $\mathbf{x}$.

Then, the objective function at the $t$-th iteration can be:

$$\mathcal{L}_t(f) = \sum_{i=0}^{n} l(\mathbf{y_i}, \hat{\mathbf{y}}_i + f_t(\mathbf{x}_i)) + \Omega(f_t) \tag{1}$$

where $l(\cdot)$ is the loss function and $\hat{\mathbf{y}}_i = \sum_{k=1}^{t-1} f_k(\mathbf{x}_i)$ is the prediction of the first $(t-1)$ trees. $\Omega(f_t)$ is the regularization term, which constrains the model's complexity and improves the model's generality.

The leaf nodes in a decision tree are independent of each other. As a result, we can apply the objective function of Equation (1) to each leaf $j$ independently. Then for the leaf $j$ of the decision tree $t$, we have

$$\mathcal{L}_t(\mathbf{v}_j) = \sum_{q(\mathbf{x}_i)=j} l(\mathbf{y_i}, \hat{\mathbf{y}}_i + \mathbf{v}_j) + \Omega(\mathbf{v}_j)$$

We employ $l_2$ regularization, i.e., $\Omega(\mathbf{v}) = \frac{\lambda}{2}\|\mathbf{v}\|_2^2$, on leaf values with a parameter $\lambda > 0$. By using the second-order Taylor expansion to approximate the loss function $l(\cdot)$, we can get

$$\mathcal{L}_t(\mathbf{v}_j) = \sum_{q(\mathbf{x}_i)=j} \left\{ l(\mathbf{y_i}, \hat{\mathbf{y}}_i) + \mathbf{g}_i^T \mathbf{v}_j + \frac{1}{2}\mathbf{v}_j^T \mathbf{H}_i \mathbf{v}_j \right\} + \frac{\lambda}{2}\|\mathbf{v}_j\|_2^2 \tag{2}$$

where $\mathbf{g}_i = \frac{\partial l}{\partial \hat{y}_i}$ and $\mathbf{H}_i = \frac{\partial l}{\partial \hat{y}_i^2}$. Training GBDTs is to minimize the objective function iteratively. Then, we can set $\frac{\partial \mathcal{L}_t}{\partial \mathbf{v}_j} = 0$ for Equation (2), and we get the optimal values and objective of the leaf $j$ as follows.

$$\mathbf{v}_j^* = -\left(\sum_{q(\mathbf{x}_i)=j} \mathbf{H}_i + \lambda \mathbf{I}\right)^{-1} \left(\sum_{q(\mathbf{x}_i)=j} \mathbf{g}_i\right)$$

$$\mathcal{L}_t^*(\mathbf{v}_j) = -\frac{1}{2}\left(\sum_{q(\mathbf{x}_i)=j} \mathbf{g}_i\right)^T \left(\sum_{q(\mathbf{x}_i)=j} \mathbf{H}_i + \lambda \mathbf{I}\right)^{-1} \sum_{q(\mathbf{x}_i)=j} \mathbf{g}_i$$

where $\mathbf{I}$ is an identity matrix.

To avoid computationally expensive matrix inversions, it is common practice to simplify the Hessians $\mathbf{H}$ to diagonal matrices. Thus, the loss function $l$ can be independently computed for each output. For the $k$-th dimension, the optimal leaf values can be rewritten as:

$$|\mathbf{v}_j^*|_k = -\frac{\sum_{q(\mathbf{x}_i)=j} |\mathbf{g}_i|_k}{\sum_{q(\mathbf{x}_i)=j} |\mathbf{h}_i|_k + \lambda}$$

where $\mathbf{h}_i$ is a diagonal element of $\mathbf{H}_i$ and $|\cdot|_k$ denotes the $k$-th element of the vector. Then, the overall objective can be obtained by the sum of objectives for each output dimension as follows.

$$\mathcal{L}_t^*(\mathbf{v}_j) = -\frac{1}{2}\sum_{k=1}^{d}\left\{\frac{(\sum_{q(\mathbf{x}_i)=j} |\mathbf{g}_i|_k)^2}{\sum_{q(\mathbf{x}_i)=j} |\mathbf{h}_i|_k + \lambda}\right\}$$

## 2.3 Gain of a split point

A crucial task when building decision trees includes identifying the best split given a set of training instances. To determine these splits, we consider all the candidate split points for a node pending to be split, including left and right splits denoted by $S_L$ and $S_R$. Then the gain for each split is:

$$\begin{aligned} gain &= \mathcal{L}^*(\mathbf{v}) - (\mathcal{L}^*(\mathbf{v_L}) + \mathcal{L}^*(\mathbf{v_R})) \\ &= \frac{1}{2}\sum_{k=1}^{d}\left\{\frac{(G_L^k)^2}{H_L^k + \lambda} + \frac{(G_R^k)^2}{H_R^k + \lambda} - \frac{(G_L^k + G_R^k)^2}{H_L^k + H_R^k + \lambda}\right\} \end{aligned} \tag{3}$$

where $G_L^k = \sum_{q(\mathbf{x}_i)=R_L} |\mathbf{g}_i|_k, G_R^k = \sum_{q(\mathbf{x}_i)=R_R} |\mathbf{g}_i|_k, H_L^k = \sum_{q(\mathbf{x}_i)=R_L} |\mathbf{h}_i|_k$ and $H_R^k = \sum_{q(\mathbf{x}_i)=R_R} |\mathbf{h}_i|_k$.

## 2.4 Training GBDTs with multiple outputs

Here, we describe the training process of GBDTs with multiple outputs (GBDT-MO), which can be summarized in Algorithm 1. Similar to the single output GBDT (GBDT-SO), two key components are added in the algorithm. The first component aims to find the best split point given a tree node, as shown in Lines 5 to 13. The second component is the process of splitting the instances in the parent node into its children, as shown from Line 14 to Line 17. The key difference between GBDT-MO and GBDT-SO training algorithms is that GBDT-MO needs to calculate a $g$ and $h$ pair for each output, which is shown in the for-loop (i.e., Line 11).

**Algorithm 1:** Training of GBDTs with multiple outputs

**Input :** $\mathcal{I}$: a set of instances; $d$: the maximum depth; $\gamma$: threshold for valid splits; $T$: the # of trees; $d\_outputs$: output dimension.
**Output :** $\mathcal{T}$: a set of decision trees

```
1  T ← ∅;
2  repeat
3      InitTree(t): P ← ∅, N ← GetRootNode(t);
4      foreach n ∈ N ∧ depth(n) < d do
5          I_n ← InstanceInNode(n);
6          (g*, a*, p*) ← (0, ∅, ∅);
7          foreach a ∈ A do
8              V_n ← AttributeValue(a, n);
9              foreach j ∈ V_n do
10                 gain ← 0;
11                 for v ∈ [1, d_outputs) do
12                     gain += CalculateGain(a, j, v);
13         (a*, g*, p*) ← FindBestSplit();
14         if g* = 0 then
15             RemoveLeafNode(n, N);
16         else
17             UpdateLeafNode(N, SplitNode(n, a*, p*));
18     T ← T ∪ {t};
19 until |T| > T;
```

# 3 OUR PROPOSED SOLUTION

In this section, we present the design and implementation of our GPU-accelerated GBDT-MO training system. Our solution is developed to address the computational and memory challenges of training gradient boosted decision trees with multi-dimensional outputs, particularly on large-scale datasets. We begin with a high-level overview of the system architecture and its modular components. Then, we describe the core tree construction pipeline, including efficient gradient computation and split evaluation techniques. We also introduce several GPU-specific optimizations, such as histogram building strategies and warp-level memory techniques, to improve training efficiency. Finally, we discuss how our system scales to support multiple GPUs and handles inference efficiently.

## 3.1 System overview

Figure 2 illustrates the overall workflow of our proposed GBDT-MO system. The training pipeline is organized into three key stages: 1) computing the gradients ($g_i$) and second-order derivatives ($h_i$) based on task-specific loss functions such as mean squared error (MSE) or cross-entropy, 2) producing candidate split points by enumerating feature values and constructing histograms, and 3) selecting the best split point that maximizes the gain and partitioning the data accordingly to construct the tree.

To enhance computational efficiency, particularly on GPU architectures, our system integrates a suite of optimization techniques tailored for different stages of the pipeline. These include sparsity-aware computation, histogram-building strategies (global memory,

shared memory, and sort-and-reduce), as well as warp-level optimizations. Each of these techniques is designed to address specific bottlenecks, such as memory access inefficiencies or thread-level conflicts, and is tightly integrated into the tree construction process.

Notably, in multi-class or multi-label learning scenarios, GBDT-MO achieves significant computational advantages by training a single set of trees with multi-dimensional outputs at the leaf nodes. In contrast, traditional single-output GBDTs (GBDT-SO) require training $k$ separate tree ensembles—one per class or label—where $k$ is the number of outputs. This consolidation in GBDT-MO leads to substantial reductions in both computational cost and memory usage, making it especially suitable for large-scale learning tasks.

*3.1.1 Computing g and h for each instance.* To compute the gain for candidate split points, we first calculate the first- and second-order derivatives, $g_i$ and $h_i$, for each instance. To ensure the system's flexibility and general applicability, GBDT-MO is designed to accommodate user-defined loss functions [11]. In this paper, we select mean squared error as our loss function for demonstration purposes, yielding $g_i = 2(\hat{y}_i - y_i)$ and $h_i = 2$. The predicted value $\hat{y}_i$ can be updated incrementally by reusing results from previous trees, rather than recomputing from scratch via full tree traversal. This avoids excessive irregular memory accesses on GPUs. To further reduce overhead, we exploit the observation that training instances eventually reside in leaf nodes. Thus, we skip traversal altogether and directly retrieve the leaf weights for prediction, significantly improving computational efficiency.

*3.1.2 Producing split point candidates.* Once we have obtained the values of $g_i$ and $h_i$, we need to enumerate all feature values for the current node to generate candidate split points. For each candidate split points, we calculate $G_L$, $G_R$, $H_L$, and $H_R$, where $G_L$ and $G_R$ are the gradient sums of the left and right node, respectively; similarly, $H_L$ and $H_R$ are their Hessian value sums respectively, as defined in Equation (3). This enumeration allows us to consider each possible feature value as a potential split point. Alternatively, we can build histograms of the feature bins, which helps in efficiently identifying the most promising candidate split points. Since the attribute values are sorted in the training data, we can easily consider all the instances to the left or right of a candidate split point as belonging to the corresponding left or right node. Consequently, we can efficiently compute the aggregated $g_i$ and $h_i$ values for the left and right nodes (i.e., $G_L$ and $G_R$). The $G_L$ and $G_R$ can be calculated by enumerating all possible feature values or by leveraging cut points generated in the histogram construction phase.

*3.1.3 Selecting the split point with maximum gain.* We use the the histograms constructed in the previous phase to compute the gain as shown in Equation (3). The computation of $G_L$ and $H_L$ for each node can be achieved using a segmented prefix sum, so that multiple gains can be computed concurrently on the GPU. After computing the gain for all candidate split points, we select the best split for each feature within a node using segmented and global parallel reduction techniques. Segmented reduction enables parallel gain comparison across multiple feature-node pairs, where each pair forms a *segment*. This is critical for high-throughput multi-output GBDT training. A naive one-block-per-segment mapping becomes inefficient on high-dimensional datasets due to kernel launch overhead. To address this, we adopt an adaptive strategy that determines the number of segments per block based on the dataset and hardware: $1 + \frac{\#\text{ segments}}{\#\text{ SMs}} \times C$, where $C$ is a tunable constant and SMs is the number of GPU streaming multiprocessors. This approach balances resource usage and improves scalability. Finally, gain values are computed using the histograms constructed earlier. Segmented reduction identifies the best threshold within each feature, and a global reduction finds the overall best split across all features in the current node.

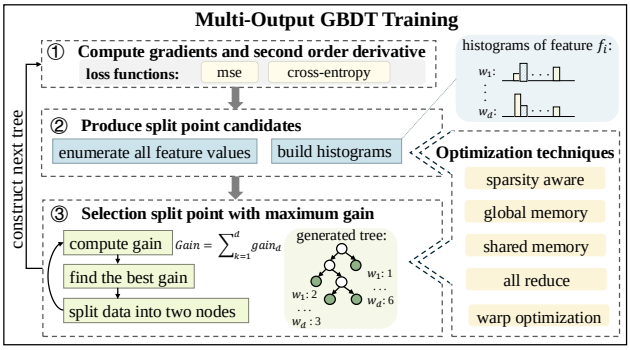



**Figure 2: Overview of the proposed system.**

## 3.2 Sparsity-aware data storage in GBDT-MO

The training dataset of GBDT-MO can be represented in two primary forms: dense and sparse. The dense representation is essentially a matrix, which offers high efficiency when accessing attribute values for a given instance. For instance, finding the third attribute of the fourth instance (i.e., $a_3$ of $\boldsymbol{x_4}$) is as simple as locating it at the intersection of the third column and the fourth row in the matrix. However, one drawback of this approach is its substantial memory consumption. On the other hand, the sparse representation only stores non-zero elements, making it more memory-efficient when the training data is sparse. It comes with a higher overhead when locating attribute values for the instances. As an illustration, we consider a training dataset with five instances: $\boldsymbol{x_0}$, $\boldsymbol{x_1}$, $\boldsymbol{x_2}$, $\boldsymbol{x_3}$, and $\boldsymbol{x_4}$, and we present both the dense and sparse representations for this training dataset.

| | dense | sparse |
|---|---|---|
| $\boldsymbol{x_0}$ | <0, 0, 3, 0, 0> | ($a_2$: 3) |
| $\boldsymbol{x_1}$ | <2, 0, 0, 0, 7> | ($a_0$: 2); ($a_4$: 7) |
| $\boldsymbol{x_2}$ | <0, 6, 0, 0, 0> | ($a_1$: 6) |
| $\boldsymbol{x_3}$ | <0, 0, 0, 0, 0> | |
| $\boldsymbol{x_4}$ | <1, 0, 0, 0, 8> | ($a_0$: 1); ($a_4$: 8) |

Sparse data is common in many real-world applications, such as natural language processing and computer vision [6]. Given the inefficiency of dense representations in terms of computational and storage resources, especially for large and sparse datasets, GBDT-MO adopts the sparse data representation in the form of the Compressed Sparse Column (CSC) format. The CSC representation consists of three arrays: the non-zero values, the row indices of

these values, and the column pointers. For non-zero values, we traverse the matrix column-wise.

$$values = [2, 1, 6, 3, 7, 8]$$

For row indices, we store the row number of each non-zero value.

$$row\_indices = [1, 4, 2, 0, 1, 4]$$

The column pointer array has an additional element indicating the end of the last column. The $i^{th}$ element in the column pointer array indicates the index in the value array where the $i^{th}$ column starts.

$$col\_pointers = [0, 2, 3, 4, 4, 6]$$

Hence, the CSC representation of the training data is given by the three arrays: *values*, *row_indices*, and *col_pointers*. The use of the CSC format in our solution provides an efficient and compact way to store and access sparse data, thereby enhancing the performance of our subsequent analyses. Moreover, by reducing memory usage, the CSC format enables our method to work with larger datasets that would otherwise be impractical with the dense data representation.

*Column-wise data distribution for GPU parallelism.* To effectively leverage the massive parallel computing capabilities of GPUs, our proposed solution assigns computational tasks based on attribute columns. Specifically, each GPU thread block is responsible for computations related to one or more specific attribute columns. Within each thread block, warps (groups of 32 threads) concurrently process different training instances of these assigned attribute columns. For example, suppose that we have $m$ features and $n$ training instances. These $m$ columns are distributed across the GPU thread blocks. Within each block, threads or warps perform parallel computations on instances corresponding to their assigned columns.

## 3.3 Histogram building optimization

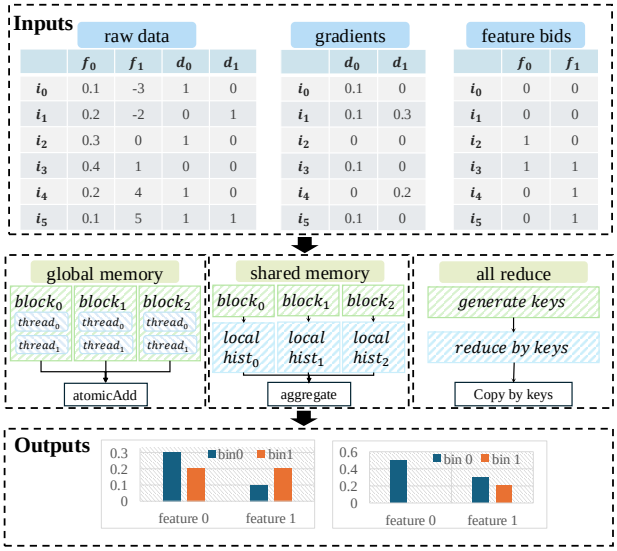



**Figure 3: Histogram building process on the GPU.**

Histogram building is a critical and computationally intensive step in training GBDT-MO, aiming to aggregate gradient statistics efficiently for optimal split-point identification. The process is depicted in Algorithm 2. By summarizing gradient and Hessian values into discrete bins for each feature, histogram-based approaches significantly reduce computational complexity. Figure 3 presents a visualization of the histogram construction pipeline in GPU-accelerated decision tree training. It elucidates the complete workflow from raw data discretization to gradient aggregation, highlighting three divergent GPU memory utilization strategies. Through its multi-stage representation, the figure shows: (1) input data preparation with feature-value bin mapping, (2) parallel histogram construction via global memory access, shared memory optimization, and sort-and-reduce approaches, and (3) final aggregated histograms for split finding.

**Algorithm 2:** Histogram building in GBDT training

**Input** : $\mathcal{S}$: training instances in node $n$;
$m$: number of features;
$\#bins$: maximum number of bins
**Output**: $Hist$: generated histogram

```
Init(Hist);
for i ∈ [0, m) do
    for j ∈ [0, d_outputs] do
        for k ∈ [0, #instances] do
            bid ← GetBid(x_jk);
            Hist.g[bid][j] ← Hist.g[bid][j] + g_jk;
            Hist.h[bid][j] ← Hist.h[bid][j] + h_jk;
```

*3.3.1 Performance analysis.* As illustrated in Figure 4, we break down the total training time and quantify the portion spent on histogram building for each dataset. The grey bars represent the total training time, while blue bars denote the histogram building time and the red annotations indicate the percentage attributable to histogram construction. Our analysis shows that histogram building remains the dominant time consumer across nearly all datasets. Notably, in *Delicious*, *NUS-WIDE*, and *MNIST*, this step accounts for 88.5%, 88.3%, and 78.5% of the total training time, respectively. Even for medium-scale datasets like *Caltech101* and *MNIST-IN*, the ratios are 67.2% and 77.9%. These findings reaffirm that histogram building is the primary computational bottleneck in GBDT-MO training, motivating our targeted GPU-based optimizations.

*3.3.2 Global memory approach.* The global memory approach leverages GPU global memory combined with `atomicAdd` operations to safely aggregate updates from multiple threads. Each thread independently processes an instance-feature pair: it computes the bin ID from the input feature value, then directly updates the corresponding gradient and Hessian accumulators in global memory. This approach offers implementation simplicity and scalability for moderate workloads. However, when many threads access and modify the same bin concurrently, contention over global memory leads to performance degradation due to atomic operation serialization and memory bank conflicts. This makes the method more suitable for small- to medium-scale datasets.

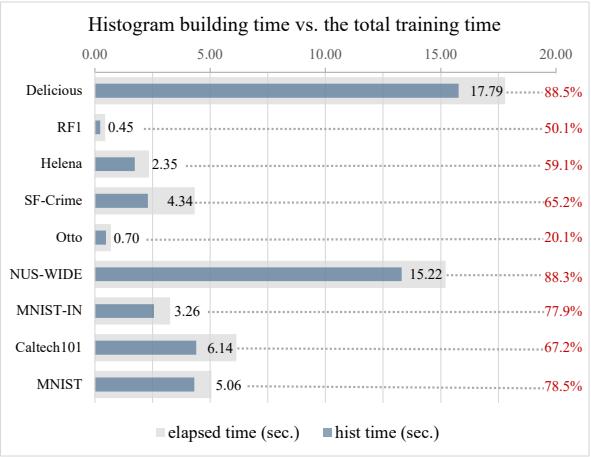



**Figure 4: Histogram building time vs. the total training time.**

*3.3.3 Shared memory approach.* To alleviate global memory contention, the shared memory approach exploits the low-latency, high-bandwidth shared memory on GPUs. Since shared memory is limited in size (typically 48 KB per thread block), we adopt a tiling strategy: the full histogram is divided into sub-histograms (tiles) that fit within shared memory. Threads in a block collaboratively initialize a local histogram, accumulate gradient statistics into it, and then synchronize to flush the results back into the global histogram. This method significantly reduces atomic conflicts and improves memory access patterns. The tiling parameters—chunk size and bin offset—are computed dynamically per block, enabling the method to handle moderately large datasets and feature dimensions efficiently.

*3.3.4 Sort-and-reduce approach.* The sort-and-reduce approach bypasses atomic operations altogether. Each thread constructs a unique key for its instance-feature-bin combination using the bin ID and a precomputed feature offset. These keys and associated gradient pairs are stored in temporary arrays. A parallel `sort_by_key` operation groups identical keys together, followed by a `reduce_by_key` to accumulate gradient and Hessian values. Finally, the reduced results are copied into the final histogram structure. While this method introduces the overhead of sorting, it avoids write contention entirely, making it ideal for high-dimensional or large-scale datasets where atomic conflict is severe.

Each approach presents trade-offs between simplicity, memory efficiency, and contention resilience, as reflected in Figure 3. Our system dynamically selects the most appropriate histogram building method from multiple optimized approaches based on the dataset characteristics and training stage, enabling scalable and efficient GBDT-MO training.

## 3.4 Warp-level and multi-GPU optimization

We implement additional optimizations through two key techniques to enhance system efficiency: warp-level memory optimization and multi-GPU parallel execution.

*3.4.1 Warp-level optimization.* On GPUs, threads are grouped into warps (typically 32).This design allows all threads within a warp to perform simultaneously, forming the foundation of GPU parallel processing. In GBDT training implementations like XGBoost [5], the maximum number of bins is normally set to 256. While this constraint allows efficient storage of bin IDs using single-byte char type, it creates memory access inefficiencies on GPUs. Modern GPU memory controllers are optimized for 4-byte (32-bit) or 8-byte (64-bit) transactions, making the single-byte memory accesses required for bin ID retrieval suboptimal for hardware utilization.

To improve memory efficiency, we propose a "bin packing" technique: multiple 1-byte bin IDs are packed into a 4-byte integer. This allows fetching multiple bin IDs in a single memory transaction, reducing bandwidth usage and improving cache locality. Threads within a warp extract their respective bin ID using lightweight bit operations (shifts and masks). With proper memory layout, shared memory bank conflicts are also minimized.

Bin packing involves two steps: *i) Packing* — multiple bin IDs are combined during preprocessing; *ii) Unpacking* — threads extract their bin ID during histogram construction. This optimization notably reduces memory overhead and accelerates the most compute-intensive stage of GBDT-MO training.

*3.4.2 Multi-GPU execution for training and inference.* To further accelerate GBDT-MO, our system supports multi-GPU execution on a single machine. During training, we apply feature-level parallelism by partitioning feature columns across GPUs. Each GPU constructs histograms for its assigned features using local data and independently evaluates splits. Partial histograms are then aggregated via CUDA-aware collective operations to form a global view for split selection. Since only summary statistics are exchanged, this method scales well with the number of GPUs and maintains low communication overhead.

Inference in GBDT-MO is tightly integrated with training, as prediction results are required to compute gradients ($g$) and second-order derivatives ($h$) for new trees. Our implementation supports both instance-level and tree-level parallelism. In instance-level parallelism, each GPU thread processes a different instance, while in tree-level parallelism, predictions across trees can be computed concurrently. The prediction algorithm recursively traverses the tree structure for each instance, using the split conditions to reach the corresponding leaf node. These optimizations ensure efficient training and inference at scale without compromising the predictive performance of the model.

# 4 EXPERIMENTS

In this section, we present an experimental analysis to evaluate the efficiency of our system compared with existing GBDT systems. Through a series of experiments, we explore and discuss the scalability and predictive accuracy of our system across diverse configurations.

## 4.1 Experimental setup

**Implementation details.** Our system was implemented in C++ and CUDA. The program was compiled with the -O3 optimization level flag. The experiments were conducted on a workstation with 1024 GB of main memory and 8 NVIDIA RTX 4090 GPUs.

**Parameter settings.** Unless otherwise specified, we use the following default parameter settings for all experiments: number of trees = 100, maximum tree depth = 7, learning rate = 1, minimum number of instances in a node = 20, and maximum number of bins = 256. These parameters were chosen based on preliminary experiments and are consistent with common settings in the literature. For all compared methods, we use their recommended default parameters when not specified otherwise.

**Datasets.** To ensure a comprehensive evaluation, we employ four real-world datasets, identical to the ones used in GBDT-MO [34] and SketchBoost [13]. These datasets cover different domains, offering diverse challenges to the GBDT models. Detailed information about these datasets is presented in Table 1. For datasets that already include a test set, no additional preprocessing is performed. For the rest, we randomly selected 20% of the training instances from the original dataset to create a test set.

**Table 1: Detailed information of datasets**

| Dataset | #instances | #features | #outputs | task |
|---|---|---|---|---|
| Otto | 61,878 | 93 | 9 | multiclass |
| SF-Crime | 878,049 | 10 | 39 | multiclass |
| Helena | 65,196 | 27 | 100 | multiclass |
| Caltech101 | 6,073 | 324 | 101 | multiclass |
| MNIST | 50,000 | 784 | 10 | multiclass |
| MINIST-IN | 50,000 | 200 | 24 | multiregress |
| RF1 | 9,125 | 61 | 16 | multiregress |
| Delicious | 16,105 | 500 | 983 | multilabel |
| NUS-WIDE | 161,789 | 128 | 81 | multilabel |

**Baselines.** Our proposed solution is compared against a mix of single-output and multi-output GBDT models. For single-output tasks, we benchmarked against **XGBoost** [5], **LightGBM** [16], and **CatBoost** [7]. These models are currently considered state-of-the-art and are capable of running on both multi-core CPUs and GPUs. For multi-output tasks, we benchmark against **SketchBoost** [13] and **GBDT-MO** [34]. SketchBoost is a GPU-accelerated multi-output GBDT system that has optimized the split-finding step, making it one of the most efficient multi-output GBDT systems currently available. **GBDT-MO**, implemented in C++, leverages parallel computing to speed up computations. It provides a useful benchmark to assess the effectiveness of GPU acceleration in our system. We used both CPU versions: *mo-full* which is GBDT-MO using dense representation ("mo-fu" for short); *mo-sparse* which is GBDT-MO with sparse representation ("mo-sp" for short).

## 4.2 Overall comparison

In this set of experiments, we comprehensively evaluate various GBDT systems on both CPU and GPU platforms, focusing on training efficiency and predictive performance. The compared systems include CPU-based implementations (*mo-fu* and *mo-sp* [34]), as well as GPU-accelerated baselines such as **CatBoost** [7], **LightGBM** [16], **XGBoost** [5], and **SketchBoost** [13].

*4.2.1 Training efficiency.* Table 4 and Table 2 report the training times on CPU and GPU platforms. Our method delivers substantial speedups over CPU-based baselines. For example, on *Caltech101*, *mo-sp* takes 1,154.88 seconds, while our method finishes in just 6.16 seconds—a speedup of 187×. Similar improvements are observed on *NUS-WIDE* (191×) and *MNIST-IN* (61×), clearly demonstrating the advantage of GPU acceleration. Compared to other GPU-based systems, our method consistently achieves the fastest training time across all datasets. On *MNIST-IN*, our method takes only 3.28 seconds, significantly outperforming *xgboost* (21.08 s) and *sk-boost* (26.61 s). Even on challenging datasets like *Delicious*, our system completes training in 17.79 seconds, much faster than *catboost* (135.40 s), *xgboost* (116.96 s), and *lightgbm* (610.30 s). In addition, with dual GPUs, the training time is further reduced. For example, on *MNIST*, it decreases from 5.04 seconds to 2.92 seconds, showing good scalability of our system with multiple GPUs.

**Table 2: Training time (seconds) comparison on single GPU and dual GPUs.**

| GPU | Dataset | catboost | lightgbm | xgboost | sk-boost | ours |
|---|---|---|---|---|---|---|
| Single GPU | MNIST | 20.13 | 42.88 | 16.51 | 28.61 | **5.04** |
| | Caltech101 | 21.55 | 32.54 | 18.31 | 28.61 | **6.16** |
| | MNIST-IN | 5.54 | 74.27 | 21.08 | 26.61 | **3.28** |
| | NUS-WIDE | 79.17 | 174.81 | 34.48 | 43.88 | **3.91** |
| | Otto | 1.78 | 34.24 | 1.28 | 22.58 | **0.22** |
| | SF-Crime | 15.08 | 18.06 | 17.51 | 32.57 | **2.07** |
| | Helena | 4.67 | 39.24 | 8.63 | 4.09 | **1.69** |
| | RF1 | 2.71 | 9.53 | 12.95 | 21.76 | **0.43** |
| | Delicious | 135.40 | 610.30 | 116.96 | 302.93 | **17.79** |
| Dual GPUs | MNIST | 8.31 | 42.26 | 4.59 | 7.69 | **2.92** |
| | Caltech101 | 9.70 | 33.22 | 6.95 | 16.31 | **3.24** |
| | MNIST-IN | 4.56 | 57.92 | 9.86 | 5.88 | **2.04** |
| | NUS-WIDE | 75.29 | 124.41 | 24.76 | 23.45 | **8.79** |
| | Otto | 1.33 | 11.19 | 1.91 | 11.40 | **0.91** |
| | SF-Crime | 3.58 | 24.18 | 9.45 | 12.16 | **3.78** |
| | Helena | 4.53 | 40.37 | 8.76 | 4.12 | **2.14** |
| | RF1 | 2.57 | 1.05 | 1.41 | 1.13 | **0.63** |
| | Delicious | 133.31 | 794.65 | 107.33 | 286.26 | **11.27** |

**Table 3: Testing accuracy or RMSE on GPU-based methods**

| Dataset | catboost | lightgbm | xgboost | sk-boost | ours |
|---|---|---|---|---|---|
| MNIST | 95.98 | 97.57 | 96.94 | 96.26 | 96.25 |
| Caltech101 | 51.11 | 55.38 | 44.44 | 51.36 | 49.31 |
| MNIST-IN | 1.67 | 0.31 | 0.36 | 0.27 | 0.28 |
| NUS-WIDE | 7.49 | 15.04 | 6.78 | 6.78 | 6.80 |
| Otto | 0.77 | 0.77 | 0.82 | 0.74 | 0.80 |
| SF-Crime | 0.16 | 0.17 | 0.17 | 0.16 | 0.21 |
| Helena | 0.22 | 0.23 | 0.23 | 0.22 | 0.23 |
| RF1 | 3.87 | 0.26 | 2.94 | 2.5 | 2.96 |
| Delicious | 0.07 | 0.02 | 0.08 | 0.07 | 0.13 |

**Table 4: Training time (seconds) and testing accuracy/RMSE on CPU-based methods vs. our system**

| Dataset | Training time (s) | | | Speedup | Accuracy / RMSE | | |
|---|---|---|---|---|---|---|---|
| | mo-fu | mo-sp | ours | vs mo-sp | mo-fu | mo-sp | ours |
| MNIST | 202.90 | 258.81 | **5.04** | **51.3**× | 96.69 | 96.25 | 96.25 |
| Caltech101 | 669.84 | 1,154.88 | **6.16** | **187.4**× | 49.38 | 48.72 | 49.31 |
| MNIST-IN | 149.36 | 200.03 | **3.28** | **61.0**× | 0.28 | 0.29 | 0.28 |
| NUS-WIDE | 401.30 | 747.37 | **3.91** | **191.2**× | 13.21 | 13.21 | 6.80 |

*4.2.2 Predictive performance.* Tables 3 and 4 report predictive performance, measured by accuracy or RMSE depending on task types. Experimental results show that our system achieves competitive or superior performance across all tasks. On classification datasets such as MNIST and Otto, our accuracy reaches 96.25% and 0.80, respectively—on par with *xgboost* and *catboost*. For regression and multi-output tasks, such as MNIST-IN and RF1, our RMSE of 0.28 and 2.96 matches or exceeds the best-performing baselines. On NUS-WIDE, our RMSE (6.80) is nearly identical to the best GPU methods (*sk-boost* and *xgboost*, both 6.78).

## 4.3 Sensitive study

Here, we aim to assess the scalability of our proposed system through a comprehensive evaluation of training times across various factors on RTX 3090, including different numbers of trees, tree depths, histogram building methods, and a range of class counts.

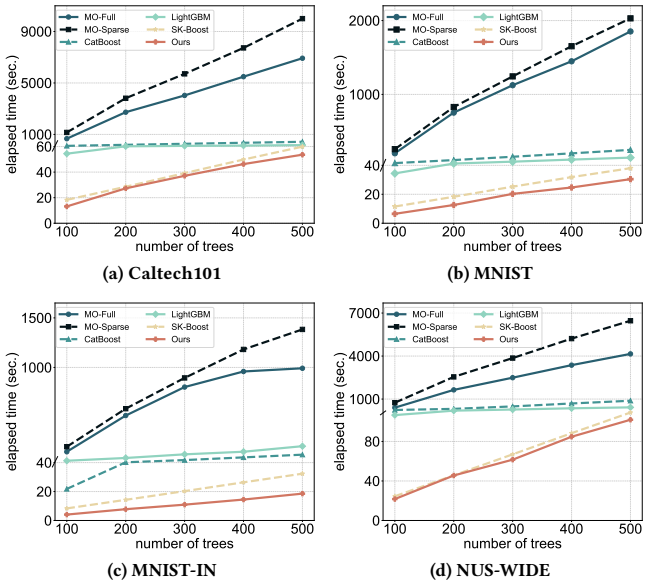



**Figure 5: Training time (in seconds) varies as the number of trees (#*trees*) is adjusted across four distinct datasets.**

*4.3.1 Scalability with respect to the number of trees.* Figure 5 illustrates the training time as the number of trees increases from 100 to 500 across four representative datasets. As expected, training time grows with the number of trees for all methods. However, the growth rate and absolute runtime differ significantly across systems. CPU-based implementations (*mo-fu* and *mo-sp*) consistently incur the highest cost, with training times increasing sharply—especially on large datasets like Caltech101 and NUS-WIDE. In contrast, GPU-based methods show substantially lower runtimes. Among them, our approach consistently achieves the fastest training time across all tree counts and datasets. For instance, even at 500 trees, our method remains below 100 seconds on all datasets, while *sk-boost*, *catboost*, and *lightgbm* incur noticeably higher costs. Moreover, our system exhibits near-linear scalability with respect to the number of trees, indicating efficient GPU resource utilization and minimal overhead accumulation as model size grows. These results further validate the efficiency and scalability of our implementation for large-scale GBDT-MO training.

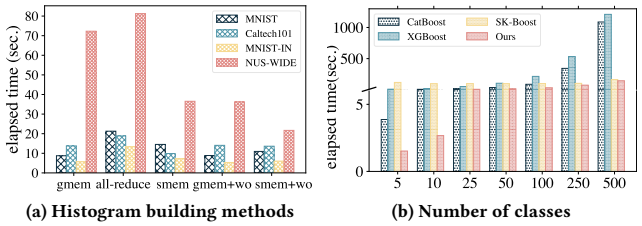
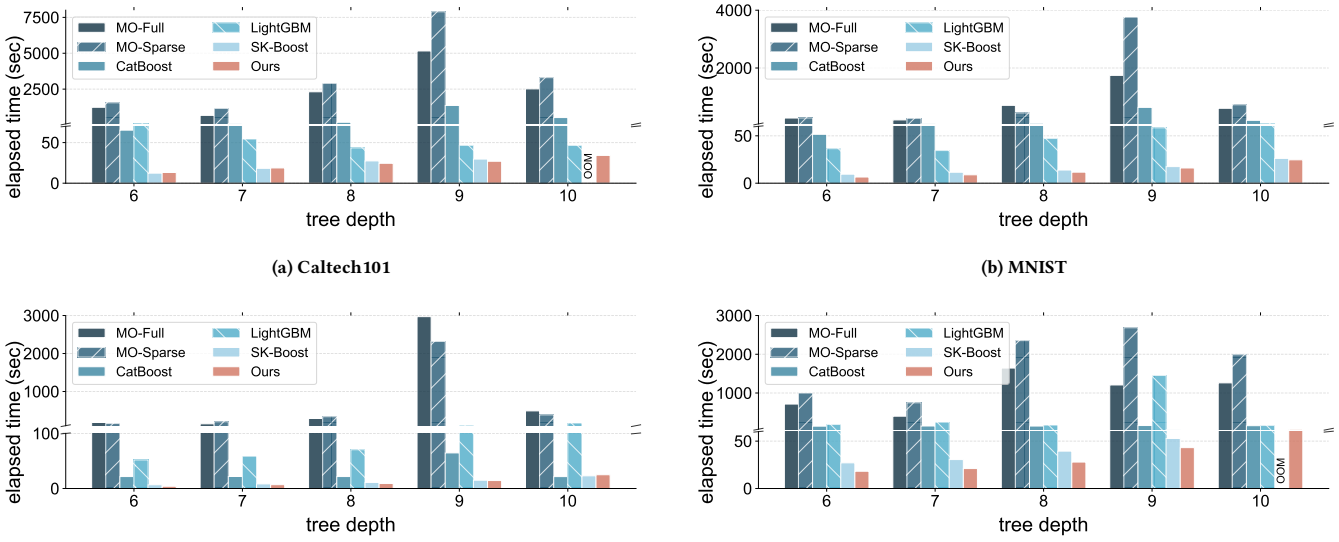



**Figure 6: Effects of different histogram building methods and the number of classes on training time.**

*4.3.2 Effect of histogram building methods.* Figure 6a compares the training time of five histogram building strategies across four datasets. Here, "gmem" and "smem" refer to global and shared memory methods, "all-reduce" denotes sort-and-reduce, and "+wo" indicates the use of warp-level optimization. Without warp optimization, "gmem" performs best on MNIST and MNIST-IN, while "smem" is more efficient on Caltech101 and NUS-WIDE. The sort-and-reduce method consistently incurs the highest cost, highlighting its overhead in sorting and reduction. With warp optimization, both "gmem+wo" and "smem+wo" show substantial improvements—especially on larger datasets like NUS-WIDE, where training time drops by nearly 50%. These results underscore the importance of tailoring histogram building strategies to dataset characteristics, and demonstrate that warp-level optimization is particularly effective at scale.

*4.3.3 Effect of the number of classes.* Figure 6b shows the training time of four methods on synthetic datasets with increasing numbers of classes (from 5 to 500). All experiments use 100 trees of depth 6, and datasets are generated using `scikit-learn`'s multi-class API. We observe that both *catboost* and *xgboost* exhibit steep increases in training time as the number of classes grows, suggesting scalability limitations in multi-class settings. In contrast, *sk-boost* remains relatively stable, but at a consistently higher runtime. Our method shows a moderate increase in training time with class count, but remains the fastest across all settings—demonstrating better scalability to large-output tasks. Additionally, we find that *sk-boost* is significantly more sensitive to class count and tree depth in terms of memory usage, while our method remains stable, further confirming its robustness.

*4.3.4 Scalability with respect to tree depth.* Figure 7 shows the impact of increasing tree depth on training time. As expected, deeper trees lead to higher computation costs. CPU-based methods (*mo-fu* and *mo-sp*) are consistently the slowest and often run out of memory at greater depths. GPU-based methods handle deeper trees more efficiently, and our approach achieves the lowest training time across all depths and avoids out-of-memory failures mostly. This confirms the scalability and robustness of our method under increasing model complexity.




(a) Caltech101 (b) MNIST

(c) MNIST-IN (d) NUS-WIDE

**Figure 7: Training time (in seconds) varies as the depth of trees (*depth*) is adjusted across four distinct datasets.**

## 5 RELATED WORK

In this section, we discuss the literature related to multi-output models, GBDTs with multi-outputs (GBDT-MO), and the acceleration techniques for these models.

**Multi-output models.** Multi-output learning is an established area of machine learning that focuses on the simultaneous prediction of multiple target variables [15]. These models are particularly effective in contexts where output variables are interdependent or correlated, such as in multi-task learning [26], multi-label classification [3], and multi-output regression [25]. Over the years, the field has seen the development of a variety of multi-output models, such as multi-output Support Vector Machines (SVMs) [33], multi-output neural networks [21], and multi-output random forests [24]. Each of these models presents unique advantages, with SVMs providing a strong theoretical foundation, neural networks offering high expressive power, and random forests ensuring robustness and interpretability. However, the increasing complexity and scale of multi-output tasks are presenting new challenges for these models in terms of computational efficiency and scalability.

**Multi-output GBDTs.** The multi-output GBDT (GBDT-MO) is an extension of the traditional GBDT algorithm that is capable of managing multiple outputs, capturing intricate dependencies between input features and multiple target variables [34]. The potential of GBDT-MO has been demonstrated in various tasks including image segmentation [17], multi-label classification, and multi-task learning. In recent years, enhancements to GBDT-MO have been proposed, such as output feature sharing and correlation-guided tree construction [31]. These innovations have further improved the predictive performance and interpretability of GBDT-MO. Nonetheless, the computational burden associated with training these models, especially for large-scale problems, remains a significant hurdle.

**Acceleration for GBDTs and multi-output models.** Given the computational complexity of GBDT-MO and multi-output models, researchers have explored different acceleration techniques. Some studies have focused on parallel and distributed computing for multi-output models, such as distributed multi-output SVMs [19] and parallel multi-output neural networks [9]. These approaches have shown great potential in large-scale settings but are often constrained by communication overheads and synchronization issues. On the other hand, Graphics Processing Units (GPUs) have been adopted to accelerate single-output GBDT training algorithms. A variety of systems have been proposed to support efficient GBDT training, such as XGBoost [5], LightGBM [16, 32], CatBoost [7], DimBoost [14], and ThunderGBM [28]. All of these existing systems except DimBoost support GBDT training on GPUs. [22] proposes a new GBDT learning algorithm for high dimensional outputs. [2] developed parallel decision trees for stream data using approximation. A more recent study investigates the performance of the different GBDT systems [10]. Except for GBDT systems for single output, SketchBoost [13] utilizes GPUs to accelerate the training of multiple-output GBDTs. Besides, approximate strategies for gradient reduction are proposed to further speed up the training process. However, the GBDT training is memory bound so that the efficiency improvement is limited due to irregular memory accesses [27]. There are also issues with histogram building in the training of GBDT-MO pending to be solved. For instance, the memory consumption of histogram building of GBDT-MO is a magnitude larger than that of GBDT-SO.

Based on the aforementioned, this work proposes a GBDT-MO system, which leverages the parallelism of GPUs to speed up the training process. We also optimize the histogram building process in GBDT-MO training. We build upon the advancements in GPU-accelerated single-output GBDTs and extend them to handle

multiple outputs efficiently. We evaluate our proposed solution on various real-world datasets and compare it with the state-of-the-art methods, demonstrating the effectiveness of our proposed solution.

## 6 CONCLUSION

Multi-output learning has attracted increasing attention due to its ability to handle complex dependencies between input features and multiple outputs. Among various multi-output models, the multi-output Gradient Boosted Decision Tree (GBDT-MO) has demonstrated remarkable performance and interpretability in a wide range of applications. However, the computational complexity of GBDT-MO training poses challenges in large-scale datasets. In this paper, we have proposed a GPU-accelerated GBDT-MO training system that significantly speeds up the training process while maintaining competitive model performance. Our system leverages the highly parallel architecture of GPUs to efficiently compute gradient updates and construct decision trees, enabling scalable and efficient multi-output GBDT training. Extensive experiments across diverse datasets show that our system achieves speedup over 30× on CPU-based baselines and up to 170× acceleration compared with leading GPU-based methods, while delivering comparable predictive performance. These results confirm the scalability and effectiveness of our system for practical, large-scale multi-output learning scenarios.

## ACKNOWLEDGMENTS

This work is supported by the Guangzhou Industrial Information and Intelligent Key Laboratory Project (No. 2024A03J0628). It is also funded by the NSFC Project (No. 62306256) and the Natural Science Foundation of Guangdong Province (No. 2025A1515010261).

## REFERENCES

[1] Stephan Allenspach, Jan A Hiss, and Gisbert Schneider. 2024. Neural multi-task learning in drug design. *Nature Machine Intelligence* 6, 2 (2024), 124–137.
[2] Yael Ben-Haim and Elad Tom-Tov. 2010. A streaming parallel decision tree algorithm. *Journal of Machine Learning Research (JMLR)* 11, Feb (2010), 849–872.
[3] Deblina Bhattacharjee, Tong Zhang, Sabine Süsstrunk, and Mathieu Salzmann. 2022. Mult: An end-to-end multitask learning transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 12031–12041.
[4] João Bravo. 2024. NRGBoost: Energy-Based Generative Boosted Trees. *arXiv preprint arXiv:2410.03535* (2024).
[5] Tianqi Chen and Carlos Guestrin. 2016. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data mining (SIGKDD)*. 785–794.
[6] Jyotikrishna Dass, Shang Wu, Huihong Shi, Chaojian Li, Zhifan Ye, Zhongfeng Wang, and Yingyan Lin. 2023. Vitality: Unifying low-rank and sparse approximation for vision transformer acceleration with a linear taylor attention. In *International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 415–428.
[7] Anna Veronika Dorogush, Vasily Ershov, and Andrey Gulin. 2018. CatBoost: gradient boosting with categorical features support. *arXiv preprint arXiv:1810.11363* (2018).
[8] Lei Du, Haifeng Song, Yingying Xu, and Songsong Dai. 2024. An Architecture as an Alternative to Gradient Boosted Decision Trees for Multiple Machine Learning Tasks. *Electronics* 13, 12 (2024), 2291.
[9] Martin Ferianc and Miguel Rodrigues. 2023. MIMMO: Multi-input massive multi-output neural network. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 4564–4569.
[10] Fangcheng Fu, Jiawei Jiang, Shaoxia Ying, and Bin Cui. 2019. An Experimental Evaluation of Large Scale GBDT Systems. *Very Large Data Bases (VLDB)* (2019).
[11] Yusheng Huang, Jiexing Qi, Xinbing Wang, and Zhouhan Lin. 2023. Asymmetric Polynomial Loss for Multi-Label Classification. In *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 1–5.
[12] Carlos Huertas. 2024. Gradient Boosting Trees and Large Language Models for Tabular Data Few-Shot Learning. *arXiv preprint arXiv:2411.04324* (2024).
[13] Leonid Iosipoi and Anton Vakhrushev. 2022. SketchBoost: Fast Gradient Boosted Decision Tree for Multioutput Problems. *arXiv preprint arXiv:2211.12858* (2022).
[14] Jiawei Jiang, Bin Cui, Ce Zhang, and Fangcheng Fu. 2018. DimBoost: Boosting gradient boosting decision tree to higher dimensions. In *International Conference on Management of Data (SIGMOD)*. ACM, 1363–1376.
[15] Karin Joan, Robyn Irawan, and Benny Yong. 2025. APPLICATION AND PERFORMANCE COMPARISON OF MULTI-OUTPUT MACHINE LEARNING FOR NUMERICAL-NUMERICAL AND NUMERICAL-CATEGORICAL OUTPUTS. *BAREKENG: Jurnal Ilmu Matematika dan Terapan* 19, 2 (2025), 1421–1432.
[16] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. LightGBM: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems (NeurIPS)*. 3149–3157.
[17] Chenxin Li, Xinyu Liu, Wuyang Li, Cheng Wang, Hengyu Liu, Yifan Liu, Zhen Chen, and Yixuan Yuan. 2025. U-kan makes strong backbone for medical image segmentation and generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 39. 4652–4660.
[18] Chengliang Liu, Jinlong Jia, Jie Wen, Yabo Liu, Xiaoling Luo, Chao Huang, and Yong Xu. 2024. Attention-induced embedding imputation for incomplete multi-view partial multi-label classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 38. 13864–13872.
[19] Peng Lu, Lin Ye, Wuzhi Zhong, Ying Qu, Bingxu Zhai, Yong Tang, and Yongning Zhao. 2020. A novel spatio-temporal wind power forecasting framework based on multi-output support vector machine and optimization strategy. *Journal of Cleaner Production* 254 (2020), 119993.
[20] Jiaqi Luo, Yuan Yuan, and Shixin Xu. 2025. Improving GBDT performance on imbalanced datasets: An empirical study of class-balanced loss functions. *Neurocomputing* 634 (2025), 129896.
[21] Joseph Shenouda, Rahul Parhi, Kangwook Lee, and Robert D Nowak. 2024. Variation spaces for multi-output neural networks: Insights on multi-task learning and network compression. *Journal of Machine Learning Research* 25, 231 (2024), 1–40.
[22] Si Si, Huan Zhang, S Sathiya Keerthi, Dhruv Mahajan, Inderjit S Dhillon, and Cho-Jui Hsieh. 2017. Gradient boosted decision trees for high dimensional sparse output. In *International conference on machine learning*. PMLR, 3182–3190.
[23] Gokul Swamy, Anoop Saladi, Arunita Das, and Shobhit Niranjan. 2024. PEMBOT: Pareto-ensembled multi-task boosted trees. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 5752–5761.
[24] Yin Wan, Ding Liu, and Jun-Chao Ren. 2024. A modeling method of wide random forest multi-output soft sensor with attention mechanism for quality prediction of complex industrial processes. *Advanced Engineering Informatics* 59 (2024), 102255.
[25] Andong Wang, Yuning Qiu, Mingyuan Bai, Zhong Jin, Guoxu Zhou, and Qibin Zhao. 2024. Generalized Tensor Decomposition for Understanding Multi-Output Regression under Combinatorial Shifts. *Advances in Neural Information Processing Systems* 37 (2024), 47559–47635.
[26] Dong Wang, Shaoguang Yan, Yunqing Xia, Kavé Salamatian, Weiwei Deng, and Qi Zhang. 2022. Learning Supplementary NLP Features for CTR Prediction in Sponsored Search. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 4010–4020.
[27] Zeyi Wen, Bingsheng He, Ramamohanarao Kotagiri, Shengliang Lu, and Jiashuai Shi. 2018. Efficient gradient boosted decision tree training on GPUs. In *International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 234–243.
[28] Zeyi Wen, Hanfeng Liu, Jiashuai Shi, Qinbin Li, Bingsheng He, and Jian Chen. 2020. ThunderGBM: Fast GBDTs and Random Forests on GPUs. *Journal of Machine Learning Research (JMLR)* 21, 108 (2020), 1–5.
[29] Jiahuan Yan, Jintai Chen, Qianxing Wang, Danny Z Chen, and Jian Wu. 2024. Team up GBDTs and DNNs: Advancing Efficient and Effective Tabular Prediction with Tree-hybrid MLPs. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 3679–3689.
[30] ZhenZhe Ying, Zhuoer Xu, Zhifeng Li, Weiqiang Wang, and Changhua Meng. 2022. MT-GBM: A Multi-Task Gradient Boosting Machine with Shared Decision Trees. *arXiv preprint arXiv:2201.06239* (2022).
[31] Xingbin Zhan, Shuaichao Zhang, Wai Yuen Szeto, and Xiqun Chen. 2020. Multi-step-ahead traffic speed forecasting using multi-output gradient boosting regression tree. *Journal of Intelligent Transportation Systems* 24, 2 (2020), 125–141.
[32] Huan Zhang, Si Si, and Cho-Jui Hsieh. 2017. GPU-acceleration for Large-scale Tree Boosting. *arXiv preprint arXiv:1706.08359* (2017).
[33] Xianxia Zhang, Gang Zhou, Hanyu Yuan, and Bing Wang. 2025. Online Three-Dimensional Fuzzy Multi-Output Support Vector Regression Learning Modeling for Complex Distributed Parameter Systems. *Applied Sciences* 15, 5 (2025), 2750. https://doi.org/10.3390/app15052750
[34] Zhendong Zhang and Cheolkon Jung. 2020. GBDT-MO: gradient-boosted decision trees for multiple outputs. *IEEE transactions on neural networks and learning systems* 32, 7 (2020), 3156–3167.
[35] Xin Zhou and Qiquan Ran. 2023. Optimization of Fracturing Parameters by Modified Genetic Algorithm in Shale Gas Reservoir. *Energies* 16, 6 (2023), 2868.