Dean&Francis
ISSN 2959-6157

The Study of Machine Learning Inference Tasks Based on Serverless Computing Platforms

Chang Liu[1, *]

[1]Department of Information Technology, Chang'an University, Shaanxi, China

*Corresponding author: iliylv16233@gmail.com

Abstract:

This study proposes a distributed inference method based on the ResNet50 model, aiming to improve inference efficiency and resource utilization by dividing the model into multiple sub-models. Specifically, the model is divided into the initial convolutional layer, four stages of residual blocks, and the subsequent global average pooling and fully connected layers. Each sub-model independently handles specific tasks, allowing for parallel execution on different computing devices, thereby accelerating the overall inference process. This partitioning strategy effectively addresses high-concurrency requests, enhancing the system's response speed. Additionally, it enables dynamic scaling of resources based on workload demands, which is crucial for real-time applications. The implementation of distributed inference also makes the model more flexible, adapting to various computing resources and application scenarios. Experimental results indicate that this method significantly enhances inference efficiency while maintaining model performance, providing new ideas and solutions for the practical application of deep learning models. These findings underscore the potential of distributed architectures in advancing the deployment of complex neural networks across diverse environments.

Keywords: Serverless; ResNet50; Model Partitioning; Parallel Execution; Inference Efficiency.

1. Introduction

In modern deep learning applications, inference efficiency and resource utilization are crucial factors for achieving high-performance systems. In recent years, researchers worldwide have made significant progress in optimizing machine learning inference efficiency on serverless computing platforms. In 2019, Zhang Chengliang and colleagues proposed a method that utilizes cloud computing services to achieve low-cost, high-benefit machine learning inference services that meet service quality requirements [1]. This research employed model compression and dynamic resource scheduling strategies to enhance

ISSN 2959-6157

service quality while ensuring cost-effectiveness, optimizing the performance of enterprise-level inference services. Similarly, Bhattacharjee et al. presented an efficient and scalable serverless system for deep learning prediction tasks in a 2020 paper [2]. This study explored effective resource allocation and scheduling management, as well as optimizing the inference efficiency of deep learning models. On the cloud service platform front, Alibaba Cloud's Model Service offers comprehensive support, enabling developers to quickly deploy trained models as service endpoints via its API and SDK, while utilizing traffic distribution strategies for automatic scaling [3]. Google Cloud Functions, on the other hand, allows users to run code without managing infrastructure, providing high flexibility for machine learning applications based on serverless computing platforms [4].

Existing research has focused on improving the efficiency of public cloud applications, such as achieving smarter resource utilization in serverless environments and exploring technologies like dynamic task scheduling and predictive model optimization to reduce latency and enhance system efficiency [5]. Additionally, researchers have made discussions about how to support various deep learning frameworks and tools to meet the needs of different models [6]. These experimental results provide valuable theoretical foundations and practical experience for future research, as well as guidance for future directions.

This paper focuses on the increasing demand for intelligent applications and identifies that traditional single-model inference methods struggle to handle the challenges of high concurrent requests. To address this issue, the study proposes a distributed inference method based on the ResNet50 model. This approach leverages model partitioning for parallel processing, thereby accelerating the inference process and enhancing system response speed [7]. Each sub-model focuses on specific tasks and can flexibly adapt to different computing devices and resource configurations. This distributed inference strategy not only effectively improves inference efficiency but also provides new ideas and solutions for the practical application of deep learning models. Experimental results validate that this method enhances inference efficiency while maintaining model performance, laying a foundation for future research and applications.

2. Algorithm Design

2.1 Overview

In the process of optimizing the inference performance of deep learning models on serverless computing platforms, the key lies in improving the consumption of computational resources and time. In traditional single-node inference methods, large models often face issues such as excessive inference latency and limited computational resources, resulting in low actual inference performance [8]. To address this problem, this paper proposes a method of model partitioning and distributed inference based on the large deep learning model ResNet50. By dividing the model into multiple sub-models and deploying them in different serverless functions, the method aims to parallelize inference tasks, thereby optimizing inference performance.

2.2 Model Partition Strategy

2.2.1 Details of ResNet50

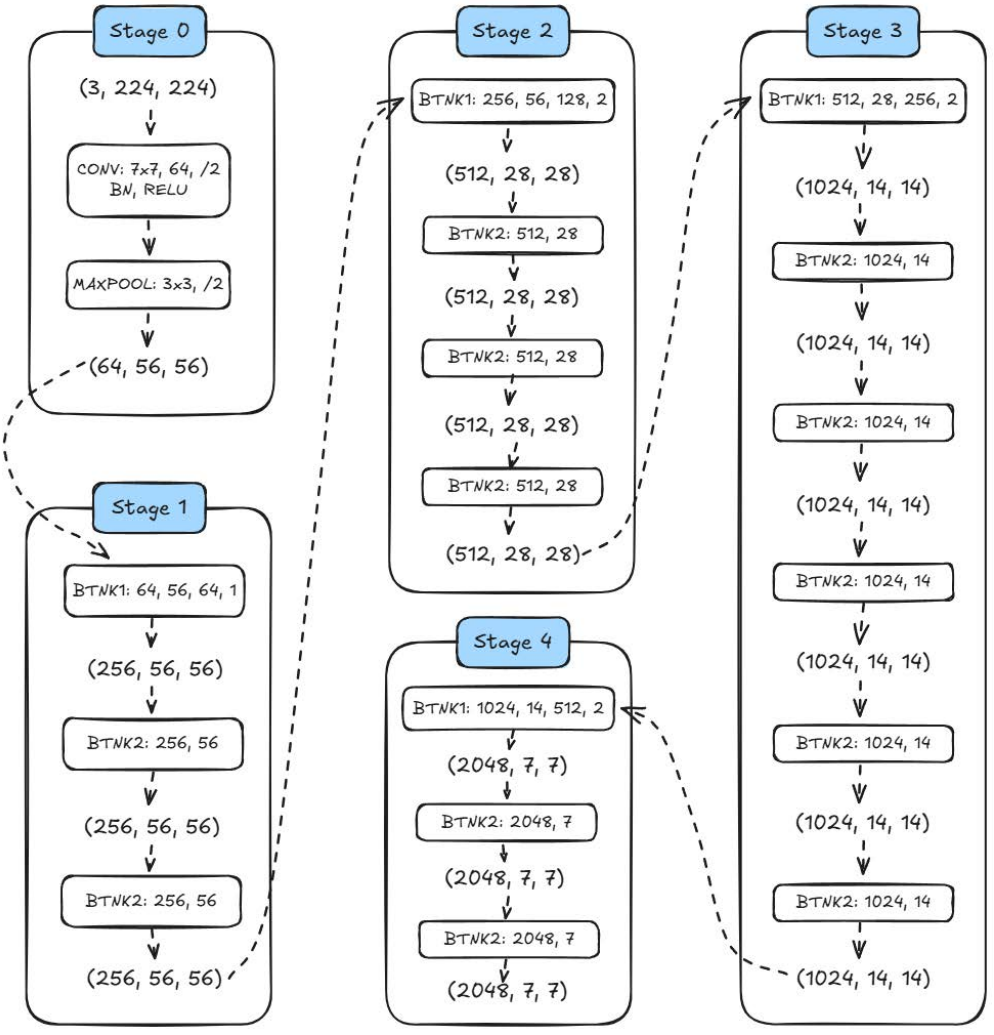

Fig. 1 Structure of ResNet50

ResNet50 is a deep residual network [9], as illustrated in Fig. 1. The diagram illustrates a typical ResNet50 model architecture. The model consists of five main stages (Stage 0 to Stage 4), each containing a series of convolutional layers and residual blocks. First, the input image size is (3,224,224). After the initial CONV layer (7x7 convolution and 3x3 max pooling), the output size becomes (64,56,56). This is Stage 0. Then, the model enters Stage 1, which contains a single BTKW1 residual block. BTKW stands for "Bottleneck Transformation with Kernel Weights", which is a type of residual transformation with a bottleneck structure. The input and output sizes of this block are (64,56,56) and (256,56,56), respectively. Next is Stage 2, which includes three parallel BTKW2 residual blocks. Each BTKW2 block has an input and output size of (512,28,28). These parallel residual blocks can enhance the model's representational capacity. Afterward, there are Stage 3 and Stage 4, containing 4 and 6 parallel BTKW residual blocks, respectively. The input and output sizes of these blocks continue to evolve, reflecting the transformation of feature maps. The overall model structure utilizes residual connections (dashed arrows) to enhance gradient flow, increasing the model's depth and performance. The numbers in parentheses indicate the channel, height, and width dimensions of each layer. This residual block-based deep learning model architecture is highly powerful and has demonstrated excellent performance in various computer vision tasks. This design also allows each block to be processed independently of the others. As a result, the modular nature of the residual structure enables them to be partitioned and combined effectively.

2.2.2 Partition Concept

The ResNet50 model consists of five stages, allowing the model to be partitioned into five sub-models by stage. The principles that should be followed when partitioning are:

1. Balanced Computational Load: Ensure that the computational workload of each sub-model is as balanced as possible to prevent any single sub-model from becoming

ISSN 2959-6157

a bottleneck.

2. Minimize Data Transfer: The intermediate results passed between sub-models should be kept as small as possible to reduce network transmission latency.

2.2.3 Partition Scheme

Based on the hierarchical structure of ResNet50, it can be partitioned into the following five sub-models:

1. Initial Convolutional Layer Sub-model

a) Composition: Input Layer, Initial Convolutional Layer, Batch Normalization Layer, ReLU Activation Layer, Max Pooling Layer.

b) Function: Extract initial features from the image.

2. First Stage Residual Block Sub-model

a) Composition: The two residual blocks of the first stage, layer1.

b) Function: Extract deeper features to construct more complex representations.

3. Second Stage Residual Block Sub-model

a) Composition: The two residual blocks of the second stage, layer2.

b) Function: Further enhance features.

4. Third Stage Residual Block Sub-model

a) Composition: The two residual blocks of the third stage, layer3.

b) Function: Extract high-level features.

5. Fourth Stage Residual Block and Subsequent Layers Sub-model

a) Composition: The two residual blocks of the fourth stage, layer4, global average pooling layer, and fully connected layer.

b) Function: Complete the final feature aggregation and classification task.

2.3 Distributed Inference Strategy

2.3.1 Preliminary Scheme

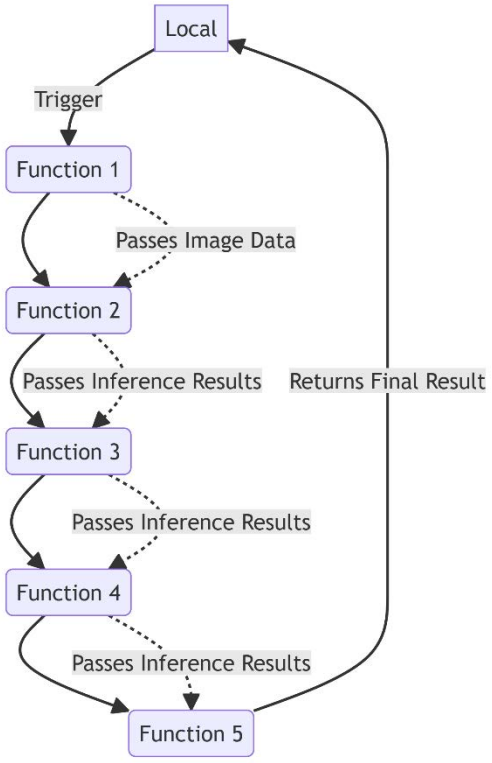

Fig. 2 Distributed Reasoning Process

According to the model partitioning scheme mentioned earlier, the sub-model deployment can be initially designed as a pipeline scheme, where the output of each sub-model directly serves as the input for the next. As shown in Fig. 2, a trigger first calls Function 1 locally and passes image data to it. Function 1 then triggers Function 2, passing the inference results from Function 1, and so on, until Function 5 returns the classification results to the local environment. Each sub-model only needs to focus on how to receive and process data from the previous sub-model, without the need to communicate with other sub-models. This maximizes the potential for parallel processing while reducing complex dependencies.

This pipelined deployment can improve the overall inference efficiency, as the individual sub-models can execute in parallel, thereby reducing the total time for the final classification result to be returned to the local environment. In contrast, if a global model deployment is adopted, each inference task would require loading the entire model, which may result in low resource utilization and longer inference times. The pipeline scheme, by breaking down the large model into smaller sub-models, effectively reduces the resource requirements for individual functions, and leverages the parallelism between functions to optimize the overall performance.

2.3.2 Advanced Scheme

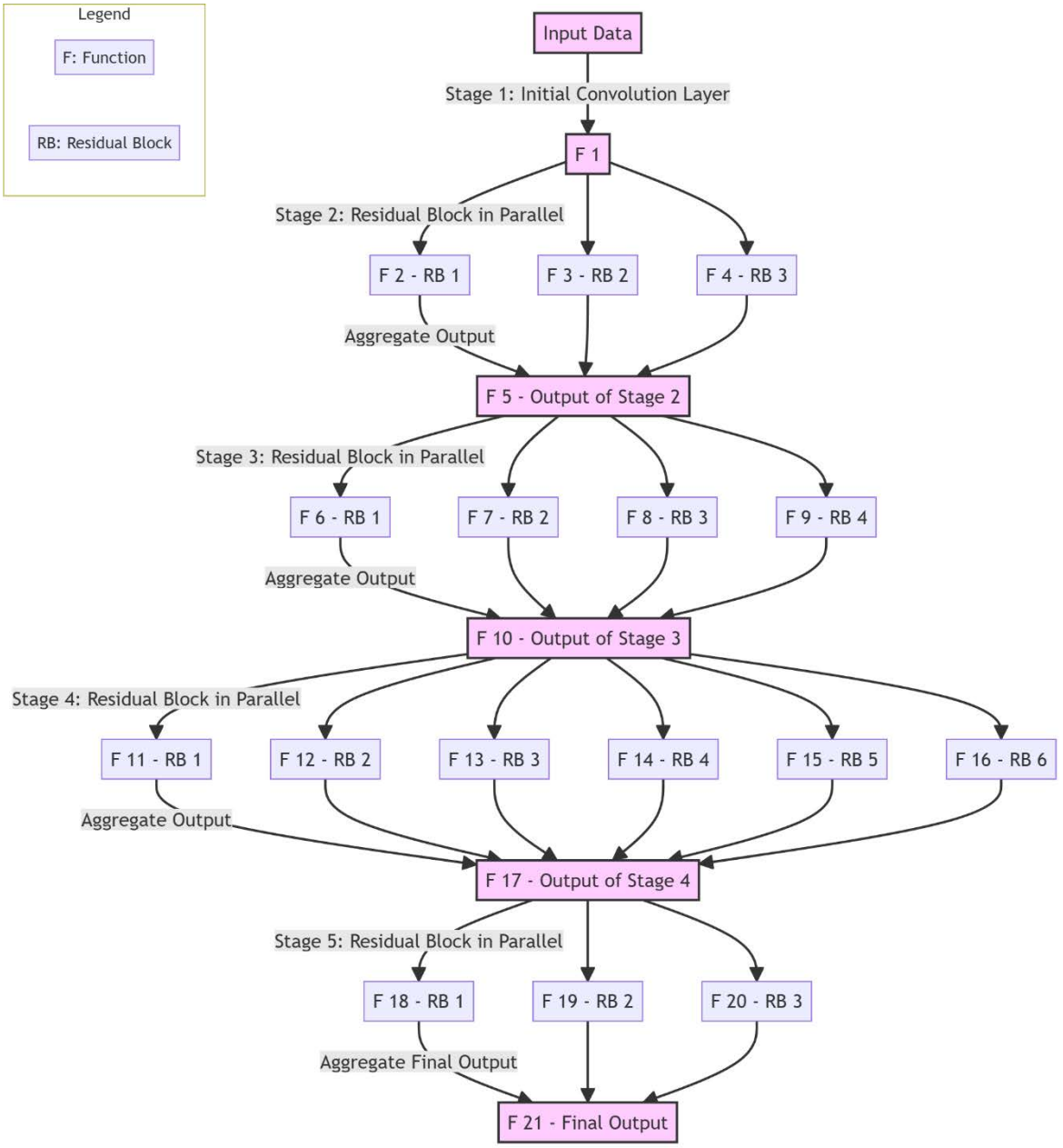

Fig. 3 Distributed Reasoning Process

The structure of ResNet50 has certain serial dependencies; however, the residual blocks within each stage possess high parallel potential, especially since the computations within each residual block are independent. Therefore, these residual blocks can be distributed across different functions for processing. Additionally, for more complex convolution operations, further parallelization can be achieved at the layer level. The structure of this scheme is illustrated in Fig. 3, with the specific parallel steps outlined as follows:

1. Stage 1 (Initial Convolution Layer and Pooling Layer): Since Stage 1 is relatively simple, it can be processed directly within a single cloud function. The input data is preprocessed to generate initial feature maps, which serve as input for Stage 2.

2. Stages 2-5 (Multiple Residual Blocks):

a) Residual Blocks Executed in Parrallel: There are essentially no dependencies between the residual blocks within each stage, allowing for parallel processing. Each residual block in Stages 2-5 is individually deployed to a function for parallel computation.

b) Final Aggregation: The last residual block of each stage outputs the feature maps to the first residual block of the next stage.

ISSN 2959-6157

3. Experiments and Analysis

3.1 Experiment Design

3.1.1 Experiment Objectives

This experiment aims to evaluate the improvement in inference performance on Serverless computing platforms by comparing key metrics such as average inference time, average resource utilization, and average network latency for the model partitioning and distributed inference schemes.

3.1.2 Experiment Environment

The experiment is set up on the Aliyun Function Compute platform. Each function is configured as an independent serverless function, with a memory size of 512MB and a CPU core count of 0.35. A consistent Python runtime environment is used to ensure that the dependency library versions are uniform. For testing, 100 images are randomly selected from the ImageNet validation set. Parameter passing between functions occurs over the server‘s internal network, while local calls and result returns are transmitted via the external access address provided by the server. The comparison includes a baseline scheme, where the complete ResNet50 model is deployed in a single serverless function for inference, and an optimized scheme that employs model partitioning and distributed inference methods. The real-time monitoring data provided by the Alibaba Cloud platform includes inference time and resource utilization, while network latency needs to be extracted from the results returned by the functions.

3.2 Experiment Procedure

3.2.1 Comparison Metrics

The evaluation metrics for the experiment include: average inference time, which represents the total time required to process a single image; average cold start time, which is the initialization time during the first function call; average CPU utilization; average memory usage; and average network latency, which refers to the average network delay when passing parameters between functions. Together, these metrics provide a comprehensive understanding of the inference performance under different schemes.

3.2.2 Experiment Scheme

First, the ResNet50 deep learning model needs to be pre-downloaded to the Alibaba Cloud Function Compute platform. Then, three different experiments are designed based on three schemes: single-function inference (Baseline Scheme), inference after preliminary model partitioning (Preliminary Scheme), and inference after further parallel processing (Advanced Scheme).

The Baseline Scheme involves directly deploying the inference function to the cloud platform, then locally invoking the trigger and monitoring relevant data once the results are returned.

For the Preliminary Scheme, five different functions need to be created on the platform. In each function, the model is chunk-loaded according to the partitioning scheme using PyTorch‘s loading function. The functions interact with each other and return results according to a pipelined flowchart. Finally, the first cloud function is called locally via an HTTP trigger, and relevant data is monitored after the final result is returned.

The Advanced Scheme builds on the Preliminary Scheme by distributing the residual blocks of each stage into additional functions for parallel processing, which allows for simultaneous execution of tasks, thereby enhancing inference efficiency.

The final result is the average of multiple experimental results, excluding all outliers.

3.2.3 Experiment Results

Based on the updated data presented in Table 1, the average inference time for the non-partitioned inference (Baseline Scheme) was 379 ms. In contrast, the pipeline partitioned inference (Preliminary Scheme) increased this time to 482 ms. However, the Advanced Scheme, which combines pipeline partitioning with parallel processing, significantly reduced the average inference time to 313 ms, demonstrating the effectiveness of parallel execution in optimizing inference speed.

The average cold start time was 947 ms for the Baseline Scheme, which increased to 1023 ms for the Preliminary Scheme, indicating that model partitioning may introduce additional initialization overhead. The Advanced Scheme, however, reduced the cold start time to 731 ms, suggesting that the parallelization approach helps mitigate some of the initialization delays.

In terms of CPU utilization, the Baseline Scheme achieved an average utilization of 92.1%, while the Preliminary Scheme showed a slight decrease to 89.3%. The Advanced Scheme, on the other hand, increased the average CPU utilization to 93.5%. This suggests that parallel processing may improve resource efficiency in inference tasks; however, the changes observed in the data are not particularly significant, indicating that other factors, such as CPU performance bottlenecks, may be influencing the results.

Regarding memory usage, the Baseline Scheme consumed an average of 1481.2 MB, which is significantly higher than the 684.8 MB used in the Preliminary Scheme. The Advanced Scheme showed a slight increase in memory

usage to 779.4 MB, reflecting the additional overhead from parallel execution while still remaining lower than the Baseline Scheme.

Lastly, the average network delay for parameter transmission was 50 ms in the Baseline Scheme, which increased to 179 ms for the Preliminary Scheme and further to 223 ms for the Advanced Scheme. This indicates that while partitioning and parallelization improve inference speed, they also introduce additional latency due to the need for inter-function communication over the network, which could impact overall performance.

Table 1 Experimental Results

Metrics	Baseline Scheme	Preliminary Scheme	Advanced Scheme
Average Inference Time (ms)	379	482	313
Initial Load Time (ms)	947	1023	731
CPU Utilization (%)	92.1	89.3	93.5
Memory Usage (MB)	1481.2	684.8	779.4
Network Latency (ms)	50	179	223

3.3 Analysis and Discussion

The proposed approach offers several advantages, including notable performance improvements, as concurrent inference effectively enhances service efficiency. The distributed architecture also provides better resilience and fault tolerance; if one function fails, it does not result in a complete service outage. Furthermore, the method reduces the resource configuration requirements for each function, leading to lower costs for cloud service usage. Although the average inference time increased in the pipeline partitioned scheme, the advanced scheme demonstrated significant improvements in inference speed, suggesting that parallel processing can optimize resource utilization.

Despite these advantages, potential issues remain. Network latency may introduce additional delays, particularly during cross-region deployments, as indicated by the increased average network delay in the pipeline partitioned and advanced schemes. This challenge can be mitigated by optimizing function deployment locations and employing more efficient serialization methods. Additionally, the increased complexity of the system architecture requires robust monitoring and management mechanisms to ensure smooth operation, especially given the varying cold start times and the need for effective resource allocation to prevent CPU performance bottlenecks.

4. Conclusion

This study proposes a model partitioning and distributed inference method aimed at optimizing deep learning inference tasks on serverless computing platforms. The experiments involved splitting a large model into multiple sub-models and deploying them as independent serverless functions, allowing for parallelization of the inference process and efficient resource utilization. The experimental results demonstrate that this approach significantly reduces average inference time and resource consumption while maintaining high CPU utilization.

However, several potential issues have been identified. Future research should focus on optimizing cross-region deployment strategies to minimize network latency, which has been shown to increase in distributed settings. This includes developing intelligent function placement algorithms and exploring more efficient serialization methods to enhance overall performance. Additionally, as system complexity increases, robust monitoring and management solutions will be crucial for ensuring smooth operation. Investigating automated tools and adaptive frameworks will help improve user experiences and operational efficiency across various applications.

References

[1] Zhang C, Yu M, Wang W, et al. Enabling cost-effective, slow machine learning inference serving on public cloud. IEEE Transactions on Cloud Computing, 2020, 10(3): 1765-1779.

[2] Bhattacharjee A. Algorithms and Techniques for Automated Deployment and Efficient Management of Large-Scale Distributed Data Analytics Services. Vanderbilt University, 2020.

[3] Wen X, Zeng T, Li C, et al. Research on Model Inference Service Switching Method for Serverless Computing. Computer Engineering and Science, 2024, 46(07): 1210.

[4] Chaitanya K T. EXPLORING SERVER-LESS COMPUTING FOR EFFICIENT RESOURCE MANAGEMENT IN CLOUD ARCHITECTURES. Journal of Science Technology and Research (JSTAR), 2023, 4 (1):77-83.

[5] Mampage A, Karunasekera S, Buyya R. A holistic view on resource management in serverless computing environments:

ISSN 2959-6157

Taxonomy and future directions. ACM Computing Surveys (CSUR), 2022, 54(11s): 1-36.
[6] Shafiei H, Khonsari A, Mousavi P. Serverless computing: a survey of opportunities, challenges, and applications. ACM Computing Surveys, 2022, 54(11s): 1-32.
[7] Yu M, Jiang Z, Ng H C, et al. Gillis: Serving large neural networks in serverless functions with automatic model partitioning//2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS). IEEE, 2021: 138-148.
[8] Hassan H B, Barakat S A, Sarhan Q I. Survey on serverless computing. Journal of Cloud Computing, 2021, 10: 1-29.
[9] Koonce B. ResNet 50. In: Convolutional Neural Networks with Swift for Tensorflow. Apress, Berkeley, CA. https://doi.org/10.1007/978-1-4842-6168-2_6, 2021.