applied sciences

MDPI

Article

A Novel Multi-Task Performance Prediction Model for Spark

Chao Shen [1], Chen Chen [1] and Guozheng Rao [2,*]

[1] School of Information and Electrical Engineering, Hebei University of Engineering, Handan 056038, China; shenchao1010@aliyun.com (C.S.); cc0104xy@163.com (C.C.)
[2] College of Intelligence and Computing, Tianjin University, Tianjin 300354, China
* Correspondence: rgz@tju.edu.cn

Abstract: Performance prediction of Spark plays a vital role in cluster resource management and system efficiency improvement. The performance of Spark is affected by several variables, such as the size of the input data, the computational power of the system, and the complexity of the algorithm. At the same time, less research has focused on multi-task performance prediction models for Spark. To address these challenges, we propose a multi-task Spark performance prediction model. The model integrates a multi-head attention mechanism and a convolutional neural network. It implements the prediction of execution times for single or multiple Spark applications. Firstly, the data are dimensionally reduced by a dimensionality reduction algorithm and fed into the model. Secondly, the model integrates a multi-head attention mechanism and a convolutional neural network. It captures complex relationships between data features and uses these features for Spark performance prediction. Finally, we use residual connections to prevent overfitting. To validate the performance of the model, we conducted experiments on four Spark benchmark applications. Compared to the benchmark prediction model, our model obtains better performance metrics. In addition, our model predicts multiple Spark benchmark applications simultaneously and maintains deviations within permissible limits. It provides a novel way for the assessment and optimization of Spark.

Keywords: spark performance prediction; multi-head attention mechanism; convolutional neural network; prediction model

check for updates

Citation: Shen, C.; Chen, C.; Rao, G. A Novel Multi-Task Performance Prediction Model for Spark. *Appl. Sci.* **2023**, *13*, 12242. https://doi.org/10.3390/app132212242

Academic Editor: Yoshiyasu Takefuji

Received: 21 September 2023
Revised: 1 November 2023
Accepted: 9 November 2023
Published: 11 November 2023

Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Spark [1] has emerged as one of the most popular big data computing frameworks in recent years. It finds significant applications in diverse scenarios such as social media [2], smart cities [3], and intelligent transportation [4], delivering robust and consistent support for big data tasks in these areas. However, the ever-growing scale and complexity of big data applications [5] in these domains leads to substantial resource consumption. Efficiently processing these data and optimizing resource utilization is not only a technical challenge but also vital to ensuring smooth computational tasks while meeting business demands. Against this backdrop, predicting Spark performance in advance becomes crucial for making accurate resource allocation and scheduling decisions in real-world applications. Yet, predicting Spark performance is intricate, involving various considerations such as parameter settings, application types, and data scales. Thus, delving into and enhancing the performance prediction methods [6] for Spark to meet the demands of practical applications is a central goal in current research.

In recent years, Spark performance prediction has faced many challenges, such as complex computing environments, large-scale data processing, variable application scenarios, and dynamically changing cluster configurations. In addition, there are significant differences in standardized metrics for Spark performance prediction. These issues make Spark performance prediction challenging. The big data application computing environment is complex and contains numerous parameters such as data distribution, the logical structure of the application, and the configuration of hardware and software. These parameters

can have an impact on the performance of the application. Therefore, performance prediction models need to consider these parameters for more accurate prediction. To solve the above problems, rule-based methods, statistics-based methods, and analytical model-based methods have been proposed successively. Wang et al. [7] proposed an analytical and simulation-based approach for predicting Spark performance. Gao et al. [8] proposed a method based on a black-box model to forecast the execution time of Spark applications. Shah et al. [9] presented an approach that combines model and log analysis to predict the execution time of Spark applications. Al-Sayeh et al. [10] introduced a gray-box modeling technique that integrates both white- and black-box models for predicting the runtime of Spark jobs. Based on the work of some of the researchers mentioned above, it is known that Spark application execution time prediction is a core element of Spark performance prediction. Therefore, the main research goal of this paper is to implement a method for predicting Spark application execution time. However, there are some problems and limitations of these methods mentioned above. Rule-based methods rely heavily on the experience and knowledge of domain experts. It requires constant adjustment and updating of rules to efficiently handle new or complex operations. Statistics-based methods rely on a large amount of historical data. It requires manual feature selection and model selection. This approach may affect the prediction accuracy when dealing with high-dimensional, nonlinear, and large-scale datasets.

Recently, machine learning and deep learning have attracted much attention due to their significant advantages in image recognition, speech recognition, and natural language processing. In addition, deep learning is able to automatically learn high-level features from raw data, which has advantages when dealing with high-dimensional, nonlinear, and large-scale data. AlQuwaiee et al. [11] proposed a regression-based machine learning model to predict the performance of Spark-HBase applications on Hadoop. Singhal et al. [12] proposed a gray-box approach to estimate the execution time of an application on a Spark cluster. In addition, they discussed a machine-learning-based method to build the model. Huang et al. [13] used deep neural networks to train a performance prediction model for Spark applications. Therefore, we believe that using deep learning for Spark performance prediction has great potential. However, Spark typically executes multiple different applications sequentially. A trained prediction model can usually only predict the execution time of one application of Spark and cannot predict the execution of multiple different applications. Therefore, we propose a multi-tasking performance prediction model for Spark that combines multi-head attention networks and convolutional neural networks. The model can predict the execution time of multiple Spark applications after one training.

The contributions in this paper are as follows:

(1) We developed and implemented an automated Spark history log reader. It efficiently extracts historical data from Spark and stores them in a text file.
(2) Among the various configuration parameters of Spark, we carefully selected specific parameters that have a significant impact on execution efficiency. In addition, we employed a dimensionality reduction algorithm to simplify the complexity of the data.
(3) We developed and implemented a neural-network-based multi-tasking performance prediction model for Spark. The model can accurately predict the execution time of single or multiple Spark applications. In addition, we conducted a series of comprehensive experiments from different perspectives to demonstrate the accuracy and applicability of our model.

Section 2 discusses related work, exploring prior research on the performance prediction of Spark. Section 3 introduces the methodology, offering a comprehensive framework of the proposed performance prediction model. Section 4 presents the experimental section, elucidating the conducted experiments in this study and analyzing the obtained results with precision and rigor. Section 5 summarizes the paper.

2. Related Work

In recent years, the research and practice of exploring Spark performance prediction have been deepened. For example, Spark can be used to run big data queries, so query-wise performance prediction is also important. Azhir et al. [14] used Spark and Hadoop to cluster query datasets of various sizes and evaluate query performance. Yadav et al. [15] analyzed the impact of data size on the query execution time for Spark, which is a popular big data query framework. There are also various fields and methods proposed for Spark performance prediction. Lin et al. [16] predicted and compared the performance of Streaming applications running on different configurations of stream processing frameworks and validated them with the Spark Streaming framework. Matteussi et al. [17] provided a comprehensive performance evaluation of Spark Streaming backpressure. Ahmed et al. [18] proposed two different parallelization models for performance prediction. Zhu et al. in [19] proposed a model to capture the execution behavior of tasks, phases, and jobs, and based on the model, implemented a prototype system. The system was used to collect and analyze various performance and system metrics such as execution time and CPU utilization. Prasad et al. in [20] explored the performance of a streaming media application under various adjustable parameters in Spark. Dong et al. in [21] used multiple linear regression, MLR, to derive a correlation formula between cluster load and performance metrics.

With the rapid development of machine learning and neural networks, Spark performance prediction models based on machine learning or neural networks have been proposed. Machine learning uses machine learning algorithms and techniques to build predictive models. It predicts job performance by learning and analyzing factors such as the input data, job characteristics, and execution environment. Cheng et al. [6] proposed a performance prediction model for Spark applications based on the machine learning algorithm AdaBoost. Gao et al. in [8] presented a black-box model that uses various forms of machine learning methods to predict the execution time of Spark applications. Li et al. [22] presented a supervised machine learning model for predicting the performance of Spark applications. They studied and evaluated the utility of using supervised machine learning models to predict the performance of Spark applications. Ye et al. [23] used a hierarchical neural network model with stage-awareness and predicted the performance of Spark in big data computing systems. Ding et al. [1] used GBDT to predict the execution time of Spark jobs. Kordelas et al. in [24] used machine learning to predict upcoming data loads. AlQuwaiee et al. in [11] used regression-based machine learning models to predict the performance of Spark-HBase applications. Machine-learning-based methods are able to automatically learn patterns and trends in data without manually designing rules or assumptions. Ahmed et al. showed in [25] that gradient-promoted ML regressors are more accurate than any existing analytical model as long as the prediction horizon follows the training horizon. An extensive comparison of performance prediction accuracy based on machine learning and existing analytical models was also presented. The miscellaneous framework proposed by Al-Sayeh et al. in [26] can use multiple machine learning algorithms to automatically select appropriate datasets for caching. Machine-learning-based methods are highly adaptable. The constructed machine learning models can be adapted to different types of jobs and data with some generalization ability. In addition, machine learning-based methods can provide high prediction accuracy due to the large amount of data trained and model parameters optimized. However, machine-learning-based methods usually require a large amount of labeled data for training, and collecting and labeling data are time-consuming and resource-intensive. In addition, the process of building and training machine learning models is relatively complex and requires certain specialized knowledge and skills to select appropriate algorithms, process data, and optimize the model. In addition, machine-learning-based methods require model inference at prediction time, which may require some computational resources and time.

Neural networks are different from general machine learning methods in that they process information in a nonlinear and adaptive manner. Neural networks overcome the shortcomings of traditional machine learning for intuition and experience, such as patterns,

speech recognition, and unstructured information processing. However, an RNN with its variants of long- and short-term memory networks is usually suitable for processing sequential data. It is more suitable for dealing with sequence prediction in prediction tasks, such as time series prediction. For example, Lavanya et al. [27] presented an end-to-end big data analytics service based on Spark, Kafka, and LSTM for real-time weather analysis and forecasting. The generalization performance is better than the method based on the regression model. However, neural networks can also be used to make regression predictions. Ye et al. [28] used linear regression (LR), support vector machines (SVM), and artificial neural networks (ANNs) for modeling and prediction. Huang et al. [13] used a reinforcement-learning-based approach to optimize Spark configuration parameters. Our proposed model builds on deep neural networks to predict the Spark execution time for a variety of different applications.

3. Proposed Model

In this section, we detail our constructed model, the multi-head attention–convolutional neural network model (MHAC). As shown in Figure 1, the model consists of three main parts. Firstly, we collect data from Spark runtime activities through the history server and construct the dataset. Secondly, the collected data are processed separately based on whether they are continuous or discrete, and then they are downscaled using a dimensionality reduction algorithm. Finally, we employ the multi-head attention mechanism and a convolutional network to train the model and predict Spark performance.

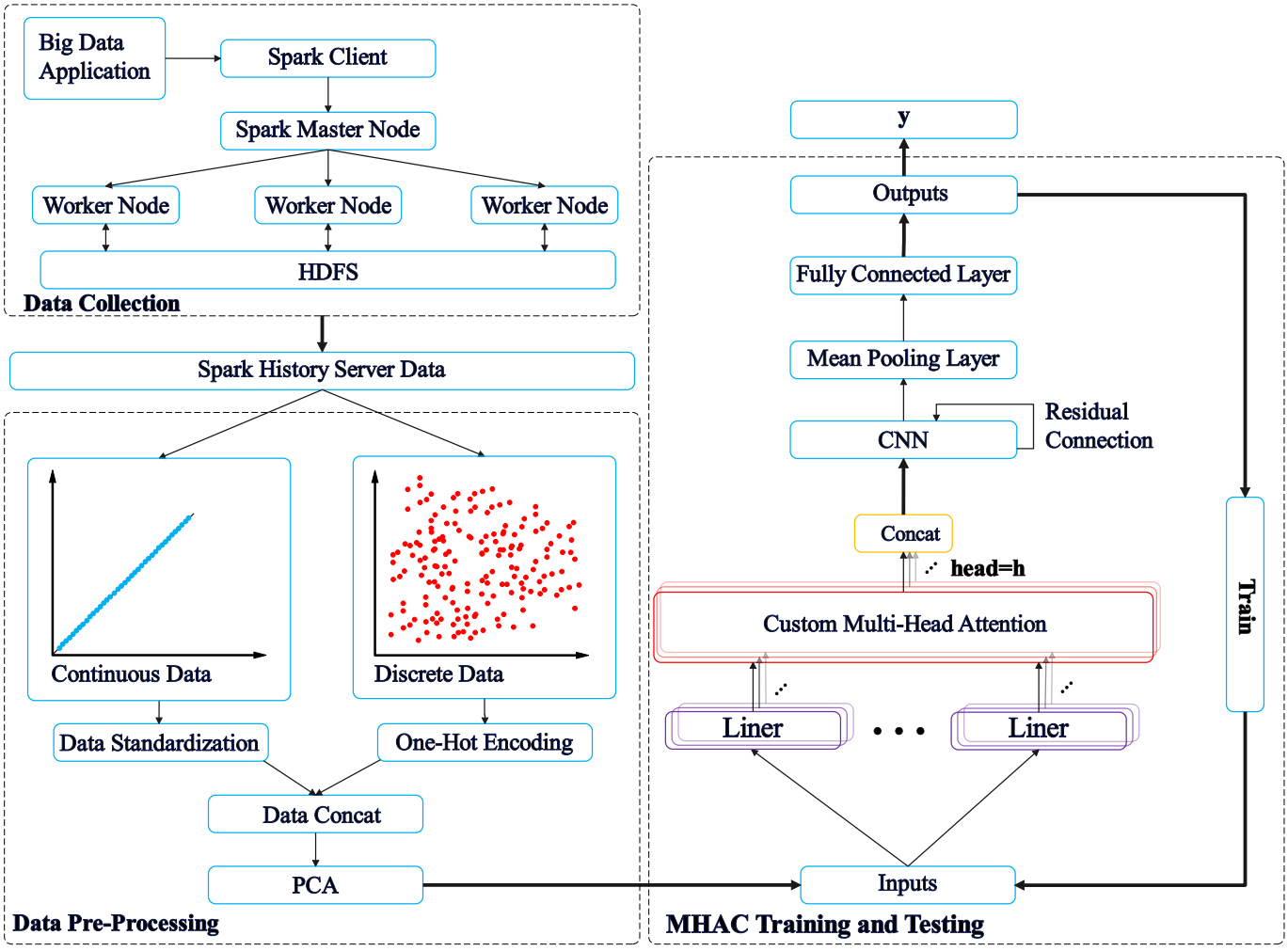


Figure 1. The architecture of the MHAC model.

3.1. Data Collection

Data collection is the first key step in building our MHAC model, aiming to accumulate a large amount of historical data from Spark run activities. Our data collection is primarily based on the Spark History Server. The Spark History Server functions as a log processing tool for Spark clusters, enabling the collection of historical runtime information of Spark jobs. We define the set of all collected Spark log data as $Data = \{data_1, data_2, data_3, \ldots, data_i, \ldots, data_n\}$, where $n, i \in \mathbb{Z}^+$, $data_i$ represents a single historical data entry. The data include start and end times, job execution status, resource utilization, task runtime, and CPU and memory usage. In addition, these data record detailed information about job failures, log errors, and performance bottlenecks. We refer to this dataset as the raw Spark log data.

3.2. Data Preprocessing

Data preprocessing is a crucial step in the process of building a performance prediction model. During this phase, our objective is to transform the Spark log data into a format suitable for model input while ensuring data quality and consistency.

3.2.1. One-Hot Feature Encoding and Feature Normalization

Feature encoding and normalization are essential steps in data preprocessing. Spark log data contain some discrete features, such as some parameters in the Spark execution process. We use One-Hot encoding to treat these discrete features, enabling them to be accepted by the model.

One-Hot encoding is a method to represent discrete variables as vectors, transforming variables with multiple possible values into numeric vectors suitable for machine learning algorithms. We denote the data requiring One-Hot encoding as $data_{dis}$, specifically presented in Equations (1)–(3).

$$data_{dis} = \{(x_{11}, x_{12}, \ldots, x_{1k}), \ldots, (x_{i1}, x_{i2}, \ldots, x_{ik}), \ldots, (x_{n1}, x_{n2}, \ldots, x_{nk})\} \tag{1}$$

$$data_{dis_i} = (x_{i1}, x_{i2}, \ldots, x_{ij}, \ldots, x_{ik}) \tag{2}$$

$$x_j = [1, 2, \ldots, t] \tag{3}$$

Here, $i, j, k, n, t \in \mathbb{Z}^+$. The dataset $data_dis$ consists of n discrete data points. $data_{dis_i}$ denotes any given data point in the discrete dataset, corresponding to the i-th entry and its entire set of discrete feature values. Each data point consists of k discrete features, and x_j stands for any one of these k features, which possesses t potential values.

Given that all feature variables x_j in the dataset $data_{dis}$ are discrete, we can employ One-Hot encoding to transform them into numeric vectors, making them compatible with neural network models.

In conclusion, we successfully transformed discrete variables into numeric vectors using One-Hot encoding, optimizing them for machine learning algorithms. For continuous attributes, such as data scale, normalization was undertaken to eliminate scale disparities among features, ensuring that the model is not overly influenced by a particular feature. After feature encoding and normalization, the consolidated dataset is denoted as $Data = \{data_1, \ldots, data_i, \ldots, data_n\}$, where $i, n \in \mathbb{Z}^+$. Each $data_i$ is represented by the feature set X_i and the predicted or actual value y_i. Any given X_i comprises $\{x_1, x_2, \ldots, x_k, \ldots, x_m\}$, *where* k, $m \in \mathbb{Z}^+$, and x_k represents any specific feature value.

3.2.2. Principal Component Analysis (PCA) Dimensionality Reduction Algorithm

During the data preprocessing stage, we start by performing One-Hot encoding on categorical variables. However, this method could lead to a significant increase in data dimensions, thereby raising the complexity of model training and potentially increasing the risk of overfitting. On the other hand, we apply standardization to continuous variables,

which ensures these variables possess uniform magnitudes during the model training process. This prevents adverse effects caused by magnitude disparities in model training. After processing, we combine the standardized continuous variables with the One-Hot encoded categorical variables through concatenation, forming a unified dataset. Nonetheless, it is worth noting that this dataset's dimensions might be somewhat larger compared to the original data.

To solve this problem, we decided to introduce a dimensionality reduction algorithm such as PCA. It is a common data dimensionality reduction method that reduces high-dimensional data to lower dimensions while retaining as much important information as possible from the original data. The main steps of PCA include:

(1) Data standardization: since PCA is sensitive to the variance in the initial variables, all features need to be standardized to have a mean of 0 and a standard deviation of 1. We set the features as a feature matrix $X \in R^{n\times m}$, as shown in Equation (4).

$$X = \begin{pmatrix} x_{11} & \cdots & x_{1k} & \cdots & x_{1m} \\ \vdots & \ddots & & & \vdots \\ x_{i1} & & x_{ik} & & x_{im} \\ \vdots & & & \ddots & \vdots \\ x_{n1} & \cdots & x_{nk} & \cdots & x_{nm} \end{pmatrix}_{n\times m} \tag{4}$$

where $i, k, n, m \in \mathbb{Z}^+$, n is the number of data points, m is the number of features, and x_{ik} is a feature value of data point i. For each feature x_k, we calculate the mean value μ_{x_k}, which is calculated as shown in Equation (5).

$$\mu_{x_k} = \frac{1}{n}\sum_{i=1}^{n} x_{ik} \tag{5}$$

where x_{ik} represents the original value of data point i on feature k. The standard deviation σ_{x_k} is then calculated, as shown in Equation (6).

$$\sigma_{x_k} = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_{ik} - \mu_{x_k})^2} \tag{6}$$

Finally, the feature matrix X is standardized. The standard transformation of each element $x_{ik}^{'}$ in matrix X is shown in Equation (7).

$$x_{ik}^{'} = \frac{x_{ik} - \mu_{x_k}}{\sigma_{x_k}} \tag{7}$$

where $x_{ik}^{'}$ is the normalized value. The feature matrix X becomes $X^{'}$.

(2) Calculate the covariance matrix: we set the covariance matrix as $C \in R^{m\times m}$, which represents the covariance between pairs of features. It is calculated based on all data points to quantify and understand how the features in the dataset are related to each other from a global perspective. Finally, we obtain the covariance matrix as shown in Equation (8).

$$C = \begin{pmatrix} c_{11} & \cdots & c_{1m} \\ \vdots & \ddots & \vdots \\ c_{m1} & \cdots & c_{mm} \end{pmatrix}_{m\times m} \tag{8}$$

In the covariance matrix C, each element c_{jk} is computed as shown in Equation (9).

$$c_{jk} = \frac{1}{n-1}\sum_{i=1}^{n}(x'_{ij} - \mu_{x_j})(x'_{ik} - \mu_{x_k}) \tag{9}$$

where i, j, k, and $n \in \mathbb{Z}^+$. Here, n represents the number of data points and i denotes any data point index, while j and k are indices for any two features of a data point.

(3) Calculate the eigenvalues and eigenvectors: we decompose the covariance matrix into its eigenvalues and eigenvectors. The eigenvalues and eigenvectors of the covariance matrix satisfy the conditions of Equation (10).

$$C\mathbf{a} = \lambda\mathbf{a} \tag{10}$$

where $\mathbf{a}$ is the eigenvector of matrix C and λ is the corresponding eigenvalue. In PCA, the eigenvalues reflect the variability of the data in the direction of the corresponding eigenvectors. Therefore, we usually select the top k largest eigenvectors according to the size of the eigenvalues as the principal components for dimensionality reduction, and k is the dimensionality of the data after dimensionality reduction.

(4) Selection of principal components: the eigenvectors corresponding to the k largest eigenvalues are selected to form an $m \times k$ matrix P, with m being the dimension of the eigenvectors. Specifically, this is shown in Equation (11).

$$P = (a_1, a_2, \ldots, a_k) \tag{11}$$

where the matrix $P \in \mathbb{R}^{m\times k}$ is composed of m-dimensional eigenvectors $\mathbf{a}$.

(5) Mapping to a lower-dimensional space: In this step, we transform the normalized data X' using the matrix of eigenvectors P. This achieves dimensionality reduction, producing the data matrix $Q \in \mathbb{R}^{n\times k}$ in a new lower-dimensional space, as detailed in Equation (12).

$$Q = X^{'}P \tag{12}$$

where X' is the feature matrix after standard transformation and Q is the feature matrix in the reduced space. Finally, we obtain $n \times k$ dimensional data by PCA dimensionality reduction, where n is the number of data points.

3.3. MHAC Model

In this section, as depicted in Figure 1, we introduce the Spark performance prediction model named MHAC.

3.3.1. Custom Multi-Head Attention Networks

In our Spark performance prediction model MHAC, the data after PCA dimensionality reduction are fed into our customized multi-head attention network for processing. Different from the conventional multi-head attention network, our model does not use the traditional K, V, and Q settings, but instead uses specially designed h "heads" to better adapt to our application requirements, and we default to $h = 4$ in this paper. These h "heads" are able to pay attention to each feature of the input data at the same time, learn the weights and attention distributions among them, and capture the complex dependencies among the features. This processing helps extract valuable semantic information from input data and job features, which can then be used for performance prediction. The architecture of the customized multi-head attention network is shown in Figure 2.

The specific implementation and computational process of the customized multi-head attention network are shown in Figure 2. Initially, for the input feature matrix $Q \in \mathbb{R}^{n\times k}$, we generate the new feature matrices $Q_i \in \mathbb{R}^{n\times d}(i = 1, 2, 3, 4)$ through multiple (4 by default) independent linear transformations. Each of these new feature matrices is derived using Equation (13). The expansion of Q_i is detailed in Equation (14).

$$Q_i = QW_i \tag{13}$$

$$Q_i = \begin{pmatrix} q_{i11} & \cdots & q_{i1d} \\ \vdots & \ddots & \vdots \\ q_{ij1} & \cdots & q_{ijd} \\ \vdots & \ddots & \vdots \\ q_{iz1} & \cdots & q_{izd} \\ \vdots & \ddots & \vdots \\ q_{in1} & \cdots & q_{ind} \end{pmatrix}_{n \times d} \tag{14}$$

where $n, j, d, z \in \mathbb{Z}^+$, $W_i \in \mathbb{R}^{k \times d}$ represents the corresponding weight matrix, and $i \in \{1, 2, 3, 4\}$. These four feature matrices represent four different "heads", each focusing on different aspects of the input features. During the training process of the entire multi-head attention network, these weight matrices W_i are updated to minimize the prediction error, thereby optimizing the model. Q_i is a feature matrix with n rows and d columns, where n denotes the number of data points and d represents the dimension of the new feature space.

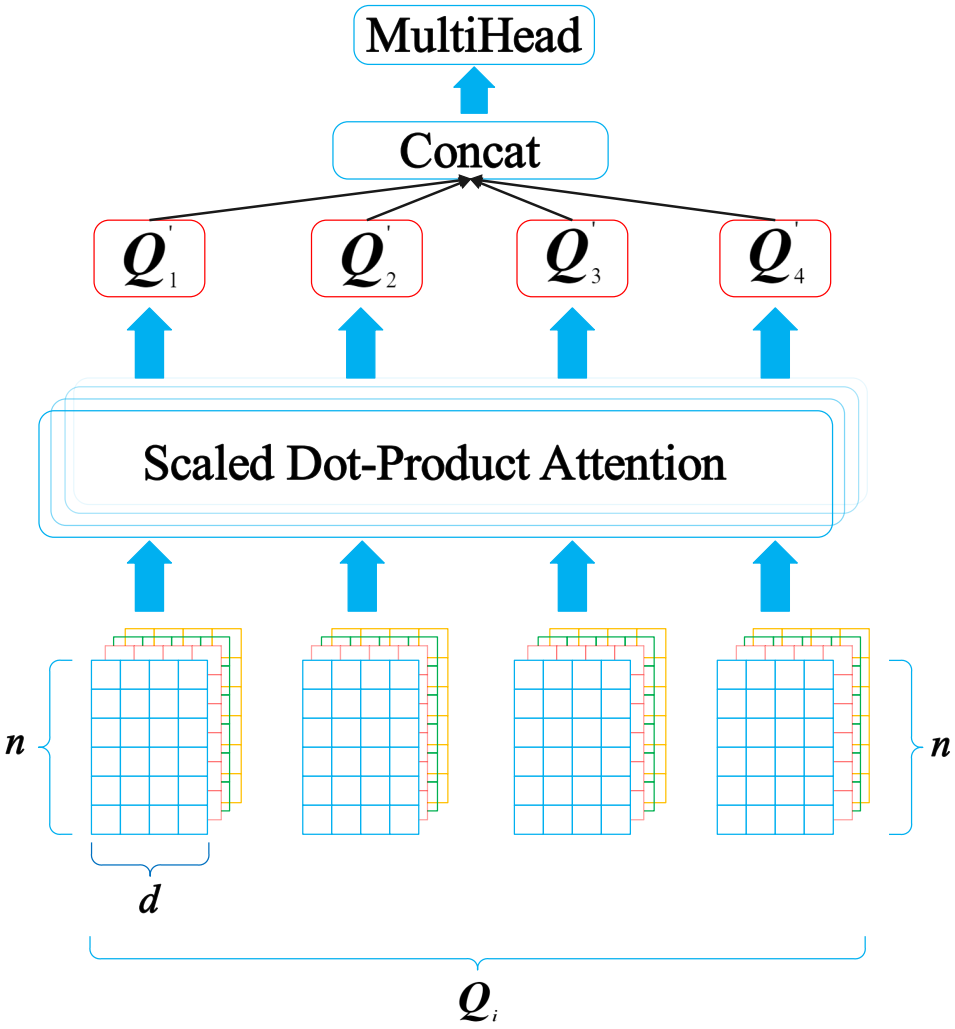


Figure 2. The structure of the multi-head attention network.

Next, the second step involves computing the dot-product attention scores. For each pair of features, we calculate their dot-product attention scores across each head. Specifically, for any two data point features $Q_{li}, Q_{lj} \in \mathbb{R}^{1 \times d}$ from Q_l (where $i, j \in [1, n]$), the attention score att_score_{lij} on head l (with $l \in [1, 4]$) is computed as shown in Equation (15).

The specific expansions of the data point features Q_{li} and Q_{lj} are given in Equation (16) and Equation (17), respectively.

$$att_score = softmax(\frac{Q_{li} * Q_{lj}^T}{sqrt(k)}) \tag{15}$$

$$Q_{li} = (q_{li1}, q_{li2}, \ldots, q_{lid})^{1\times d} \tag{16}$$

$$Q_{lj} = (q_{lj1}, q_{lj2}, \ldots, q_{ljd})^{1\times d} \tag{17}$$

Here, $att_score_{head_{ijl}}$ denotes the attention score between features Q_{li} and Q_{lj} within the l-th head. Both Q_{li} and Q_{lj} are features of data points i and j in the l-th head after the transformation by the weight matrix. The $softmax(*)$ function is an activation function, transforming any real-valued vector into a probability distribution. Its function ensures that all attention scores lie between 0 and 1, and their cumulative sum equals 1. The dot product $Q_{li}Q_{lj}^T$ between features Q_{li} and Q_{lj} is a prevalent method for gauging their similarity. Its magnitude reflects the semantic similarity between feature i and feature j, with a larger dot product indicating greater similarity. The normalization factor $\sqrt{k}$, where k represents the dimensionality of the feature vector, is incorporated to tackle the issue of the dot product rapidly increasing with dimensionality. Without this normalization, the magnitude of the dot product could become excessively large, causing the gradient of the softmax function to approach zero and severely hampering the model's training. Through this formula, we can determine the attention score between features Q_{li} and Q_{lj} for each head. This score signifies the strength of the relationship between Q_{li} and Q_{lj}, aiding the model in capturing and leveraging intricate dependencies among features.

Upon computing the dot-product attention scores, the third step involves leveraging these scores to compute the weighted sum of features for each data point, yielding the novel feature representation for each head. For head l (where $l \in \{1, 2, 3, 4\}$) and data point i, the new feature representation Q'_{li} can be acquired using Equation (18).

$$Q'_{li} = \sum_{j=1}^{n} att_score_{lij} * Q_{lj} \tag{18}$$

Herein, att_score_ijl denotes the attention score between data points i and j in head l. Q_{lj} signifies the feature representation of the j-th data point in head l after transformation by the weight matrix W. Through this weighted summation process, we acquire the new feature representation Q'_{li} for each data point across all heads. This enhanced representation aptly captures the dependencies between data points and features, rendering it beneficial for performance prediction tasks.

After completing the computation of new feature representations for each head, we then proceeded to aggregate these feature representations. We did this by concatenating the new feature representations from the four heads and passing them through a learnable linear transformation W^O. The final feature representation was obtained, as shown in Equation (19).

$$MultiHead = Concat(Q'_1, Q'_2, Q'_3, Q'_4)W^O \tag{19}$$

Here, `MultiHead` denotes the ultimate feature representation, while W^O signifies the linear transformation matrix to be optimized during the training phase. The term $Concat(Q'_1, Q'_2, Q'_3, Q'_4)$ represents the column-wise concatenated feature vectors from the four distinct heads. Upon completion of this step, we secure a novel feature representation encapsulating information from all heads. This derived representation is subsequently input into the Convolutional Neural Network (CNN).

3.3.2. Convolutional Neural Network

Next, we process the extracted features using CNN. After obtaining new feature representations from the multi-head attention network, we input these feature representations into a CNN. CNN is able to capture spatial correlations and local patterns of features through convolution and pooling operations. We chose to use CNN because their deep feature extraction capability helps capture complex patterns latent in the data, which is especially important for prediction tasks.

It is worth noting that we also introduced residual connectivity in the CNN model. The residual connection can help avoid the problem of gradient vanishing during deep network training, enabling the model to learn more complex functions. The network structure of ResNet is shown in Figure 3. With the above design, our CNN model can effectively process and extract deep features to meet the needs of Spark performance prediction tasks. The specific process is shown in Algorithm 1.

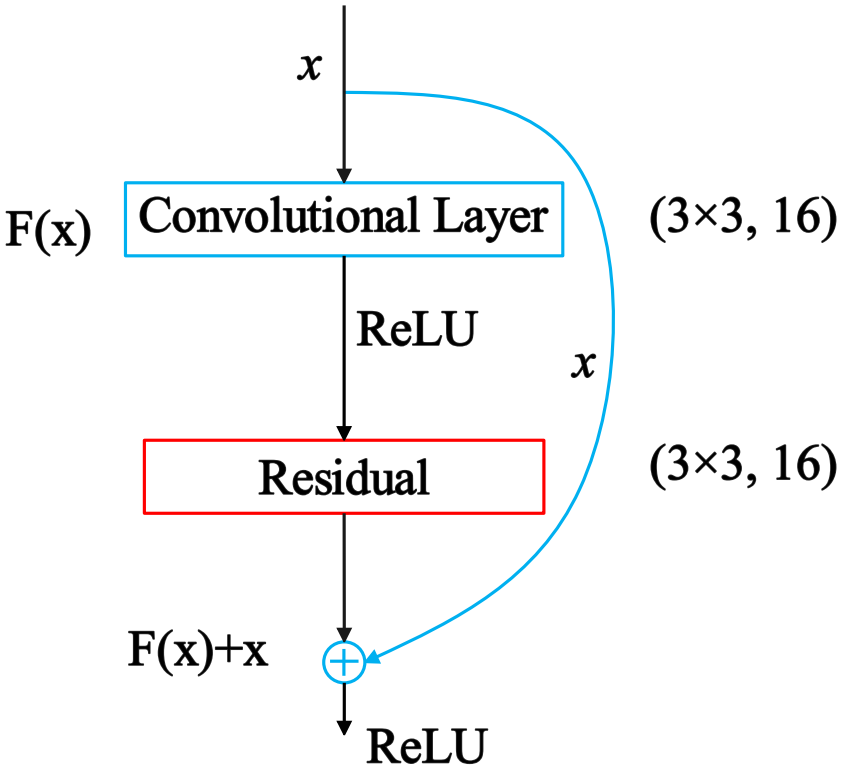


Figure 3. The structure of the ResNet.

Algorithm 1 Residual CNN

1. Input X
2. For each convolutional layer l do
3. Convolution operation with kernel h: $Y = h * X + b$
4. Apply batch normalization: $Y' = \gamma * \left(\frac{Y-\mu}{sqrt(\sigma^2+\epsilon)}\right) + \beta$
5. Apply ReLU activation function: $ReLU(Y) = max(0, Y')$
6. If l>1, residual connection: $X_{l+1} = Y_l + X_l$
7. End for
8. Mean pooling layer: $X'_i = \frac{1}{|W|} * \sum_{j \in W_i} X_j$
9. Apply fully connected layer to obtain the prediction: $Y_{final} = X'W + b$
10. Output: Prediction result Y_{final}

Step 1. Initialize input data, where X represents input features and serves as the final representation of MultiHead.

Step 2. Iterate through each layer l of the CNN.

Step 3. Perform convolution operation using kernel h, with b as bias term, resulting in output Y. The size of h is 3×3 as shown in Figure 3.

Step 4. Apply batch normalization to standardize output Y from step 3. Normalize Y using scaling factor γ, add bias factor β, yielding Y'. Here, ϵ is a small constant to prevent division by zero.

Step 5. Apply the ReLU activation function, known for its simplicity and effectiveness in mitigating gradient vanishing during neural network training.

Step 6. For the second or deeper convolutional layers, execute residual connection. Add the current layer's output Y to the previous layer's input X. This design enables learning of residual mappings, avoiding direct input–output mapping. X_{l+1}, Y_l, and X_l represent inputs and outputs. This enhances the deep network training and model performance.

Step 7. If all convolutional layers are iterated, proceed; otherwise, return to step 2.

Step 8. For each pooling window W_i, perform average pooling. X_i' corresponds to new feature matrix X'.

Step 9. Execute fully connected layer for prediction results. W is the weight matrix, b is the bias vector—both learned during training. X' is the feature matrix after pooling, Y_{final} is the model's prediction.

After passing through the CNN, the model carries out backpropagation to continuously iterate to improve the performance of the model.

4. Experimentation

4.1. Dataset and Experimental Environment

We used four virtual machines to build a Spark cluster with four nodes, using a master–slave architecture, where one node is the master node, and the remaining three nodes are the slave nodes, Worker. The operating system of the virtual machines is CentOS 7.9, the memory of each node is 12 GB, the number of processors is 4, and the number of cores of each processor is 3, i.e., the total number of virtual CPU cores of each node is 12, the hard disk capacity is 100 GB, and the network adapter uses NAT mode. The total number of virtual CPU cores is 12, the hard disk capacity is 100 GB, the network adapter is in NAT mode, and the four virtual machines can communicate with each other. The physical machine has 64.0 GB of memory (63.9 GB available), a system type of 64-bit operating system, and an x64-based processor, all of which are hardware configurations that can support the operation of the Spark cluster.

During the experiment, the Spark cluster built by a virtual machine can efficiently execute multiple benchmarking tasks. We adopt distributed computing, which can fully utilize the computing resources of the cluster, greatly improve the computation efficiency, and shorten the computation time. At the same time, the use of virtual machines also makes the construction of the cluster more flexible and convenient, which can be quickly built and adjusted to improve the operability and stability of the experiment. In this experiment, we choose WordCount, PageRank, Pi, and Logistic Regression (LR) as Spark benchmark test tasks. Among them, the WordCount task needs to count the words in a text file, the PageRank task needs to calculate the PageRank value of a web page, the Pi task needs to calculate the value of π, and the LR task needs to classify the data. The data for the four tasks we used were automatically generated on a virtual machine by HiBench 7.1.1, a big data benchmarking suite that tests the performance of different distributed computing frameworks (e.g., Hadoop, Spark, etc.) under a variety of datasets and workloads. The data sizes performed by the four applications are shown in Table 1.

For each application, we set 50 different data sizes for experimentation. For the WordCount application, we used different sizes of text data for the experiments. The size of the experimental data starts from 100 MB and increases by 100 MB each time until it reaches 5000 MB. At each data size, we run the application and collect its performance metrics such as runtime, CPU utilization, etc. For PageRank, Pi, and LR applications, we adopt a

similar strategy, except that the content and format of the data are adjusted accordingly to the specific application.

Table 1. This is the table of input data sizes used by the Spark applications.

Application Name	Input Data Size	Step
Pi	1–50	10,000
PageRank	1–50	10,000 Pages
WordCount	1–50	100 MB
LR	1–50	100 MB

4.2. Spark Parameter

Spark has hundreds of configuration parameters, but it is unrealistic to use all of them as feature inputs to a Spark application time prediction model. First, too many parameters increase the complexity of building and running the model. Second, some configuration parameters do not have a significant impact on Spark performance. Therefore, we chose some configuration parameters that determine the allocation and usage of resources when Spark executes an application. For example, resources such as the number of CPU cores and the size of memory can have a significant impact on Spark's computational performance. In addition, we also selected configuration parameters that directly or indirectly affect the number of shuffles and parallelism of tasks. The shuffling process is a process that very much affects the execution time of Spark, while the degree of parallelism determines whether Spark can maximize the use of resources reasonably and efficiently. Based on these parameter selection criteria, we chose the Spark configuration parameters listed in Table 2 as potential parameters for the prediction model.

Table 2. This is the table of parameter information.

Parameter Name	Description
spark.num.executors	Number of executors
spark.executor.cores	CPU cores per executor
spark.executor.memory	Memory size per executor
spark.driver.memory	Total amount of memory allocated to the Driver driver
spark.driver.cores	Number of CPU cores allocated to the Driver driver process
spark.executor.instances	Number of instances of executing programs
spark.default.parallelism	Default number of Tasks per stage
spark.memory.fraction	Proportion of memory used for execution and storage
spark.task.cpus	Number of CPU cores allocated to each Task
spark.shuffle.memoryFraction	Percentage of memory occupied by the Shuffle process
spark.shuffle.file.buffer	Buffering of write files during Shuffle
spark.reducer.maxSizeInFlight	The size of the buffer during a Shuffle read
spark.reducer.maxReqSizeShuffleToMem	Maximum value of the data buffer

4.3. Comparative Modeling and Experimental Metrics

We selected the baseline models kNN, LR, SVC, RF, GBDT, and XGB for regression prediction as comparison models. Comparison is made with the proposed MHAC prediction model. In the stage of testing the neural-network-based prediction model, in order to make an assessment of the performance of the prediction model, the experimental evaluation metrics of the prediction model we selected are the mean absolute percentage error (MAPE), the common root mean square error (RMSE), the mean absolute error (MAE), and the R-Square (R^2). Among them, the formula for MAPE is shown in Equation (20), the formula for RMSE is shown in Equation (21), the formula for MAE is shown in Equation (22), and

the formula for R^2 is shown in Equation (23). The smaller MAPE, RMSE, and MAE are, the larger R^2 is, and then the smaller the prediction error of the model is.

$$\text{MAPE} = \frac{100\%}{n}\sum_{i=1}^{n}\frac{|y_i - y_i'|}{y_i} \tag{20}$$

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{n}(y_i' - y_i)^2} \tag{21}$$

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - y_i'| \tag{22}$$

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - y_i')^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{23}$$

where N is the number of sample treatments; y_i' is the predicted values for Spark execution; y_i is the actual value; and $\bar{y}$ is the average value of y.

4.4. Experimental Results

We conducted prediction experiments on the datasets of four baseline application tasks and divided the training set and test set by experiments, which ensured the model was tested on the test set, and the test results are shown in Table 3. In addition, we compared the model with baseline regression prediction models such as kNN, LR, SVC, RF, GBDT, etc., and it can be seen that our proposed model outperforms most of the baseline models.

Table 3. This is the table of experimental results on WordCount, PageRank, Pi, and LR datasets.

Task Name	Model Name	MAPE	RMSE	MAE	R^2
WordCount	KNN	0.208	5.410	4.059	0.503
	LR	0.250	6.166	4.735	0.355
	SVC	1.000	22.758	21.425	−7.787
	RF	0.180	5.493	3.671	0.488
	GBDT	0.304	6.701	5.251	0.238
	MHAC	0.148	3.917	2.834	0.740
PageRank	KNN	0.283	1.715	1.361	−0.132
	LR	0.328	2.134	1.615	−0.752
	SVC	0.999	5.744	5.513	−11.694
	RF	0.284	1.921	1.389	−0.420
	GBDT	0.291	1.564	1.328	0.059
	MHAC	0.255	1.419	1.212	0.225
Pi	KNN	0.113	0.636	0.341	0.511
	LR	0.118	0.720	0.384	0.375
	SVC	1.000	2.624	2.461	−7.307
	RF	0.121	0.745	0.374	0.330
	GBDT	0.236	0.810	0.598	0.208
	MHAC	0.089	0.537	0.278	0.651
LR	KNN	0.164	5.964	4.135	0.872
	LR	0.334	10.917	8.336	0.572
	SVC	0.999	31.995	27.305	−2.677
	RF	0.114	5.304	3.300	0.899
	GBDT	0.583	13.999	10.617	0.296
	MHAC	0.098	3.994	2.647	0.943

As depicted in Table 3, the metrics MAPE, RMSE, MAE, and R^2 for our proposed MHAC model outperform those of the baseline prediction models across four benchmark datasets. Specifically, on the WordCount dataset, the MAPE of our MHAC model shows a

17.7% reduction compared to the RF model, and the RMSE witnesses a 27.6% decline against the kNN model. On additional datasets, our model consistently yielded the optimal metrics.

Our model is a multi-task prediction model. Therefore, we merged two or more benchmark datasets and conducted experiments on this combined dataset. The results, when compared with six benchmark prediction models, are presented in Table 4.

From Table 4, we can observe that our model essentially achieved optimal performance across all metrics in the mixed datasets, surpassing the majority of the benchmark models. In the dataset combined from PageRank and WordCount, the MAPE metric, when compared with other models, showed reductions of 87.9%, 11.6%, 66.4%, 88.0%, 83.7%, and 80.2%, respectively. Furthermore, when comparing the MAPE values of the models in Table 3, we find that the kNN, a simple model, performs exceptionally well in single-task predictions, achieving a low MAPE. However, under multi-task conditions, the predictive performance of kNN deteriorates significantly, with its MAPE exceeding 2. In contrast, the LR model exhibits better performance in multi-task scenarios compared to other models, indicating its stronger adaptability to complex data. Our model surpasses the majority of the models in multi-task metrics, suggesting that it is also well-suited for complex tasks and capable of handling Spark performance prediction tasks in multi-task settings.

To demonstrate the convergence of our model during the training process, we plotted the loss function. As shown in Figure 4, Figure 4a, Figure 4b, Figure 4c, and Figure 4d represent the loss function values on the WordCount, PageRank, Pi, and LR datasets, respectively. From Figure 4, we can observe that the loss function gradually decreases and converges as the number of training iterations increases.

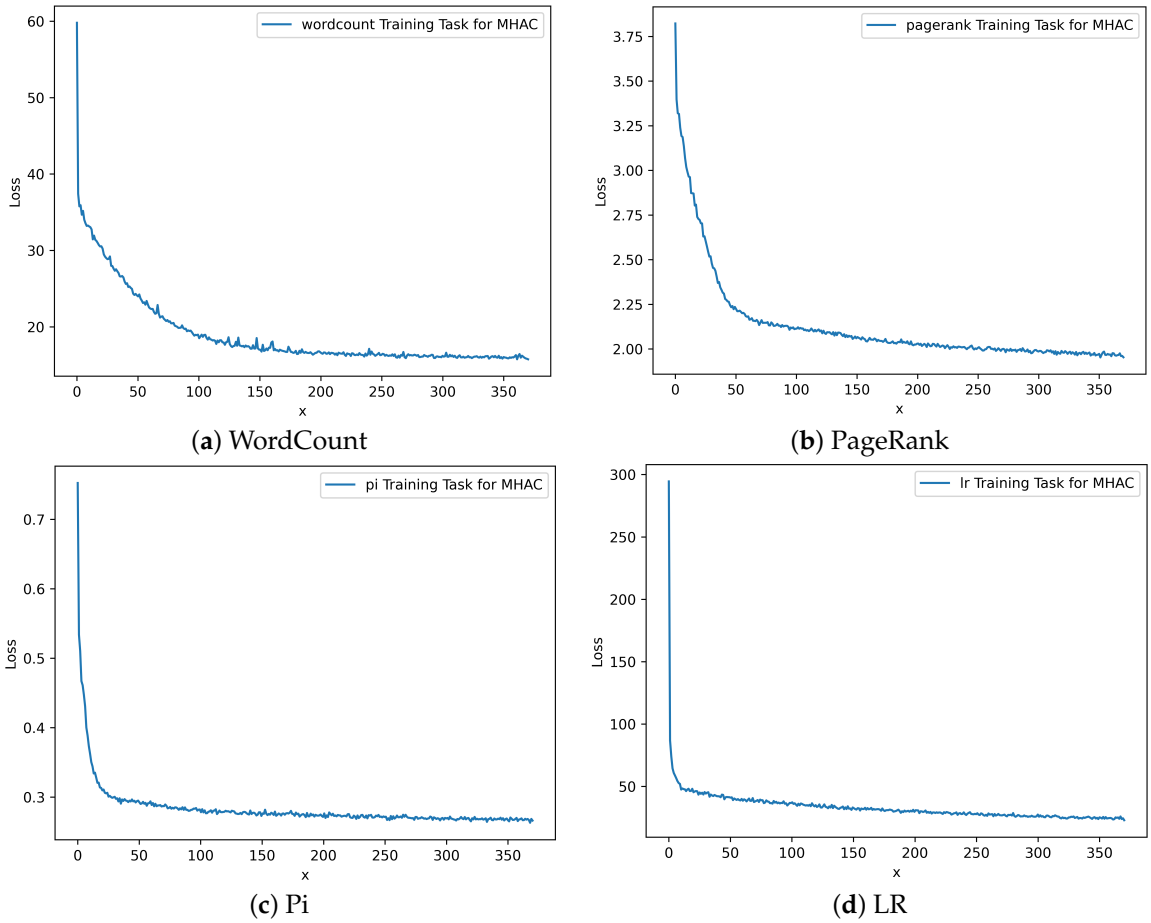


Figure 4. This is the loss function graph when the MHAC model is trained on four benchmark datasets.

Table 4. This is a table of experimental results for the PageRank–WordCount, PageRank–WordCount–LR, and Pi–PageRank–WordCount–LR datasets.

Task Name	Model Name	MAPE	RMSE	MAE	R^2
PageRank–WordCount	KNN	2.772	7.952	5.860	−0.722
	LR	0.377	3.444	2.575	−0.417
	SVC	0.992	8.759	6.325	−1.089
	RF	2.793	8.298	6.162	−0.875
	GBDT	2.052	5.930	4.014	0.043
	XGB	1.689	5.785	4.058	0.089
	MHAC	0.333	2.494	1.958	0.257
PageRank–WordCount–LR	KNN	2.772	7.952	5.860	−0.722
	LR	0.417	6.903	4.064	−0.306
	SVC	0.993	8.761	6.326	−1.089
	RF	2.796	8.362	6.162	−0.904
	GBDT	2.052	5.930	4.014	0.043
	XGB	1.689	5.785	4.058	0.089
	MHAC	0.477	5.213	3.515	0.255
Pi–PageRank–WordCount–LR	KNN	2.348	6.675	4.814	−0.213
	LR	1.917	6.516	4.363	0.115
	SVC	14.596	31.133	29.386	−25.389
	RF	2.056	5.765	4.082	0.095
	GBDT	2.023	5.673	4.033	0.124
	XGB	1.989	5.785	4.058	0.089
	MHAC	1.863	5.625	4.021	0.138

4.5. Ablation Experiments

In order to test the usefulness of each part of the proposed model, ablation experiments of the MHAC model are presented in this section. We analyze the MHAC model by removing the customized multi-head attention network, the CNN, the residual connection, and the PCA algorithms, respectively. We obtain four new models: MHAC-Att, MHAC-CNN, MHAC-Residual, and MHAC-PCA. Finally, we compare with the original model MHAC in four benchmark applications. The experimental results are shown in Figure 5.

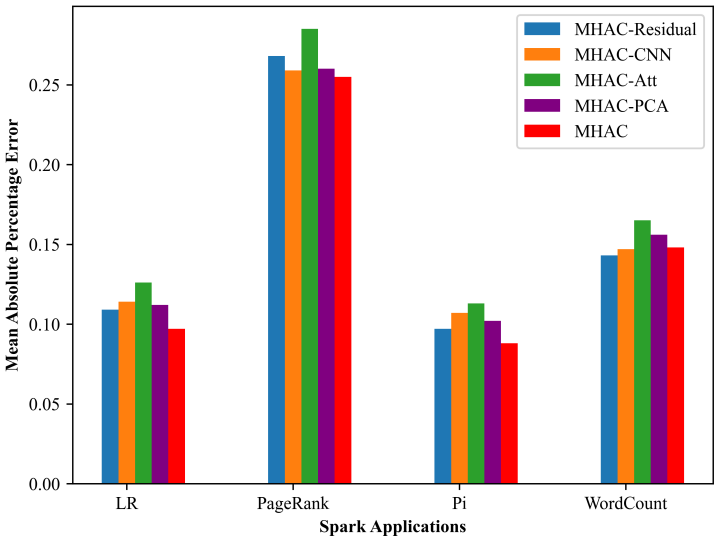


Figure 5. This is the MAPE result for the benchmark dataset of ablation experiments.

The experimental comparison results are shown in Figure 5. In the application of WordCount, the MAPE of the MHAC model is 0.148, and that of MHAC-CNN is 0.147, which is 0.001 different from that of MHAC. In addition, our model outperforms MHAC-Residual, MHAC-CNN, MHAC-PCA, and MHAC-Att in the MAPE of four benchmark

applications. For example, in LR, the MHAC-Residual is 0.109, the MHAC-CNN is 0.114, the MHAC-Att is 0.126, the MHAC-PCA is 0.112, and the MAPE of MHAC is 0.098. The metric of MHAC is lower than that of other models. In addition, the MAPE metric of MHAC is 10.53% lower than MHAC-Att in PageRank.

4.6. Baseline Dataset Comparison Experiments

In order to be able to better evaluate our model, we used the baseline dataset of regression predictions of California house prices for our experiments and compared it with the baseline model. The experimental results MAPE and RMSE are shown in Figure 6 and Figure 7, respectively.

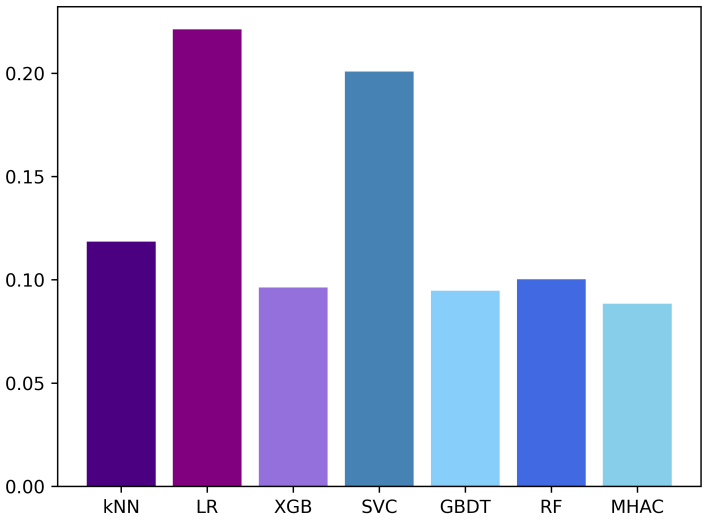


Figure 6. This is the MAPE result for the benchmark dataset of California housing prices.

As shown in Figures 6 and 7, in the experiments on the baseline dataset California house prices, the errors of the LR and SVC models are significantly higher than the other models; in the MAPE metrics in Figure 6, our model MHAC significantly outperforms the other baseline models, and in the RMSE metrics in Figure 7, the metrics of our model are 0.4692, the second model is GBDT with a metric of 0.4721, and our model also achieves better metrics in RMSE. In summary, our model proves that the prediction performance is better than the baseline regression prediction model through the comparison experiment.

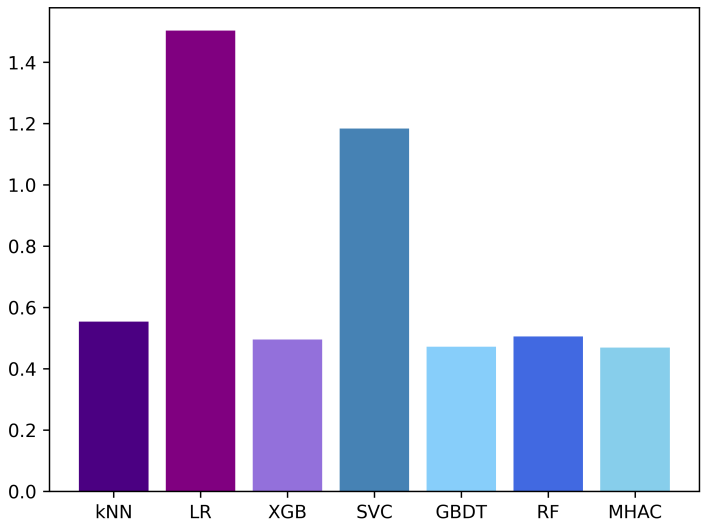


Figure 7. This is the RMSE result for the benchmark dataset of California housing prices.

5. Conclusions

In this paper, we present an approach that combines multi-head attention networks and CNN models for spark performance prediction. We first introduce the basic principle of a multi-head attention network. It can effectively capture complex dependencies in data and extract deep features to help improve the accuracy of prediction models. Then, we combine the multi-head attention network with the CNN to utilize the advantage of the CNN in processing local features in grid-structured data to further improve the performance of the prediction model.

We conducted comprehensive evaluations of the proposed model using datasets of various sizes and multiple common Spark applications. Experiments were also performed on mixed datasets. The experimental results indicate that the model can effectively predict Spark performance. Compared to traditional methods, it offers higher accuracy and better generalizability.

However, there is still some potential room for improvement in our model. For example, the structure of multi-head attention networks can be further optimized to try different attention mechanisms to adapt to different data characteristics. Meanwhile, other types of neural network structures, such as recurrent neural networks, can be explored for processing data with sequential characteristics. In addition, more feature engineering and data enhancement techniques can be considered to be introduced to further improve the performance of the model.

In summary, our work provides a new solution in the field of Spark performance prediction and also provides a valuable reference for other research on similar tasks. Future work can further improve and extend our model, and validate and apply it in a wider range of application scenarios. We believe this research will contribute to improving the performance and efficiency of distributed computing systems.

Author Contributions: C.S. and C.C. proposed the idea of the paper. C.S. and G.R. helped manage the annotation group and helped clean the raw annotations. C.C. conducted all experiments and wrote the manuscript. C.S., G.R. and C.C. revised and improved the text. C.C. is the person in charge of this project. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data presented in this study are available on request from the corresponding author. The data are not publicly available due to privacy issues.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Ding, Z.; Zhang, C. A method of classification-based Spark job performance modeling. In Proceedings of the 2nd International Conference on Applied Mathematics, Modelling, and Intelligent Computing (CAMMIC 2022), Kunming, China, 25–27 March 2022; Volume 12259, pp. 1310–1315.
2. Awan, M.J.; Khan, M.A.; Ansari, Z.K.; Yasin, A.; Shehzad, H.M.F. Fake profile recognition using big data analytics in social media platforms. *Int. J. Comput. Appl. Technol.* **2022**, *68*, 215–222. [CrossRef]
3. Ameer, S.; Shah, M.A. Exploiting big data analytics for smart urban planning. In Proceedings of the 2018 IEEE 88th Vehicular Technology Conference (VTC-Fall), Chicago, IL, USA, 27–30 August 2018; pp. 1–5.
4. Agafonov, A.; Yumaganov, A. Short-term traffic flow forecasting using a distributed spatial-temporal k nearest neighbors model. In Proceedings of the 2018 IEEE International Conference on Computational Science and Engineering (CSE), Bucharest, Romania, 29–31 October 2018; pp. 91–98.
5. Shen, C.; Tong, W.; Hwang, J.N.; Gao, Q. Performance modeling of big data applications in the cloud centers. *J. Supercomput.* **2017**, *73*, 2258–2283. [CrossRef]
6. Cheng, G.; Ying, S.; Wang, B.; Li, Y. Efficient performance prediction for apache spark. *J. Parallel Distrib. Comput.* **2021**, *149*, 40–51. [CrossRef]

7. Wang, K.; Khan, M.M.H. Performance prediction for apache spark platform. In Proceedings of the 2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems, New York, NY, USA, 24–26 August 2015; pp. 166–173.
8. Gao, Z.; Wang, T.; Wang, Q.; Yang, Y. Execution Time Prediction for Apache Spark. In Proceedings of the 2018 International Conference on Computing and Big Data, Charleston, SC, USA, 8–10 September 2018; pp. 47–51.
9. Shah, S.; Amannejad, Y.; Krishnamurthy, D.; Wang, M. Quick execution time predictions for spark applications. In Proceedings of the 2019 15th International Conference on Network and Service Management (CNSM), Halifax, NS, Canada, 21–25 October 2019; pp. 1–9.
10. Al-Sayeh, H.; Hagedorn, S.; Sattler, K.U. A gray-box modeling methodology for runtime prediction of apache spark jobs. *Distrib. Parallel Databases* **2020**, *38*, 819–839. [CrossRef]
11. AlQuwaiee, H.; Wu, C. On Performance Modeling and Prediction for Spark-HBase Applications in Big Data Systems. In Proceedings of the ICC 2022-IEEE International Conference on Communications, Seoul, Republic of Korea, 16–20 May 2022; pp. 3685–3690.
12. Singhal, R.; Singh, P. Performance assurance model for applications on SPARK platform. In Proceedings of the Performance Evaluation and Benchmarking for the Analytics Era: 9th TPC Technology Conference, TPCTC 2017, Munich, Germany, 28 August 2017; pp. 131–146.
13. Huang, X.; Zhang, H.; Zhai, X. A Novel Reinforcement Learning Approach for Spark Configuration Parameter Optimization. *Sensors* **2022**, *22*, 5930. [CrossRef] [PubMed]
14. Azhir, E.; Hosseinzadeh, M.; Khan, F.; Mosavi, A. Performance Evaluation of Query Plan Recommendation with Apache Hadoop and Apache Spark. *Mathematics* **2022**, *10*, 3517. [CrossRef]
15. Yadav, M.L. Query Execution Time Analysis Using Apache Spark Framework for Big Data: A CRM Approach. *J. Inf. Knowl. Manag.* **2022**, *21*, 2250050. [CrossRef]
16. Lin, J.C.; Lee, M.C.; Yu, I.C.; Johnsen, E.B. A configurable and executable model of Spark Streaming on Apache YARN. *Int. J. Grid Utility Comput.* **2020**, *11*, 185–195. [CrossRef]
17. Matteussi, K.J.; Dos Anjos, J.C.; Leithardt, V.R.; Geyer, C.F. Performance evaluation analysis of spark streaming backpressure for data-intensive pipelines. *Sensors* **2022**, *22*, 4756. [CrossRef] [PubMed]
18. Ahmed, N.; Barczak, A.L.; Rashid, M.A.; Susnjak, T. An enhanced parallelisation model for performance prediction of apache spark on a multinode hadoop cluster. *Big Data Cogn. Comput.* **2021**, *5*, 65. [CrossRef]
19. Zhu, C.; Han, B.; Zhao, Y. A comparative performance study of spark on kubernetes. *J. Supercomput.* **2022**, *78*, 13298–13322. [CrossRef]
20. Prasad, B.R.; Agarwal, S. Performance analysis and optimization of spark streaming applications through effective control parameters tuning. In *Progress in Intelligent Computing Techniques: Theory, Practice, and Applications, Proceedings of the ICACNI 2016, Rourkela, Odisha, India, 22–24 September 2016*; Springer: Berlin, Germany, 2018; Volume 2, pp. 99–110.
21. Dong, L.; Li, P.; Xu, H.; Luo, B.; Mi, Y. Performance Prediction of Spark Based on the Multiple Linear Regression Analysis. In Proceedings of the Parallel Architecture, Algorithm and Programming: 8th International Symposium, PAAP 2017, Haikou, China, 17–18 June 2017; pp. 70–81.
22. Maros, A.; Murai, F.; da Silva, A.P.C.; Almeida, J.M.; Lattuada, M.; Gianniti, E.; Hosseini, M.; Ardagna, D. Machine learning for performance prediction of spark cloud applications. In Proceedings of the 2019 IEEE 12th International Conference on Cloud Computing (CLOUD), Milan, Italy, 8–13 July 2019; pp. 99–106.
23. Ye, G.; Liu, W.; Wu, C.Q.; Shen, W.; Lyu, X. On Machine Learning-based Stage-aware Performance Prediction of Spark Applications. In Proceedings of the 2020 IEEE 39th International Performance Computing and Communications Conference (IPCCC), Austin, TX, USA, 6–8 November 2020; pp. 1–8.
24. Kordelas, A.; Spyrou, T.; Voulgaris, S.; Megalooikonomou, V.; Deligiannis, N. KORDI: A Framework for Real-Time Performance and Cost Optimization of Apache Spark Streaming. In Proceedings of the 2023 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), Raleigh, NC, USA, 23–25 April 2023; pp. 337–339.
25. Ahmed, N.; Barczak, A.L.; Rashid, M.A.; Susnjak, T. Runtime prediction of big data jobs: Performance comparison of machine learning algorithms and analytical models. *J. Big Data* **2022**, *9*, 67. [CrossRef]
26. Al-Sayeh, H.; Memishi, B.; Jibril, M.A.; Paradies, M.; Sattler, K.U. Juggler: Autonomous cost optimization and performance prediction of big data applications. In Proceedings of the 2022 International Conference on Management of Data, Philadelphia, PA, USA, 12–17 June 2022; pp. 1840–1854.
27. Lavanya, K.; Venkatanarayanan, S.; Bhoraskar, A.A. Real-Time Weather Analytics: An End-to-End Big Data Analytics Service Over Apach Spark With Kafka and Long Short-Term Memory Networks. *Int. J. Web Serv. Res. (IJWSR)* **2020**, *17*, 15–31. [CrossRef]
28. Ye, K.; Kou, Y.; Lu, C.; Wang, Y.; Xu, C.Z. Modeling application performance in docker containers using machine learning techniques. In Proceedings of the 2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS), Singapore, 11–13 December 2018; pp. 1–6.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.