Visual Diagnostics of Parallel Performance in Training Large-Scale DNN Models

Yating Wei, Zhiyong Wang, Zhongwei Wang, Yong Dai, Gongchang Ou, Han Gao, Haitao Yang, Yue Wang,Caleb Chen Cao, Luoxuan Weng, Jiaying Lu, and Rongchen Zhu,Wei Chen

Abstract—Diagnosing the cluster-based performance of large-scale deep neural network (DNN) models during training is essential for improving training efficiency and reducing resource consumption. However, it remains challenging due to the incomprehensibility of the parallelization strategy and the sheer volume of complex data generated in the training processes. Prior works visually analyze performance profiles and timeline traces to identify anomalies from the perspective of individual devices in the cluster, which is not amenable for studying the root cause of anomalies. In this paper, we present a visual analytics approach that empowers analysts to visually explore the parallel training process of a DNN model and interactively diagnose the root cause of a performance issue. A set of design requirements is gathered through discussions with domain experts. We propose an enhanced execution flow of model operators for illustrating parallelization strategies within the computational graph layout. We design and implement an enhanced Marey's graph representation, which introduces the concept of time-span and a banded visual metaphor to convey training dynamics and help experts identify inefficient training processes. We also propose a visual aggregation technique to improve visualization efficiency. We evaluate our approach using case studies, a user study and expert interviews on two large-scale models run in a cluster, namely, the PanGu-α 13B model (40 layers), and the Resnet model (50 layers).

Index Terms—Visual Analysis, Deep Neural Network, Model Training, Parallel Performance

1 Introduction

In recent years, Deep Neural Networks (DNNs) have grown dramatically. Researchers and engineers have applied DNN models to solve their problems in various fields, including natural language processing (NLP) [1], [2], [3], image classification [4], and speech recognition [5]. Accordingly, sizes of DNN models have increased dramatically, making DNNs greatly computation- and storage-intensive. Thus, it is becoming popular to parallelize the training of DNN models in a distributed cluster. The most extensively used parallelization strategy is data parallelism (DP) [6]. It is used for scenarios where the bulk of training data exceeds the capacity of a single device. To solve this problem, DP places a duplicate of the entire DNN model on each device, permits training data to be divided into numerous shards and distributed across devices, and synchronizes model parameters across replicas at the end of an iteration to obtain a global model. Model parallelism (MP) [7] is another prevalent parallelization strategy. It is utilized when a model is too huge to fit in local memory. In this case, the model is split into different modules. Each module can be placed in a different training device. Depending on the split method, there are two types of MP, namely intra-layer MP (also known as Tensor Slicing) [8], [9] and inter-layer MP (also known as Pipeline Parallelism) [10], [11]. The choice of strategy directly affects the training efficiency. For the same model, the training time of different strategies may differ by several times. For example, ImageNet/ResNet-50 with a relatively large training dataset have been successfully trained in 74.7 seconds with DP [12]. However, MP is not suitable for this case. In general, every parallelization strategy has the potential to fail when the underlying model is too large. Diagnosing performance issues in a large-scale cluster is, however, not trivial. On the one hand, the usable resources in a cluster are always limited. On the other hand, the training behaviors can have a large negative impact on the diagnosis process.

For these reasons, there is a rising interest in visually comprehending and diagnosing the parallel training process of a DNN model, which has theoretical and practical benifits for deep learning experts. There are three major challenges. The first challenge is the understanding of the DNN models and parallelization strategies in the clusters. It is difficult to intuitively represent parallelization strategies and the execution logic within the computational graph of the underlying model. Especially in large-scale clusters, where each device holds a part of the entire computational graph, understanding the execution context of operators is difficult. The second challenge is to efficiently convey the massive quantity of profiling data generated during the parallel training of a DNN model. The profiling data from the distributed training process includes training dynamics such as memory usage, computation time, communication time, *etc.*, which is represented with a set of high-dimensional time series. To comprehensively understand the data and possible performance issues, multi-faceted patterns should be clearly conveyed, such as the temporal patterns of devices in a cluster in terms of multiple metrics. The third challenge is

- *Y. Wei, Z. Wang, Z. Wang, Y. Dai, L. Weng, J. Lu, R. Zhu and W. Chen are with The State Key Lab of CAD & CG, Zhejiang University, Hangzhou, Zhejiang 310058, China.*
  E-mail: {weiyating, zerowangzy, wzw09, daiyong, lukeweng, 3180103570, zrcrcz, chenvis} @zju.edu.cn
- *G. Ou, H. Gao, H. Yang, Y. Wang and C. Cao are with Distributed Data Lab, Huawei Technologies Co., Ltd., Shenzhen 518129, China.*
  E-mail: {ougongchang, gaohan19, yanghaitao1, wangyue53} @huawei.com, {caochen.hkust} @gmail.com
- *Wei Chen is the corresponding author.*

identifying the causes of the inefficiency of the training process. An performance issue can be caused by several root causes. There are various parallelization strategies, yielding different ways of identifying performance issues. Given the scale of DNN models and clusters, there is a need to clearly show the anomalous patterns of performance issues with a level-of-detail visualization.

A few studies have focused on visual analysis of parallel model training. Tensorboard displays the computational graph of a DNN model using the hierarchical structure from the perspective of a single device in the cluster [13]. Chrome trace viewer (chrome://tracing) provides rich analysis and visualization capabilities for performance profile and timeline tracing. It arranges model operators in the order of the execution sequence from a single-device execution perspective. All these works focuses on cases in a single device, and can not be directly applied to the analysis of a bottleneck's root cause.

To address these challenges, we have formulated the design requirements through cooperation with experts of MindSpore (co-authors of this paper). On the basis of the features of DNN computational graphs, we employ a directed acyclic graph (DAG) to describe it, where nodes indicate operators, and edges indicate the dataflow between nodes. To improve the comprehensibility of the DAG, we have built a force-directed graph layout by applying the execution sequence of operators to constrain the repulsive force against each node. Information of parallelization strategies is incorporated into the computational graph. We have provided edge configurations to hide edges that are not important for understanding the main logic, such as edges representing the data flow in the data preparation phase. Furthermore, we have automatically identified similar substructures in the computational graph and stack them, such as optimizers used to reduce the losses [14]. To effectively convey complicated performance metrics and their relationships, we have built a visual interface with multiple views. The visualization and interaction designs support the visual analysis of performance issues at three levels, namely cluster-level, device-level, and block-level, to present multi-faceted patterns. In particular, we have enhanced Marey's graph by incorporating the concept of time-span and a banded visual metaphor to facilitate the detection of inefficiencies in the training processes and relationships between the various DNN model operators. The communication view presents communication details from the cluster's perspective and thus facilitates the refinement of the root cause. Finally, Finally, we evaluate our work utilizing two case studies and one user study of real-world application scenarios and collect experts feedback.

In conclusion, the main contributions are as follows:

- We propose a visual analytics approach that assists experts in comprehending the parallel training process and interactively diagnosing the causes of training issues.
- We introduce the execution sequence into the computational graph layout to enhance the study of parallelization strategies.
- We enhance Marey's graph by incorporating the concept of time-span and a banded visual metaphor to convey training dynamics and help experts identify the causes of inefficiency.

The remaining sections of this paper are structured as follows. Section 2 discusses the related work. Section 3 introduces the parallelization strategies for DNNs and performance profiles. In Section 4, requirement analysis and system overview are elaborated. Section 5 describes the visualization design. Section 6 evaluates our work from different perspectives. In section 7, we discuss our work from multiple perspectives. Section 8 concludes the paper.

2 Related Work

2.1 Visualization of DNN Computational Graph

Several studies [15], [16], [17] demonstrate the mechanism of DNNs from a network-centric perspective. Directed acyclic graphs (DAGs) are often employed to illustrate the computational graph. Operators are indicated by nodes and connected by links [18]. CNNVis [19] utilizes a number of clustering techniques to effectively represent a deep model and decrease the visual clutter generated by a large number of nodes and links. TensorBoard [20] provides a scalable graph view of a DNN's computation graph. TensorBoard [20] presents the computational graph of a DNN with a scalable graph visualization. It aggregates nodes into high-level blocks. High-degree nodes are not included to reduce visual clutter. The above approaches facilitate experts in better understanding the network structure and reducing visual clutter. However, sometimes experts might not understand the execution logic of operators. For example, collapsed nodes lead to the loss of the execution context and make it inconvenient to recognize predecessors and successors of operators. High-degree nodes in TensorBoard [20] may result in interruptions in the data flow. In this paper, we introduce execution sequence into the computaional graph layout.

2.2 Visualization of Cluster Performance

Descriptions of distributed cluster performance are complex. Description data could be textual, hierarchical, and temporal, *etc.* Researchers have proposed several schemes for cluster performance data visualization. La VALSE [21] visualizes tens of millions of RAS (reliability, availability, and serviceability) logs by using a scalable design to facilitate users in efficiently identifying the causes of failure events. The performance of a cluster is substantially determined by inter-device communication. Fujiwara et al. [22] offer a visual analysis system to assist users in comprehending diverse visual patterns of communication and identifying communication bottlenecks to improve the communication efficiency in parallel applications. To reveal the execution details of the operating system (such as CPU, network, and I/O), researches are conducted for jobs like anomaly identification. To evaluate the work load balancing, Xian et al. [23] propose a load scoring algorithm based on high-performance cluster data. The profile module of Cloud TPU [1] provides performance visualization from different scales.

2.3 Visualization of Temporal Data

A great deal of study has been conducted on temporal data visualization in recent years. Bach et al. [24] survey a variety of techniques and categorize them from a novel standpoint. They explain techniques as a series of actions performed on an imaginary space-time cube. Extraction, flattening, geometry transformation, and content transformation comprise these actions. Among temporal data visualization techniques, the event sequence visualization technique is the most relevant to our study. Marey's graph [25] is first created using space cutting to visualize train schedules in the 19th century. Inspired by this design, Palolm et al. [26] provide a visual analytic method for analyzing transportation timetables. ViDX [27] applies and extendeds Marey's graph to track and troubleshoot manufacturing assembly line performance. Lifelines [28] presents a visualization for patient medical records.

1. https://cloud.google.com/tpu/docs/cloud-tpu-tools

Sankey diagram has been widely used for the analysis of human mobility [29] and electronic health records [30], [31], [32]. In this paper, we extend Marey's graph to include the idea of time-span and a band visual metaphor for efficient identification of inefficiencies in the training processes.

3 Background

In this section, the strategy abstraction and several performance issues are introduced.

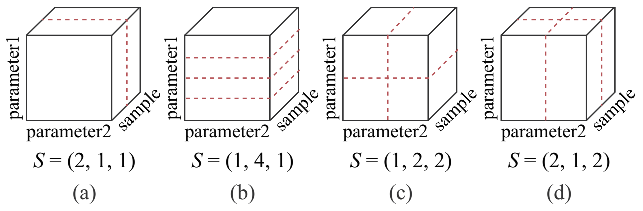


Fig. 1. The illustration of the data parallelism (a), the model parallelism (b) and two types of the hybrid parallelism (c-d). Each cube represents the output tensor of an operator with three parallelizable dimensions, i.e. $sample$, $parameter1$ and $parameter2$.

Strategy abstraction. The training of a DNN involves iterations of forward and backward computations. Each iteration of the training loop processes a subset of the input data and updates the model parameters. However, modern DNN models and the size of the training dataset are growing extremely large, resulting in longer training time and requiring more hardware resources. Therefore, parallel distributed training has been introduced. The fundamental parallelization strategies include data parallelism and model parallelism. For some complex models, experts usually use a hybrid of the fundamental parallelization strategies. Such hybrid parallelization strategies divide the output tensor of operators into different parallelization forms [33], which brings challenges to the strategy understanding during performance diagnosis. For simplicity, parallelism strategies can be defined in the same form. Given an operator op_i, parallelizable dimensions P_i represents all divisible dimensions in its output tensor. $P_i = \{s, p_1, p_2, ..., p_n\}$, where s indicates a sample dimension (i.e. the training data samples) and p_i indicates a parameter dimension. If the parameter dimension is partitioned, the model parameters will be splitted. Thus, the parallelism strategy $S = (n_s, n_{p_1}, n_{p_2}, ..., n_{p_n})$, where n indicates the number of divisions in the dimension. Figure 1 shows examples of parallelism strategies, where operators are partitioned over one dimension or multiple combined dimensions.

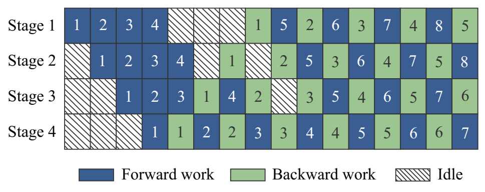


Fig. 2. The execution timeline for a pipeline with four stages, each running on one device. This illustration is proposed in PipeDream [34].

In particular, pipeline parallelism (PP) cannot be represented using the form S. PP splits the layers of the model, obtaining several consecutive stages. Different strategies S can be used in each stage. Experts typically focus on the execution timeline within and between stages during performance diagnosis, as shown in Figure 2. Numbers indicate minibatch IDs. The input stage executes four minibatches and propagates to the output stage. After the output stage executes the forward pass for the first minibatch, all stages begin to alternate between forward and backward passes for all minibatches. The latter stage depends on the execution results of the previous stage of the pipeline.

Performance issues. There are three main performance issues. (1) Long waiting time for collective communication. Collective communication is responsible for merging gradients or parameters resulting from parallel processing. Affected by various factors (such as computation amount and speed), device execution may be uneven, resulting in long wait times for collective communication. (2) Long waiting time between stages. When using pipeline parallelism, the previous stage sends parameters to the later stage through a peer-to-peer communication operator. If the previous stage does not finish executing, the later stage will keep waiting, making the idle time longer. (3) Communication link issues. Due to the different bandwidths of different communication links, the collective communication time may be longer.

4 System Design

4.1 Requirement Analysis

The general goal is to efficiently and accurately diagnose root causes of performance issues in parallel training of large-scale DNN models. To this end, we collaborated with 7 experts (DL engineers with 8-10 years of experience in model performance diagnostics, average experience: 8.5 years) from the Mindspore team for 6 months. At the beginning of the collaboration, we reviewed literature on mechanisms of parallelization strategies [6], [7], [10], [11] and visual analysis of cluster performance [21], [22], [23], and conducted a 2-hour discussion with experts every week. The discussion content revolves around experts' daily model performance issue analysis, including the understanding of parallelization strategies, key factors affecting performance (such as bandwidth, memory, etc.), current root cause location methods and processes, etc. Based on discussions, we refined the general goal and correspondingly derived the following requirements to guide the system's design.

R1: Facilitate the comprehension of cluster parallelization training strategy. The system should convey the parallelization strategy to the user through a vivid visual representation, so that the user may readily determine if the present training strategy needs to be optimized. Currently, the process of experts reviewing strategies is indirect and inefficient. They need to obtain the specific strategy from the intermediate representation (IR) file which is the code used internally by a compiler to represent source code, and then manually draw the operator execution flow to view the parallelization strategy of operators. If the model uses a complex hybrid parallelization strategy, the diagnosis process will take a long time. Efficient and easy-to-use visualizations for all kinds of parallelization strategies are urgently needed.

R2: Convey complicated performance metrics and their relationships. Considering that the cluster training performance would be influenced by many factors, such as collective communication latency, communication link bandwidth, memory usage, etc., the system should create easy-to-understand designs to connect different factors of performance data.

- **R2.1: Provide an overview of distributed training process over time.** Experts agreed that it is required for diagnostics to provide an overview of the execution times on each device.

The distributed training process over time can help experts identify the interesting training step.

- **R2.2: Provide a multi-level visualization for performance data exploration.** Experts need to explore performance data from different levels, such as cluster-level and device-level, to discover multi-faceted patterns. The latest performance analysis tool, Cloud TPU, displays performance metrics as independent charts. In the process of analysis, experts need to jump between different charts to establish a level of analysis in the brain.

R3: Connecting the performance data with the computational graph. Experts need performance data to check when performance issues occur by comparing with empirical thresholds. They also need to examine the execution flow via computational graph to identify in which the strategy need to be optimized. For example, the input of operator *A* is small, but it is allocated to 4 devices for execution, resulting in high communication cost during data merging. In addition, the experts said that they want to connect performance data to the computational graph throughout the diagnosing process. This connection might allow them to effectively identify when, where, and why the anomalies occur. However, existing tools do not provide this function. They need to manually draw the execution flow, and use event tracking to obtain the operator execution timestamp.

4.2 System Overview

As shown in Figure 3, the system's workflow consists of three key modules: a preprocessor module, a analyzer module, and a visualizer module.

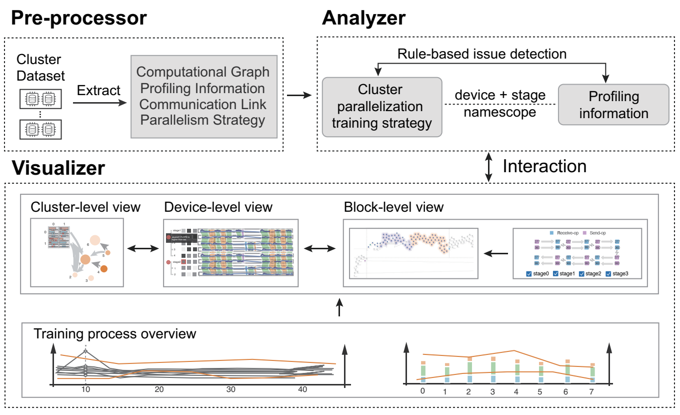


Fig. 3. Overview of the system workflow.

Given cluster training data for a large model, our Pre-processor extracts data for cluster diagnosis, including computational graph, profiling information, communication link, and parallelism strategy. The processed data are then send to the Analyzer, which establishes connections for different types of data, and computes summary statistics used in issue detection. Specifically, the Analyzer first establishes the relationship between the four processed data according to the attributes ”device” and ”the namescope of the operator”. Such relationship is mainly between the cluster parallelization training strategy and the profiling information. If the model training uses a pipeline parallelism strategy, the attribute ”stage” should also be considered when establishing the data connection. The Analyzer then calculates summary statistics such as the average computational amount of a stage, the ratio of the communication duration, the bandwidth of a communication link, the memory consumption of an device, etc. Thereafter, the Analyzer filters these summary statistics based on the experience thresholds of experts' daily diagnosis to realize preliminary issue detection of training data and provide guidances for diagnosis.

The Visualizer module allows the user to explore the cluster training dynamics at the cluster-level, the device-level, and the block-level, which corresponds to experts' typical diagnostic procedure. The overall temporal dynamics of all devices in the training cluster are shown in an Overview (**R2.1**). The user can pick the desired training step by clicking on the curve. After selecting the desired step, the Visualizer presents the training dynamics at three different levels (**R2.2**). At the cluster-level, the cluster topology view shows cluster communication. At the device-level, the profiling view shows the execution of the operator on each device. At the block-level, the parallelism strategy view uses a computational graph as a medium to present parallelism strategies, which is convenient for users to view the strategy of the operators of interest (**R1**). All views are dynamically coordinated through interactive linking, allowing seamless exploration of cluster training dynamics data from different perspectives (**R3**).

5 Visualization

Based on the design requirements, we explain the design decisions and interaction designs of our system. As shown in Figure 4, our system contains two interactively coordinated views to facilitate the exploration of the distributed training data.

TABLE 1
Terminology used in our paper.

Term	Explanation
operator	A mathematical computation in the model layer.
namescope	Used to group operators based on computational logic.
tensor	The input data or output data of an operator.
operator block	Operators assigned to a device in the cluster.
minibatch	A slice of training data in pipeline parallelism.

5.1 The Parallelism Strategy View

According to the analytical habits of experts, the parallelism strategy view (Figure 4(A)) employs a computational graph to present parallelization strategy. The left panel (Figure 4(A1)) is used for computational graph configuration. The component on the top is a namescope selector for the graph. When users select a namescope, the corresponding operators in the namescope will be highlighted in the graph. The checkboxes in the middle are used to control the display of different types of edges in the computational graph. The illustration of the training stages on the bottom shows the data flow between stages through several send and receive operators.

5.1.1 Computational Graph Layout

A computational graph is a directed graph which expresses mathematical expressions in language of graph theory. Nodes of the graph represent operators, and edges indicate input tensors and output tensors between them. Several deep learning frameworks, including TensorFlow [35] and MindSpore [2], utilize namescopes to produce a hierarchically structured graph representation. Sugiyama

2. https://github.com/mindspore-ai/mindspore

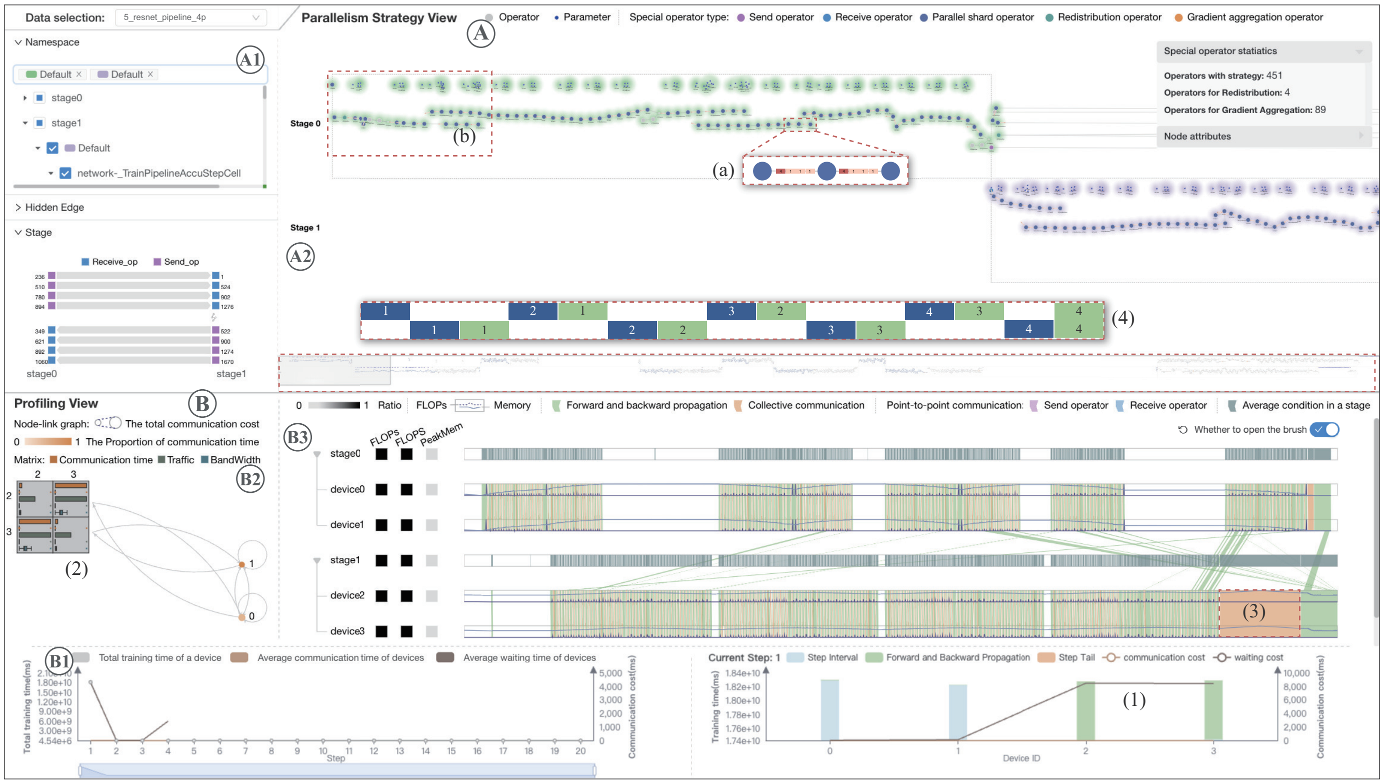

Data selection: 5_resnet_pipeline_4p
Parallelism Strategy View
Operator
Parameter
Special operator type: Send operator Receive operator Parallel shard operator Redistribution operator Gradient aggregation operator
A
A1
A2
Namespace
Hidden Edge
Stage
stage0
stage1
Stage 0
Stage 1
(a)
(b)
(4)
Special operator statiatics
Operators with strategy: 451
Operators for Redistribution: 4
Operators for Gradient Aggregation: 89
Node attributes
Profiling View
B
B1
B2
B3
Node-link graph: The total communication cost
The Proportion of communication time
Matrix: Communication time Traffic BandWidth
(1)
(2)
(3)
Ratio FLOPs Memory Forward and backward propagation Collective communication Point-to-point communication Send operator Receive operator Average condition in a stage
Whether to open the brush
FLOPs FLOPS PeakMem
device0
device1
device2
device3
Total training time of a device
Average communication time of devices
Average waiting time of devices
Step
Current Step: 1 Step Interval Forward and Backward Propagation Step Tail communication cost waiting cost
Training time(ms)
Communication cost(ms)
Device ID

Fig. 4. Our system contains two main views. (A)The parallelism strategy view presents parallelization strategies using a computational graph.(B) The profiling view shows complex performance data metrics and their relationships. The interactions between views facilitate the understanding and diagnosing the parallel training process.

Algorithm [36] is one commonly used method to draw the computational graph in a hierarchical layout. However, sometimes experts might not understand the execution logic of operators because nested namescopes make experts lose the execution context. To improve the comprehensibility of the execution logic in the computational graph, we improve a force-directed graph layout by applying the execution sequence of operators to constrain the repulsive force against each node. The specific implementation steps are as follows.

1. **Configure the type of edge.** Considering that the computational graph of a large-scale model contains a large number of edges, resulting in visual clutter and low rendering performance, we configure the type of edges for users to choose which edges can be hidden. Through discussions with experts, edges in a computational graph can be divided into primary logical edges (Figure 5(b)) and secondary logical edges (Figure 5(a)) executed by operators. The secondary edges mainly include large span edges between forward and backward propagation, etc. Currently we have configured seven types of edges.

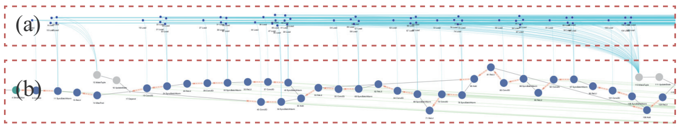


Fig. 5. The secondary logical edges (a) and the primary logical edges (b) of DNN model.

2. **Calculate the force between nodes and determine the position of nodes.** After model training, the operator execution sequence is generated. To obtain the final position of the nodes, we start by giving the nodes in the computational graph a random initial position, then traverse the execution sequence file and adjust the positions of the nodes in the graph accordingly. The adjustment details are as follows.

In the x-axis direction, suppose operator A is executed before operator B. The initial Euclidean distance and the expected distance between A and B are denoted as d and a, respectively. The difference between the actual distance and the expected distance Δd is $(d-a)$. If $\Delta d > 0$, the distance between A and B is larger than expected, and we need to reduce repulsion to pull B closer to A. If $\Delta d < 0$ and $|\Delta d| < d$, the distance between A and B is smaller than expected, and we need to increase repulsion to push B apart to increase the distance between A and B. If $\Delta d < 0$ and $|\Delta d| > d$, the B is drawn in front of A, and we need to pull B behind A to gurantee the correct execution order.

In the y-axis direction, we keep nodes as close to the central axis as possible to make nodes more compact. In our pre-experiment, we discover that the computational network has a substantial number of special nodes, which do not affect the main logic, such as *load* and *updateState* nodes used for data preparation. We treat them as subordinate nodes of other primary logical nodes. If we draw these nodes together with the main logic node, it will bring visual clutter and reduce the understandability of the execution logic. Therefore, we set their ordinate values above the main logic. In addition, there is a special topology that two nodes point to a successor node at the same time. In such case, the two nodes will coincide if nothing is done. Thus, we conduct collision detection to separate the positions of the nodes.

3. **Chunk the computaional graph and recompute the positions.** In pipelined parallelism, communication between different stages relies on *Send_op* and *Receive_op* operators. For example, after the execution of stage1, the parameters are transmitted to stage2 through a *Send_op* operator, and then in stage2, the parameters are received by a *Receive_op* operator. In order to vividly present the execution logic of the computational graph under the pipeline parallelism strategy, we take the *Send_op* and *Receive_op* operators as the boundary to divide the computational graph into small blocks, and recalculate the positions of these blocks according to the execution order of the paired *Send_op* and *Receive_op* operators. After recalculating the position of the stages, the user can clearly see the dependencies of the stage execution. As shown in Figure 2, the operator block 2 in stage2 depends on the execution of stage1's operator block 2 and stage2's operator block 1.

5.1.2 *Visual Encoding*

Considering the parallelization strategy $S = (n_s, n_{p_1}, ..., n_{p_n})$, using a 1D vector to show how the output tensor of an operator op is partitioned, we employ a set of rectangles to represent the parallelization of op, as shown in Figure 4(a). Each rectangle represents a parallelizable dimension in output tensor. In each rectangle, the specific number represents the number of partitions of the dimension, while the background color denotes the same information to provide users with intuitive perception. We place the set of rectangles along the edge, in line with the user's comprehension of the data flow. Using this representation, almost all parallelization strategies except pipeline parallelism can be presented. The pipeline parallelism is presented through vertically arranged blocks. Before finally deciding on using a set of rectangles, we considered an alternative visual design. Inspired by the strategy illustrations in Deepspeed [3], we designed different glyphs for different strategies and placed them on the corresponding operators. However, in the face of diverse hybrid parallelization strategies, this design is poorly scalable and prone to visual clutter.

To show the namescopes information, a circle blur background is added below the node (Figure 4(b)). Different namescopes are distinguished by color. Considering that there are many namescopes, it would be visually cluttered to use colors to distinguish all namescopes. Thus, we only present the user-selected namescopes, and the number of the selected namescope should be less than ten. Users can select the namescopes of interest in the configuration panel on the left. Before finally deciding on using the circle blur background, we have considered two alternative visual encodings. The first one is to use the traditional aggregation method in which the nodes in a namescope are aggregated into one group node [20]. However, such a design affects the perception of the operator's execution logic. Another alternative is to use the color of nodes to distinguish namescopes. However, the color is hard to identify when the graph is zoomed out.

There are three kinds of operators related to the parallelization strategy, namely, the operator whose output tensor is partitioned, the operator used to redistribute the tensors, and the operator used for communication. They are denoted by colors. The statistics of these operators are listed in the upper right panel, providing an overview of the parallelization strategy. When users select a node, the detailed attributes are shown in the lower right panel. Combined with the shape of the operator's output tensor (shape) in the attribute panel and the parallelization strategy of the operator in the computational graph, we can analyze whether the parallelization strategy is reasonable. For example, if the shape is [64,8] and the strategy is [4,1], indicating that the 64×8 tensor is divided into four pieces. This division is unreasonable because each piece is too small.

Since the computational graph is too long and narrow, a minimap is provided at the bottom to facilitate the exploration. Since the nodes are not aggregated by namescopes, the number of nodes is large, affecting the rendering performance. It is common that the computational graph contains similar substructures, such as optimizers for the reduction of loss. We automatically identify similar substructures in the computational graph and stack them, as shown in Figure 4.

3. https://www.microsoft.com/en-us/research/blog/deepspeed-extreme-scale-model-training-for-everyone/

5.2 The Profiling View

The profiling view (Figure 4(B)) contains a training overview, a cluster topology view and an extened Marey's graph to demonstrate performance data at different scales.

5.2.1 *The Training Overview*

The training overview (Figure 4(B1)) aims at revealing an anomaly overview of the important tracking metrics, including total training time and communication cost of each device, over training step. The overview demonstrates the whole profiling data in a temporal context. The dual-axis linechart on the left shows the time comsumption in terms of different metrics at every step. The left y-axis denotes the total training time of each device, while the right y-axis denotes the average communication cost of devices, including communication duration and waiting duration. A categorical color scheme is used to denote different metrics. When users select a step, the stacked barchart and linechart on the right shows the detailed information of every device in this step. The stacked barchart denotes three parts of the training time, including step interval (i.e. data preparation), forward and backward propagation, and step tail (i.e. collective communication). The linechart overlapped on the stacked barchart shows the communication duration and waiting duration of each device.

5.2.2 *The Cluster Topology View*

Parallel model training performance is strongly reliant on the execution environment. The cluster topology and characteristics in the environment decide how fast data can be transmitted and facilitate the comprehension of training behavior. In the cluster topology view (Figure 4(B2)), we use a DAG to present the topology and its characteristics, including link type, and the rate of communication duration, of the selected training step. The nodes of the graph represent devices, while edges indicate communication between devices. The size of the nodes denotes the total communication cost, including communication duration and waiting duration. The color of the nodes denotes the proportion of communication duration in total communication cost.

When users click an edge or lasso-select the desired nodes, an adjacency matrix is drawn using the NodeTrix visualization technique [37]. It is convenient to display several metrics, namely, communication duration, taffic, and bandwidth of the link. A categorical color scheme is used to present different metrics. The rows and columns of a matrix indicate the source devices and the target devices, while cells indicate the communication links. If there is a communication link between two devices, the corresponding cell will present the link metric information using two types of chart. The bars show the sum of the communication duration and traffic of different communication operators on this link. The boxplots show the distribution of communication duration, traffic, and bandwidth of each communication operator using this link. The background color of each cell in the matrix is the same as the edges in the DAG, representing the link type.

5.2.3 Enhanced Marey's graph

The enhanced Marey's graph (Figure 4(B3)) is used for profiling data analysis. Directly using a Marey's graph would cause the device's execution time and idle time to be missed, affecting the effectiveness of anomaly analysis, we introduce the concept of time-span and a banded visual metaphor to enhance it. Because misaligned device timestamps would lead to visual clutter, especially when zooming in, we align device timestamps according to the semantic context of operator execution. To support smooth exploration, a visual aggregation technique is applied to the horizontal and vertical orientations of our enhanced Marey's graph.

Visual Encoding. Marey's graph is commonly used to analyze bus or train schedules. Stations are plotted on the y-axis, and x-axis denotes time. Each station corresponds to a time axis. Each polyline represents a bus or a train, with each time point indicating when it is scheduled to arrive at a station (Figure 6(a)). This visual encoding can be directly used to data profiling if each device in a cluster is considered as a station and the time at which an operator begins executing on each device is considered as the time in bus or train timetables (Figure 6(b)). This forms the execution-starting line which traces the parallel execution status of an operator in a cluster, which is similar to the extended Marey's graph in ViDX [27].

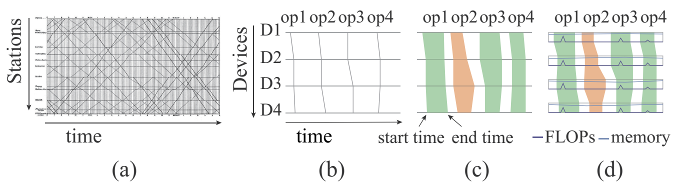


Fig. 6. The original Marey's graph and the design evolution when applied to profiling data. (a) The original Marey's graph used to analyze the bus/train timetables. (b) The directly adoption of Marey's graph to the profiling data. (c) The extention by introducing the concept of time-span, and showing an operation using a band. (d) The extra information encoding.

However, the completion history of operators on each device cannot be seen with a polyline that encodes the execution start time due to the characteristics of parallel training. In parallel training scenarios, operators are trained in parallel between devices, rather than serial bus/train schedules between stations. And it is common for the device to have idle time between operator executions, which means that the execution of the next operator does not start immediately after one operator completes. Thus, we introduce the concept of time-span. In addition to the execution-starting line, a line that indicates the end of the execution (execution-ending line) is drawn. And the space between them forms a band by means of coloring (Figure 6(c)). The color of the bands represents the operator type. As shown in Figure 4(B3), the green bands encode computational operators in forward and backward propagation, the yellow bands encode collective communication operators. As for peer-to-peer communication operators, the purple bands encode *Send_op* operators, the blue bands encode *Receive_op* operators. The band shows the completion condition of the operator on each device. Differences of operators' execution can be shown from two aspects. One is the width of the band at the intersection with the time axis (**intersection width**), as shown in Figure 7(b). The other is the height of the triangle formed by the left and right boundaries of the band (**triangle height**), as shown in Figure 7(c). Particularly, the two boundaries are of adjacent devices and the bottom of the triangle is determined by the corresponding width of the band. In addition, the time axis of each device is further indicated by a flattened line chart, in which FLOPs (floating point of operations) and memory information are depicted. On the left side of the view, we use a tree structure to denote participating devices at different training stages, and a pixel map to denote the relative size of FLOPs, FLOPS (floating point of per second) and peak memory for different devices.

The enhanced Marey's graph shows profiling data that allows the inspection of when and on which device the delay occurs. Moreover, the set of visual patterns emerging from the visualization can be used to analyze the causes of the delay. The users can identify delays, visually indicated by the thick intersection widths, or the small triangle hights. Figure 7 demonstrate the different types of visual patterns, which are listed as follows.

- ***Normal efficient operators.*** In Figure 7(a), the intersection width on each time axis is uniform and not very thick for each operator, that is, the polylines between the two time axes are nearly parallel to each other and the triangle height can be infinity. This visual pattern indicates operators are executed smoothly on each device without delay.
- ***Long-executing operators on all devices.*** In Figure 7(b), the intersection width on each time axis is uniform for each operator. But the band of the operator $op2$ is abnormally thicker than others, indicating that the execution time of this operator is too long on all devices. This might be a normal execution pattern where the long execution time is determined by the operator's own characteristics, such as the *CombineMomentum* operator. It might also be an abnormal execution pattern. For example, an unreasonable parallelization strategy can lead to excessive collective communication costs, causing the entire training to be stuck on a collective communication operator.
- ***Long-executing operators on partial devices.*** In Figure 7(c), the triangle height $h1$ between the devices $D3$ and $D4$ on the band of the operator $op1$ is too small, indicating that the execution time of $op1$ is delayed on $D4$. Consequently, $D1$, $D2$ and $D3$ must await $D4$ to complete $op1$ before they can start collective communication, which therefore causes communication latency on $D1$, $D2$ and $D3$. The communication latency can be recognized through the small triangle height $h2$.

Alternative Visual Designs. Several design alternatives were considered, as shown in Figure 8. The first and most intuitive one is to use Gantt chart [38](Figure 8(b)), a visual design using parallel-stream timelines, in which parallel data streams are projected onto a time axis. It is often used for schedule management, showing work completed in a certain period of time. The profiling data patterns are possible to be conveyed through Gantt chart. However, when the distance between visual components rises, visual sensitivity to spatial alignment diminishes [39]. Consequently, when the number of devices in the cluster increases vertically, the user's ability

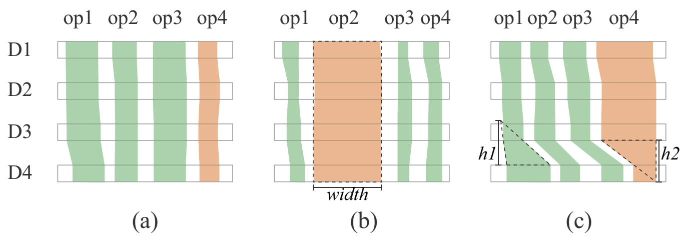


Fig. 7. The visual patterns of the enhanced Marey's graph. (a) The operators are executed smoothly on each device without abnormal delay. (b) The whole parallel training is stuck at the execution of $op2$. (c) The execution of operators is delayed on partial devices.

to discern concurrent status and delay across devices may be compromised. In our enhanced Marey's graph, vertical lines that connect the time points where operators start or end execution on each timeline are used to enhance user space alignment capabilities. Another alternative is temporal mosaics [40]((Figure 8)(c)), which employs a compact way to display concurrent event streams in a specific drawing area, dividing the area proportionately based on the number of concurrent events. However, it need to use color to distinguish the execution sequences in different devices. Considering there will be many devices in a cluster, too many colors can be visually confusing. In our design, we adopt a parallel layout of device timelines, which is easy to compare patterns between devices. In regular discussions with experts, they concluded that "When the number of devices is small, all three designs can convey patterns well, but as the number of devices increases, only the enhanced Marey's graph can effectively locate the delays." This is consistent with our analysis.

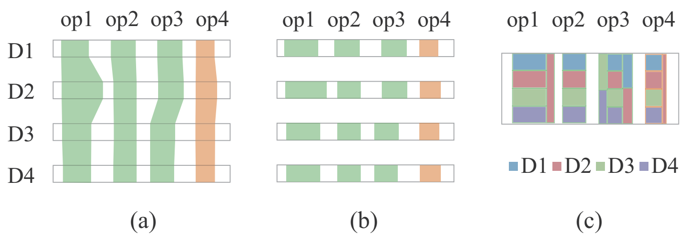


Fig. 8. The design alternatives presenting the same example data: (a) our design, (b) Gantt chart, (c) temporal mosaics.

Timestamp alignment. The timestamps captured across multiple devices may not be aligned. That is, issues such as network latency and out-of-sync device clocks can cause timestamps to be inconsistent across multiple computing devices. When diagnosing intra-stage execution anomalies, the misalignment of the timestamps of each device has little effect on the analysis. Whether aligned or not, the thickness of the band's intersection with each time axis and the angle formed by the band's two polylines between two adjacent time axis will not change, as shown in Figure 9(a). But from a visual point of view, after the oblique band is enlarged to a certain extent, it will become a straight band in the visible area, and the execution context will be lost, as shown in Figure 9(b). Through discussions with experts, the gradient aggregation operator AllReduce requires all devices to participate simultaneously, which is suitable to achieve timestamp alignment. Thus, we find the first *AllReduce* operator on each device and align them. And then other operators on the devices are offset according to the offset of this *AllReduce* operator, as shown in Figure 9(c). When analyzing inter-stage execution status, misaligned timestamps can cause users to see out-of-order minibatches. In pipeline parallelism, peer-to-peer communication is performed between minibatches through *Send_op* and *Receive_op* operators. As shown in Figure 2, after stage1 executes minibatch 1, it will pass the parameters to stage2 through the *Send_op*, and stage2 will receive the parameters through the *Receive_op*. Through discussions with experts, we find the first pair of *Send_op* and *Receive_op* operators and align them, and then shift the remaining operators accordingly. After the timestamp alignment is completed, the pipeline parallelism strategy can be intuitively perceived (Figure 8(d)).

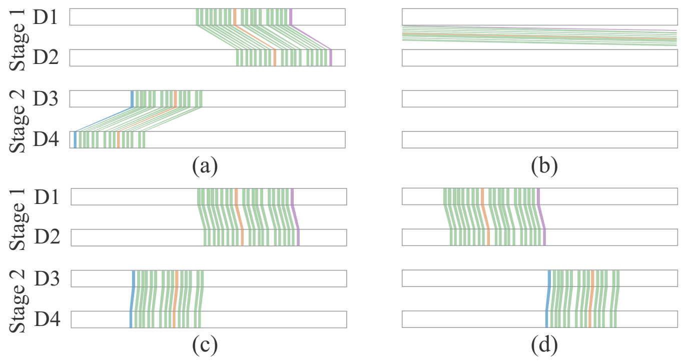


Fig. 9. The illustration of the timestamp alignment using an example pipeline with two stages each running on two devices. (a) The misalignment of the timestamps of each device. (b) When zoom in (a) to a certain extent, bands become straight. (c) The result after intra-stage timestamp alignment. (d) The result after inter-stage timestamp alignment.

Visual Aggregation. Despite the fact that the enhanced Marey's graph could present a variety of visually appealing patterns, it has poor rendering performance owing to a large number of bands. A compelling option is the use of Kernel Density Estimation (KDE), which estimates the density of the lines and generates a heat map based on that estimation. However, it obscures the visual pattern characterized by the shape of the band. In this study, we summarize density with a histogram. Based on experience, communication operators are more concerned by experts in the diagnosis process, and the number of communication operators is much smaller than that of calculation operators. Thus, we aggregate only the bands of computational operators and keep that of communication operators. When forming the histogram, we first group the bands in the visible area into bins and calculate the density of bands that fall into each bin. Once the density in the bin is greater than the average density, we merge the bands in this bin, which will greatly reduce the number of drawn bands and improve the drawing efficiency. The number of bins controls the coarseness of the density distribution. Through experiments with different bin sizes, We finally decided to horizontally divide the visible area of the view into 100 bins. In addition, we provide brushing and zoom-in interaction to enlarge the selected range horizontally to the entire graph width. After users complete brushing, we recalculate the density and aggregate the selected range. Moreover, the chart expands vertically as more devices are added to the cluster. Due to the limited height of the screen space, We aggregate devices of the same stage together. As shown in Figure 4, users can control the expansion and aggregation of devices through the tree selector on the left. The aggregated row displays the average, maximum, and minimum values of the operator's start and end execution times across devices, revealing the execution overview of operators.

5.2.4 Rule-based Issue Detection

To facilitate the diagnosis of parallel training, we automatically detect performance issues in profiling data prior to user exploration based on empirical thresholds provided by experts. The prompt information is displayed at the front of the corresponding line in the enhanced Marey's graph through a red cross glyph (Figure 4(B3)). When hovering over the glyph, the specific prompt information will be displayed. We currently only have empirical thresholds for two metrics, namely, FLOPS and FLOPs. For FLOPS, when the FLOPS value of a device is 20% lower than the average value of each device, the overall performance may suffer. Users should keep an eye on that device. For FLOPs, when the average computation amount of devices in a stage is 20% higher than the average of each stage, users can try to optimize the pipeline parallelism strategy to balance the computation amount of each stage.

5.3 Interactions Among the Views

Our system provides users with rich interactons to facilitate an efficient joint analysis of performance data and parallelization strategy data. After users select a training step in the training overview, they can diagnose the root cause of performance issues in the remaining views. (1) **Diagnosing the long waiting time for collective communicaiton.** Users identify a delay in the enhanced Marey's graph, and identify the device d_t and the operator op_t that cause the delay. Then, users click op_t in the enhanced Marey's graph, and the parallelism strategy view will highlight the corresponding operator and namescope. Users view the attribute information and parallelization strategy of op_t. Combined with the FLOPs of d_t, users can analyze whether the distribution of computation amount among devices is reasonable. Users can also analyze whether d_t is a slow computing device by the FLOPS of d_t. (2) **Diagnosing the long waiting time between stages.** Under pipeline parallelism, users find that the stage $stage_t$ has a long idle time during the training process in the enhanced Marey's graph. Then, users observe how the computational graph is partitioned into different stages in the parallelism strategy view. Combined with the average FLOPs of different stages, users can analyze whether the distribution of computation amount among stages is reasonable. (3) **Diagnosing the long collective communication time.** After users identify the devices with longer communication time in the training overview, they switch to the cluster topology view to brush the corresponding devices. In the popup matrix, users can compare the characteristics of different links to determine whether there is a slow link. (4) **Tracking the parallelization strategies.** When users find no performance issues in the profiling view, they can track the different phases of model training by selecting the namescope in the parallelism strategy view. They can then drill down to the phase of interest, and track the parallelization strategies of the operators along the execution logic. At the same time, they can click an operator and check the performance information in the profiling view. Users can judge whether the parallelization strategies still have room for improvement based on their own experience.

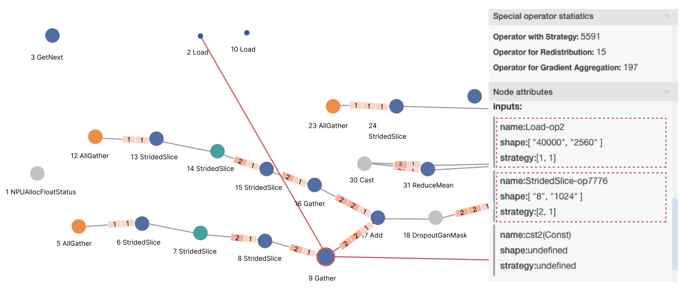
6 Evaluation

6.1 Case Study

We evaluate the usability and effectiveness of our approach by analyzing two large-scale models. These cases were discovered by one of our co-authors, the expert E.

6.1.1 Case 1: Understanding the Parallel Training Process

In the first case study, we used the PanGu-α model with 13 billion parameters [41], which is built by Mindspore and trained on a cluster of 16 devices. The model input data are sentences consisting of words. The words and their position information are vectorized using an embedding algorithm. After loading the data, E noticed there was only one stage, indicating that the pipeline parallelism strategy was not used in the model training. No prompt information was displayed. He noticed that the 16 participating devices performed equally well on metrics of communication, computation, and memory usage. Then he tracked the parallel training process in the parallelism strategy view. First, he checked the parallelism strategy of the *Embedding* layer at the beginning of the model. He started from the *AllGather* operator, and along the main logic of the computational graph, he found the *Gather* operator whose namescope is *word_embedding-VocabEmbedding*. The *Gather* operator took two inputs, namely, the data from the *StridedSlice_op7776* operator and the parameter from the *load_op2* operator. He noticed that the parameter is a 40000×2560 2D matrix without parallelization, which might need to be optimized (Figure 10). According to his diagnostic experience, we need to parallelize the input parameters, that is, use a model parallelism strategy to optimize model performance. Furthermore, he explored the remaining training process by selecting the namescope of interest.



Operator with Strategy: 5591
Operator for Redistribution: 15
Operator for Gradient Aggregation: 197

name:Load-op2
shape:["40000", "2560"]
strategy:[1, 1]

name:StridedSlice-op7776
shape:["8", "1024"]
strategy:[2, 1]

name:cst2(Const)
shape:undefined
strategy:undefined

Fig. 10. Identified the operator *Gather* that might need be optimized.

6.1.2 Case 2: Diagnosing the Cause of a Training Anomaly

We conducted the second case study with the Resnet-50 model [4] consisting of 1120 nodes and 1629 edges, which is trained on the cluster of 4 devices. E started by using the training overview to select Step 1 for detailed inspection. In this view, he observed that the communication costs of devices 2 and 3 are relatively larger than the others (Figure 4(1)). Then he switched to the cluster topology view. He brushed the nodes representing device 2 and device 3. In the pop-up adjacency matrix, he identified an abnormal communication operator (Figure 4(2)). However, he was confused that the abnormal operator had the maximum bandwidth and communication duration at the same time. By clicking the abnormal operator, he noticed that the intersection width is abnormally thick in the enhanced Marey's graph (Figure 4(3)). To further identify the root cause of the communication delay, he brushed all devices in the cluster topology view. As shown in Figure 11, the traffic between device 0 and device 1 was less than the traffic between device 2 and device 3, while the bandwidth of all devices was about the same. Thus, the communicaition duration between device 2 and device 3 was longer. What's more, he also found an interesting phenomenon. The abnormal communication operator, which uses the link between device 2 and device 0, has maximum bandwidth and the minimum communicaition duration.

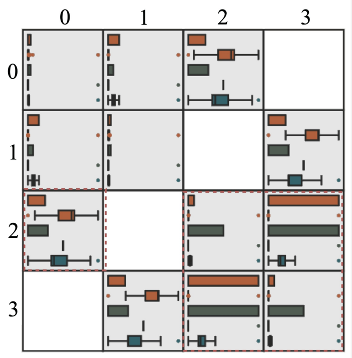


Fig. 11. Communication metrics for links between all devices when training Resnet-50. A cell with a white background represents no link between two devices.

E noticed that the pipeline parallelism strategy was adopted. The parallelism strategy view showed the pipeline with two stages each running on two devices. He selected namescopes through the namescope selector, and found that there were four minibatches in the pipeline. The illustration of the timeline for the pipeline is shown in Figure 4(4), which is similar to Figure 2. Then he noticed that stage 1 had more computation than stage 0 in both the parallelism strategy view and the enhanced Marey's graph. To verify the guess, he hovered on the pixel map denoting FLOPs, and found that the FLOPs of stage 1 were slightly larger than those of stage 0. In addition, he boldly guessed that maybe he could combine the parallelism strategy view and the enhanced Marey's graph to find a reasonable parallelization position.

6.2 User Study

The user study is conducted to prove that experts can diagnose the root cause of performance issues in parallel model training efficiently and accurately using our system. To show that our system performs better, we contrast it with the Cloud TPU, a state-of-the-art tool for diagnosing performance issues (Figure 12). In order to show the importance of the parallelism strategy perspective in performance diagnosis, we also compare our entire system with the profiling view of our system.

6.2.1 Experimental Design

Tasks. After the discussion with experts, we design two tasks, each of which has several questions.

T1: (**R1**) Explore and understand the parallelization strategies using the PanGu-α model in Case 1. This task is designed on the basis of Case 1.

- **Q1:** What parallelization strategies are used in model training?
- **Q2:** Find the *Gather* operator in the namescope *embedding-EmbeddingLayer*, check its predecessor and successor operators, and then describe the parallelization of the input position vector and word vector.

T2: (**R2, R3**) Diagnose the root cause of performance issues.

- **Q3:** Using the Resnet-50 model in Case 2, participants need to diagnose performance issues and identify the root cause. This task is designed on the basis of Case 2.

Participants and apparatus. We recruited 30 participants (10 females and 20 males; graduate students and DL engineers with

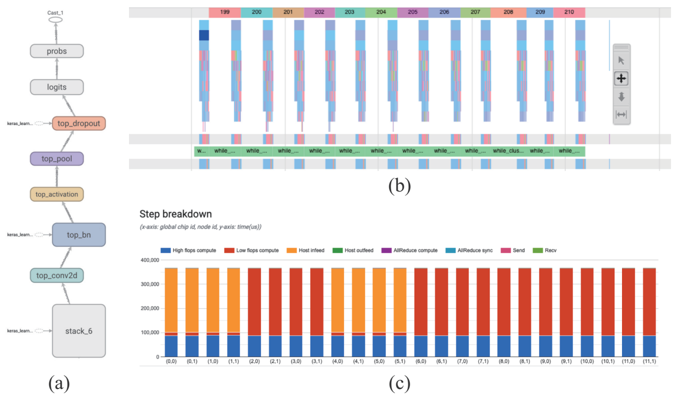


Fig. 12. Cloud TPU contains different modules, including (a) computational graph, (b) trace view and (c) profiling metric visualization. Users can use the computational graph to view operator attributes and parallel allocation. The trace view displays the execution records of operators. Metric visualization provides an overview and details of training performance. There is no explicit linking between modules.

6-9 years of experience in cluster diagnostics, average experience: 7.12 years) for the user study. Participants were invited to our lab where they complete the tasks on the same 1920×1080 display with a Chrome browser to maintain a consistent environment. The source code for our system can be seen at https://gitee.com/zerowangzy/mindinsight.

Procedure. The tools we need to compare include the Cloud TPU (**CT**), our entire system (**En**), and the profiling view of our system (**Pr**). To prevent learning effects from affecting the experimental results, we split the participants into three groups, each of which used a single tool for all tasks. We first described the visual design and system interactions prior to the formal study. Then, in order to further their grasp of system interactions, we present diagnostic examples. After completing the tutorial, the participants are required to perform the tasks and rate their confidence in answering each question on a five-point Likert scale, ranging from 1 (lowest confidence) to 5 (most confidence). We encouraged them to think aloud during their exploration. Finally, we conducted an interview with participants.

6.2.2 Results

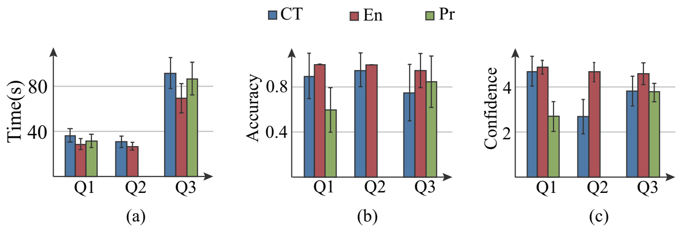
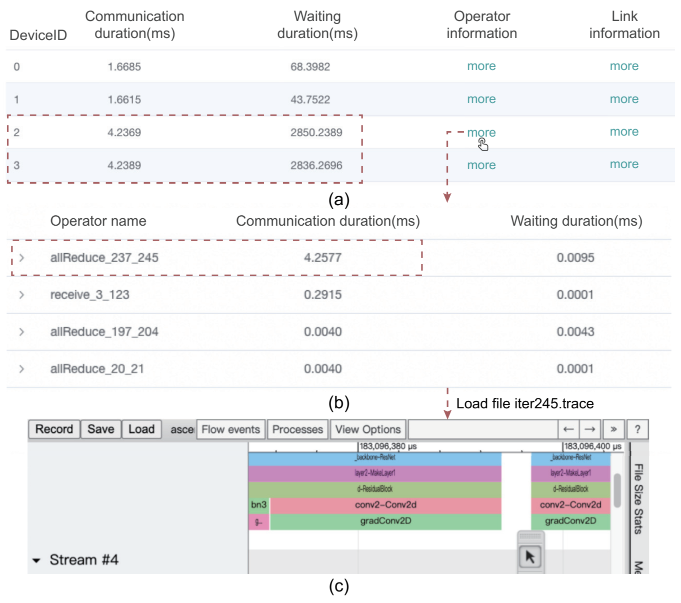


Fig. 13. The average (a) completion time, (b) accuracy, and (c) confidence of the user study when using the Cloud TPU (**CT**), our entire system (**En**), and the profiling view of our system (**Pr**). Note that there are no **Pr** results for **Q2**, because **Q2** requires the hierarchical structure information of operators, which is not provided in the separate performance view.

We collect all the results from the study, including the completion time, the answer, and the confidence in answering each question. The answers are transformed into accuracy by comparing the participants' answers to accurate answers. Then the mean and standard deviation of the completion time, accuracy, and confidence are calculated. We observe three findings from the result (Figure 13).

- **Our entire system could facilitate the comprehension of cluster parallelization strategy.** As shown in Figure 13(b), the mean accuracy of **En** is higher than that of **CT** in Q1 and Q2. And the mean confidence of **En** is also higher than that of **CT** in Q1 and Q2 (Figure 13(c)), indicating that participants could have a clearer understanding of the strategy when using our system.
- **Connecting the performance data with the computational graph would facilitate more accurate root cause diagnosis.** In Figure 13(b), the mean accuracy of **En** is higher than that of **Pr** in Q3. And in Figure 13(a), the mean completion time of **En** is also lower than that of **Pr** in Q3. This indicates that participants could diagnose performance issues relatively quickly and well.
- **Our entire system could help improve diagnostic efficiency.** In Figure 13(a), the mean completion time of **En** is the lowest among three tools in Q3, indicating improved diagnostic efficiency. Some participants reported that in the process of using Cloud TPU, it is necessary to remember the currently concerned operator and switch pages for further search and analysis.



DeviceID	Communication duration(ms)	Waiting duration(ms)	Operator information	Link information
0	1.6685	68.3982	more	more
1	1.6615	43.7522	more	more
2	4.2369	2850.2389	more	more
3	4.2389	2836.2696	more	more

(a)

Operator name	Communication duration(ms)	Waiting duration(ms)
allReduce_237_245	4.2577	0.0095
receive_3_123	0.2915	0.0001
allReduce_197_204	0.0040	0.0043
allReduce_20_21	0.0040	0.0001

(b)

(c)

Fig. 14. The interactive analytic process using the Cloud TPU.

To further examine how our system is better, we observe the operation processes of the participants and summarize the interactive analytic process using Cloud TPU, as shown in Figure 14. We note that, when diagnosing the root cause of performance issues, participants must utilize various documents to do switching comparisons in multiple views. This makes the analysis inefficient, and it is difficult to discover the root cause of issues. Using the cluster communication data, participants determine that the communication duration and waiting duration of devices 2 and 3 are anomalous (Figure 14(a)). Then, by examining operator details, it is determined that the operator *allReduce_237_245* has an abnormal communication duration (Figure 14(b)). To identify the root cause of the communication issue, they use the trace view to examine the operator execution state and check for long-running operators on the selected device (Figure 14(c)). Other than the communication operator, however, there were no other abnormal operators. In order to confirm the root cause further, they examine the operator execution state of devices 0 and 1 that are not suffering communication issues. Unfortunately, only one device's operator execution state can be displayed in the trace view. If they intend to examine with additional devices, they must reload the trace file. If they intend to inspect the execution context of the desired operator, they must switch to the computational graph and search for the operator. The Cloud TPU's computational graph has a hierarchical architecture, which is not conducive to matching the execution logic of operators. In addition, when participants encounter the pipeline parallelism strategy, they must switch between different stages and are unable to observe the execution relationship between stages, making it difficult to get more insights.

6.3 Expert Feedback

We summarized the expert feedback from the user study as follows.

System. Experts were all impressed by the system, commenting that the system was useful and user-friendly for cluster diagnostics. They all agreed that the parallel strategy data and performance data were well integrated, especially when they found that when exploring abnormal communication operators, they can see multiple aspects of information in different views. One expert commented that *"Usually I have to jump between different data to identify an abnormal communication operator, but now I can recognize it directly and find the cause easily"*. Another expert said that *"The computational graph is well combined with the performance data, and it is easy to return to the model structure for analysis after reviewing the model execution results. It is a pity that the current data does not support one-to-one matching of operators before and after model compilation"*. In addition, some experts commented that the workflow of the system was consistent with conventional analysis methods, making it easy to use. However, all the experts felt that they needed to learn a lot of things at the beginning of the use. Some experts suggested that the system could include a tutorial and give prompts for the next interaction, and adviced that some temporarily unnecessary views could be collapsed to make the interface simpler.

Visual designs. Concerning the visual designs, all experts gave positive feedback. They agreed that they could fully understand the visual encoding of the system after our introduction. The cognitive load is acceptable. Some experts mentioned that the enhanced Marey's graph was very straightforward representation of the operator execution data, and the integrated linechart was useful to track FLOPs and memory dynamics. One expert liked the design for the cluster topology, and stated that *"The visual encodings and interactions of the cluster topology view is useful for cluster-level cause location"*. Some experts agreed that the layout of the computaional graph could enhance the comprehension of operator execution logic, and recognize whether the distribution of operators was evenly between stages. However, they also felt that the graph was narrow and long, especially when the model was very large. Fortunately, the namescope selector could speed up the exploration.

7 Discussion

Target Users. Our system is designed for model parallel training experts. They can use the system to understand the parallelism strategies through the parallelization forms of a tensor, and combine profiling data to diagnose the root cause of performance issues.

Though we have provided anomaly information prompts, the target users must have training experience with large-scale models.

Scalability. A large quantity of performance data and strategy data are generated in a cluster parallelization training process. Displaying data directly without processing can lead to system failure. On the system's backend, we precompute summary statistics utilized in the visualizations and cache the results to accelerate response time. In addition, we have designed some schemes to improve the rendering performance of the computational graph and the enhanced Marey's graph. The primary purpose of schemes is to decrease the amount of components presented on a page. Specifically, our performance optimization for computational graphs is reflected in two aspects. First, we configure the type of edge and hide a large amount of edges that are not important for the comprehension of the main logic. In addition, we automatically detect similar substructures and stack them. Second, we only draw nodes within the viewable window. As for the enhanced Marey's graph, we aggregate a large number of operators horizontally by density, and perform vertical multi-level aggregation for a great number of devices. Moreover, only the data within the selected range is displayed during the process of brushing and zooming in. Besides MindSpore, our system is applicable to other deep learning frameworks. It only needs to preprocess the data into the appropriate format in the data processing stage. As the model's scale is further increased, experts may propose new parallelism strategies. Our proposed visual design for the parallelism strategy can support practically all parallelism strategies since we employ partitions of operators' output tensors to describe various parallelism strategies.

Limitations. Data is one of our primary design concerns. We have discovered concerns with data volume and quality.

Mismatch of operators: When conducting interactions among views, the operators in the parallelism view cannot be matched to the operators in the profiling view one by one. This is caused by the characteristics of the underlying data. We use the pre-compiled model data to draw a computational graph, while the profiling data used for the cluster topology view and the enhanced Marey's graph is the model data compiled and optimized by the deep learning platform. The platform usually conducts a series of optimizations to improve model performance, such as automatically generating namescopes, merging communication operators, adding communication operators, *etc.*, resulting in the inconsistency of operators before and after compilation. Therefore, users can only recognize operators with the same namescope as the selected node. Our system analysis would be more accurate if we could build a tracker that generates mapping data between pre-compiled model data and post-compiled model data.

A substantial number of operators: In order to accurately convey the execution order of computational graph operators, we open all namescopes and draw operators directly. This will greatly increase the number of nodes in the computational graph, affecting rendering efficiency. Our similar substructure stacking can optimize efficiency to some extent. More solutions need to be explored, such as expert configuration of stackable structures, optimization of graph layout, etc. In order to maintain the execution order of the operators, we add the execution order to the force-directed layout, making the computational graph narrow and long. Especially for very large models, the interaction can be very inefficient. We have designed the namescope selector to facilitate the exploration of the comptational graph with automatic jumps. The exploration efficiency under hyperscale data needs to be improved.

Future work. There are several possible future work directions.

Computational graph layout optimization: In our current implemented graph layout, the expected distance a between nodes is a constant, i.e., a is the same between every pair of nodes. We can consider setting different a according to the semantic correlation between nodes to further improve the readability of the computational graph.

Automatic detection of performance issues: Our system automatically detects performance data based on expert experience thresholds. Furthermore, data mining algorithms can be designed to mine abnormal patterns.

Online analysis: Currently, all training data is gathered offline and then supplied into the analysis tool for further analysis. With online analysis, experts are able to monitor real-time running outcomes and halt inefficient training promptly. Experts can also observe the impact of different strategies on training.

8 Conclusion

In this paper, we propose a novel visual analysis approach for the application area of diagnostics of parallel performance in training large-scale models. We design and implement a multi-view interface to effectively convey complicated performance metrics and parallelization strategies. Analysts are allowed to flexibly identify performance issues at different levels with multi-faceted patterns. In particular, we extend Marey's graph by incorporating the concept of time-span and a banded visual metaphor to convey training dynamics. In addition, we improve its visual scalability by horizontally and vertically aggregation. The execution sequence have been introduced to the computaional graph layout to facilitate the comprehension of parallelization strategies. Two case studies using large-scale models demonstrate the effectiveness of our approach. In the future, we plan to further enhance our scalability to make the exploration of large models more efficient.

References

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[2] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[3] S. Yang, Y. Wang, and X. Chu, "A survey of deep learning techniques for neural machine translation," *arXiv preprint arXiv:2002.07526*, 2020.

[4] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[5] D. Amodei, S. Ananthanarayanan, R. Anubhai, J. Bai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, Q. Cheng, G. Chen *et al.*, "Deep speech 2: End-to-end speech recognition in english and mandarin," in *International conference on machine learning*. PMLR, 2016, pp. 173–182.

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.

[7] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang *et al.*, "Large scale distributed deep networks," *Advances in neural information processing systems*, vol. 25, 2012.

[8] N. Shazeer, Y. Cheng, N. Parmar, D. Tran, A. Vaswani, P. Koanantakool, P. Hawkins, H. Lee, M. Hong, C. Young *et al.*, "Mesh-tensorflow: Deep learning for supercomputers," *Advances in neural information processing systems*, vol. 31, 2018.

[9] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-lm: Training multi-billion parameter language models using model parallelism," *arXiv preprint arXiv:1909.08053*, 2019.

[10] D. Narayanan, A. Harlap, A. Phanishayee, V. Seshadri, N. R. Devanur, G. R. Ganger, P. B. Gibbons, and M. Zaharia, "PipeDream: generalized pipeline parallelism for DNN training," in *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, 2019, pp. 1–15.

[11] Y. Huang, Y. Cheng, A. Bapna, O. Firat, D. Chen, M. Chen, H. Lee, J. Ngiam, Q. V. Le, Y. Wu *et al.*, "Gpipe: Efficient training of giant neural networks using pipeline parallelism," *Advances in neural information processing systems*, vol. 32, 2019.

[12] M. Yamazaki, A. Kasagi, A. Tabuchi, T. Honda, M. Miwa, N. Fukumoto, T. Tabaru, A. Ike, and K. Nakashima, "Yet Another Accelerated SGD: ResNet-50 Training on ImageNet in 74.7 seconds," *arXiv preprint arXiv:1903.12650*, 2019.

[13] K. Wongsuphasawat, D. Smilkov, J. Wexler, J. Wilson, D. Mane, D. Fritz, D. Krishnan, F. B. Viégas, and M. Wattenberg, "Visualizing dataflow graphs of deep learning models in tensorflow," *IEEE Transactions on Visualization and Computer Graphics*, vol. 24, no. 1, pp. 1–12, 2017.

[14] N. S. Keskar and R. Socher, "Improving generalization performance by switching from adam to sgd," *arXiv preprint arXiv:1712.07628*, 2017.

[15] A. Bilal, A. Jourabloo, M. Ye, X. Liu, and L. Ren, "Do convolutional neural networks learn class hierarchy?" *IEEE Transactions on Visualization and Computer Graphics*, vol. 24, no. 1, pp. 152–162, 2017.

[16] D. Liu, W. Cui, K. Jin, Y. Guo, and H. Qu, "Deeptracker: Visualizing the training process of convolutional neural networks," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 10, no. 1, pp. 1–25, 2018.

[17] M. Liu, J. Shi, K. Cao, J. Zhu, and S. Liu, "Analyzing the training processes of deep generative models," *IEEE Transactions on Visualization and Computer Graphics*, vol. 24, no. 1, pp. 77–87, 2017.

[18] F.-Y. Tzeng and K.-L. Ma, *Opening the black box-data driven visualization of neural networks*. IEEE, 2005.

[19] M. Liu, J. Shi, Z. Li, C. Li, J. Zhu, and S. Liu, "Towards better analysis of deep convolutional neural networks," *IEEE Transactions on Visualization and Computer Graphics*, vol. 23, no. 1, pp. 91–100, 2016.

[20] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[21] H. Guo, S. Di, R. Gupta, T. Peterka, and F. Cappello, "La VALSE: Scalable Log Visualization for Fault Characterization in Supercomputers," in *EGPGV@ EuroVis*, 2018, pp. 91–100.

[22] T. Fujiwara, P. Malakar, K. Reda, V. Vishwanath, M. E. Papka, and K.-L. Ma, "A visual analytics system for optimizing communications in massively parallel applications," in *2017 IEEE Conference on Visual Analytics Science and Technology (VAST)*. IEEE, 2017, pp. 59–70.

[23] G. Xian, Y. Tang, W. Yang, X. Li, X. Zhang, and J. Yu, "Visual Analysis of the High-performance Computing Jobs Based on the Comprehensive Load Scoring Algorithm," in *2021 7th International Conference on Computer and Communications (ICCC)*. IEEE, 2021, pp. 1436–1443.

[24] B. Bach, P. Dragicevic, D. Archambault, C. Hurter, and S. Carpendale, "A review of temporal data visualizations based on space-time cube operations," in *Eurographics conference on visualization*, 2014.

[25] E. R. Tufte, "The visual display of quantitative information," *The Journal for Healthcare Quality (JHQ)*, vol. 7, no. 3, p. 15, 1985.

[26] C. Palomo, Z. Guo, C. T. Silva, and J. Freire, "Visually exploring transportation schedules," *IEEE Transactions on Visualization and Computer Graphics*, vol. 22, no. 1, pp. 170–179, 2015.

[27] P. Xu, H. Mei, L. Ren, and W. Chen, "ViDX: Visual diagnostics of assembly line performance in smart factories," *IEEE Transactions on Visualization and Computer Graphics*, vol. 23, no. 1, pp. 291–300, 2016.

[28] B. Milash, C. Plaisant, and A. Rose, "Lifelines: visualizing personal histories," in *Conference Companion on Human Factors in Computing Systems*, 1996, pp. 392–393.

[29] T. Gu, M. Zhu, W. Chen, Z. Huang, R. Maciejewski, and L. Chang, "Structuring Mobility Transition With an Adaptive Graph Representation," *IEEE Transactions on Computational Social Systems*, no. 99, pp. 1–12, 2018.

[30] D. Gotz and H. Stavropoulos, "Decisionflow: Visual analytics for high-dimensional temporal event sequence data," *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, no. 12, pp. 1783–1792, 2014.

[31] M. Monroe, R. Lan, H. Lee, C. Plaisant, and B. Shneiderman, "Temporal event sequence simplification," *IEEE Transactions on Visualization and Computer Graphics*, vol. 19, no. 12, pp. 2227–2236, 2013.

[32] K. Wongsuphasawat and D. Gotz, "Exploring flow, factors, and outcomes of temporal event sequences with the outflow visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 18, no. 12, pp. 2659–2668, 2012.

[33] Z. Jia, M. Zaharia, and A. Aiken, "Beyond Data and Model Parallelism for Deep Neural Networks," *Proceedings of Machine Learning and Systems*, vol. 1, pp. 1–13, 2019.

[34] A. Harlap, D. Narayanan, A. Phanishayee, V. Seshadri, N. Devanur, G. Ganger, and P. Gibbons, "Pipedream: Fast and efficient pipeline parallel dnn training," *arXiv preprint arXiv:1806.03377*, 2018.

[35] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "TensorFlow: A System for Large-Scale Machine Learning," in *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, 2016, pp. 265–283.

[36] K. Sugiyama, S. Tagawa, and M. Toda, "Methods for visual understanding of hierarchical system structures," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 11, no. 2, pp. 109–125, 1981.

[37] N. Henry, J.-D. Fekete, and M. J. McGuffin, "NodeTrix: a hybrid visualization of social networks," *IEEE Transactions on Visualization and Computer Graphics*, vol. 13, no. 6, pp. 1302–1309, 2007.

[38] H. L. Gantt, *Organizing for work*. Harcourt, Brace and Howe, 1919.

[39] S. J. Waugh and D. M. Levi, "Spatial alignment across gaps: contributions of orientation and spatial scale," *JOSA A*, vol. 12, no. 10, pp. 2305–2317, 1995.

[40] S. Luz and M. Masoodian, "Visualisation of parallel data streams with temporal mosaics," in *2007 11th International Conference Information Visualization (IV'07)*. IEEE, 2007, pp. 197–202.

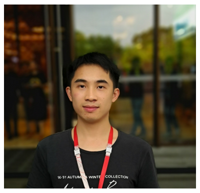
[41] W. Zeng, X. Ren, T. Su, H. Wang, Y. Liao, Z. Wang, X. Jiang, Z. Yang, K. Wang, X. Zhang *et al.*, "PanGu-α: Large-scale Autoregressive Pretrained Chinese Language Models with Auto-parallel Computation," *arXiv preprint arXiv:2104.12369*, 2021.

Yating Wei is a Ph.D. student in the State Key Lab of CAD&CG at Zhejiang University, Hangzhou. She earned the B.S. degree in software engineering from Central South University in 2017. Her research interests are visual analytics and perceptual consistency.

Zhiyong Wang is a postgraduate in the State Key Lab of CAD&CG at Zhejiang University, Hangzhou. His research interests are visual analytics.

Zhongwei Wang is a postgraduate in the State Key Lab of CAD&CG at Zhejiang University, Hangzhou. His research interests are visual analytics.

Yong Dai is a postgraduate in the State Key Lab of CAD&CG at Zhejiang University, Hangzhou. His research interests are visual analytics.

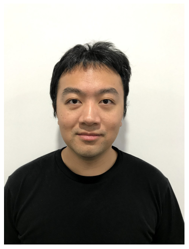


Gongchang Ou received the bachelor's degree from Chongqing University, in 2018. He is now a research engineer with the Distributed Data Lab in CSI, Huawei Technologies Co., Ltd.

Han Gao received the MS degree from Big Data Technology Program, The Hong Kong University of Science and Technology, Hong Kong SAR, China, in 2019. He is now a research engineer with the Distributed Data Lab in CSI, Huawei Technologies Co., Ltd.

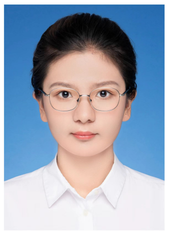
Haitao Yang received the MS degree in Communication and Information Systems from Dalian University of Technology in 2013. He is now a senior engineer in the Distributed Data Lab of Huawei Technologies Co., Ltd.

Yue Wang received the PhD degree from Institute of Computing Technology, Chinese Academy of Sciences. He is now a Senior Principle Engineer with the Distributed Data Lab in CSI, Huawei Technologies Co., Ltd.



Caleb Chen Cao is a specialist in the Dist. Data Lab in CSI, Huawei. He received the Ph.D degree in Computer Science from HKUST. His research interests include explainable AI, AI governance and data fairness.

Luoxuan Weng is a postgraduate in the State Key Lab of CAD&CG at Zhejiang University, Hangzhou. His research interests are visual analytics.



Jiaying Lu is a postgraduate in the State Key Lab of CAD&CG at Zhejiang University, Hangzhou. His research interests are visual analytics.

Rongchen Zhu is a postgraduate in the State Key Lab of CAD&CG at Zhejiang University, Hangzhou. His research interests are visual analytics.

Wei Chen is a professor in the State Key Lab of CAD&CG, Zhejiang University. His research interests include visualization and visual analysis, and has published more than 70 IEEE/ACM Transactions and IEEE VIS papers. He actively served as guest or associate editors of the ACM Transactions on Intelligent System and Technology, the IEEE Computer Graphics and Applications and Journal of Visualization.