STREAMLINING SOFTWARE RELEASE PROCESS AND RESOURCE MANAGEMENT FOR MICROSERVICES BASED ARCHITECTURE ON MULTI-CLOUD

TMP-23-193

M.R.A. Fadhil

IT20784720

B.Sc. (Hons) in Information Technology Specializing in Software Engineering

Department of Computer Science and Software Engineering Sri Lanka Institute of Information Technology

Sri Lanka

September 2023

STREAMLINING SOFTWARE RELEASE PROCESS AND RESOURCE MANAGEMENT FOR MICROSERVICES BASED ARCHITECTURE ON MULTI-CLOUD

TMP-23-193

M.R.A. Fadhil

IT20784720

Dissertation submitted in partial fulfillment of the requirements for the Bachelor of

Science specializing in Software Engineering

Department of Computer Science and Software Engineering Sri Lanka Institute of Information Technology

Sri Lanka

September 2023

## Declaration

I declare that this is my work, and this proposal does not incorporate without acknowledgement any material previously submitted for a degree or diploma in any other university or institute of higher learning. To the best of my knowledge and belief, it does not contain any previously published material written by another person except where the acknowledgement is made in the text.

Signature

|  |  |  |
| --- | --- | --- |
| IT20784720 | Fadhil M.R. A |  |

The above candidates are researching the undergraduate Dissertation under my supervision.

Signature of the Supervisor: ………………………………… Date: ………10/09/2022…………………………

# ABSTRACT

The deployment of microservices on Kubernetes clusters has become a popular approach for building scalable and resilient distributed systems. However, managing the resources of these microservices can be challenging, especially when dealing with varying workloads and resource demands. To address this challenge, we propose a resource management framework that combines machine learning and

reinforcement learning algorithms to predict the resource load of microservices and automate the deployment strategy based on the prediction.

Our framework utilizes Docker and service mesh to deploy microservices on an Azure Kubernetes cluster, and uses Prometheus, Grafana, and Kiali to collect and analyse the time-series data of resource utilization. We propose a machine learning algorithm that predicts the resource load of microservices based on the collected data, and a reinforcement learning algorithm that optimizes the deployment strategy based on the prediction.

To evaluate the performance of our framework, we conducted concurrent performance tests using Gatling, JMeter, and Locust with large loads, and also performed throttling, error injection, and chaos testing. Our results show that our framework can effectively manage the resources of microservices and achieve better performance compared to traditional deployment strategies.

In conclusion, our proposed framework provides a promising approach for managing the resources of microservices deployed on Kubernetes clusters and has the potential to improve the scalability and resilience of distributed systems.

Keywords: Kubernetes, service mesh, monitoring, Docker, Performance , resilience

, machine learning, optimization

# TABLE OF CONTENT

[Declaration 3](#_TOC_250010)

[Abstract 4](#_TOC_250009)

[Table Of Content 5](#_TOC_250008)

[List Of figures 6](#_TOC_250007)

[List Of Tables 7](#_TOC_250006)

[List Of Abbreviations 8](#_TOC_250005)

1. [Introduction 9](#_TOC_250004)

2. Research Problem 14

3. [Objectives](#_TOC_250002)

* 1. [Main](#_TOC_250001) Objective 15
  2. Specific Objective 15

4. Methodology

4.1 Requirement Gathering 16

4.2 Software Solution and System Analysis 17

4.3 Project Requirements 25

4.4 Feasibility Study 26

4.5 TimeLine 27

4.6 Commercialization 28

5.Results and Discussion

5.1 Microservice deployment 29

5.2 Prediction model selection 30

5.3 Data gathering 37

5.4 Results and findings 37

6.Conclusion 38

7.Further Improvements 39

8.References 40

9.APPENDIX 42

## List of Figures

Figure 1: Architecture of ARIMA Model 10

Figure 2: Architecture Prometheus matrices gathering for prediction 11

Figure 3: Architecture of DScaler 12

Figure 4: System Overview Diagram 17

Figure 5: Metric Diagram 19

Figure 6: Optimization Diagram 20

Figure 7: Bi-Lstm-CNN Prediction Layers Diagram 21

Figure 8: Istio Architecture Diagram 22

Figure 9: Gateway and Virtual Service Yaml files 23

Figure 10: Gantt Chart 27

Figure 11 GKE Cluster 29

Figure 12: GKE Application Deployments 29

Figure 13: LSTM Model Accuracy 31

Figure 14: ARIMA Model Accuracy 32

Figure 15: HoltWinters Model Accuracy 33

Figure 16: GRU Model Accuracy 34

Figure 17: Bi-Lstm Model Accuracy 35

Figure 18: Bi-Lstm-CNN Model Accuracy 36

Figure 19: Bi-Lstm-CNN Model Loss 37

## List of Tables

Table 1: existing research on scalability, metrics and technology used 18

Table 2: tools and technologies 12

Table 3: LSTM Model Accuracy 31

Table 4: ARIMA Model Accuracy 32

Table 5: HoltWinters Model Accuracy 33

Table 6: GRU Model Accuracy 34

Table 7: Bi-Lstm Model Accuracy 35

Table 8: Bi-Lstm-CNN Model Accuracy 36

## List of Abbreviations

VM Virtual Machine

HPA Horizontal Pod Auto Scaling

VPA Vertical Pod Auto Scaling

MAE Mean Absolute Error

MSE Mean Squared Error

RMSE Root Mean Squared Error

QOS Quality of Service

GKS Google Kubernetes Service

# INTRODUCTION

Containerization is a technology that allows applications to be packaged and run in lightweight, isolated environments. instead of running a complete virtual machine (VM) with its own operating system (OS) on the top of virtualized hardware, containers provide an isolated environment for system resources (e.g., processes, file systems and networks) to run at the host OS level, this means that multiple

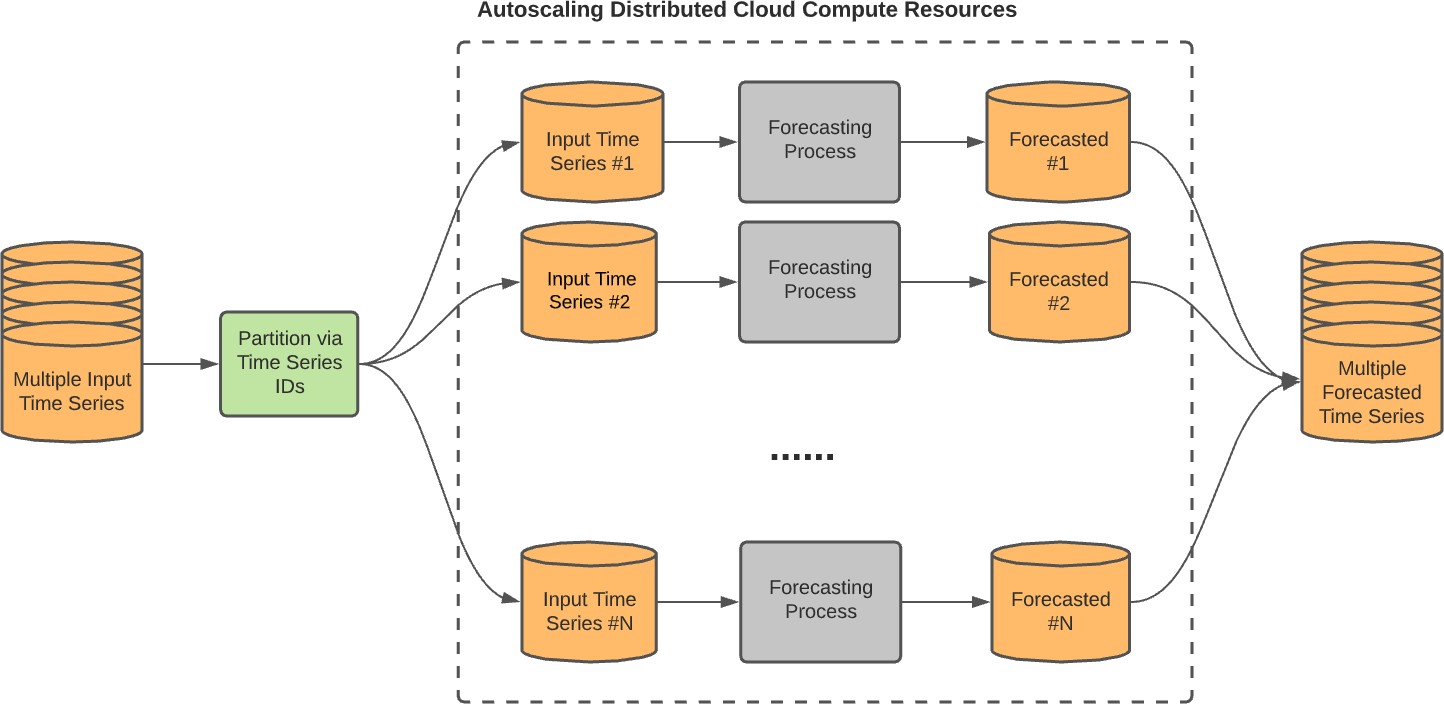
containers can be shared with the same OS kernel, hence start much faster, and use just a fraction of the system memory compared to booting an entire virtualized OS [14].

Due to the popularity of microservices the need to scale and automate the deployment was increased. so, google developed an open-sourced container orchestration platform called Kubernetes, also known as k8s. Kubernetes allows users to deploy and manage microservices in a distributed environment , make it a popular choice for building and deploying microservices based application.one of the important feature provided by Kubernetes is autoscaling in different ways such as automatically scale the number of pods running a microservice based on the matrices ( Horizontal Pod scaling ) , automatically adjust the resource limit of individual pods according to the usage (Vertical pod scaling). Kubernetes scaling was made according to the workload, so auto scaling microservices applications can be not accurate since services are distributed.

In [13] proposes an intelligent workload factoring approach to optimize resource

allocation in a hybrid cloud computing model. So, by identifying the challenges of workload factoring in hybrid cloud environments, where workloads can be distributed across multiple public and private clouds. The authors propose a solution that uses a workload characterization module to classify workloads into categories based on their resource requirements and performance characteristics. Then, a

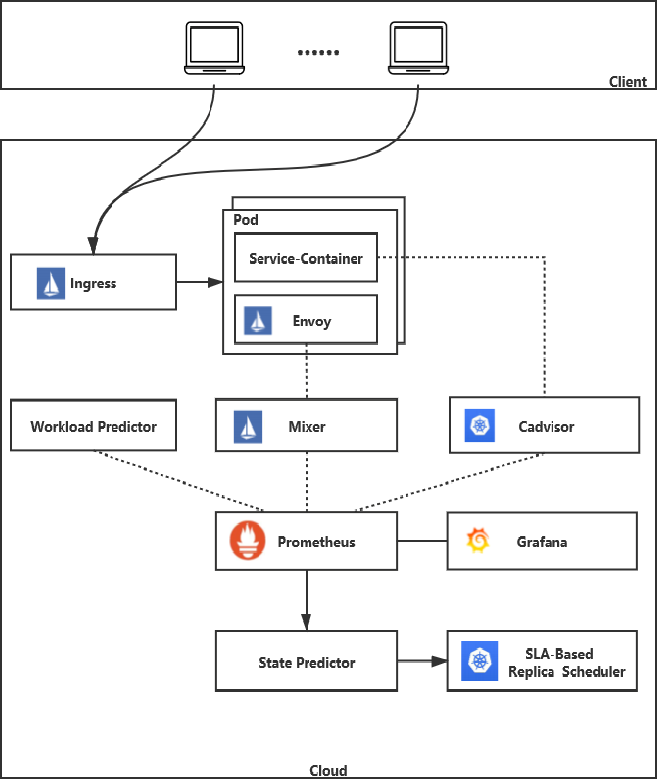
decision module uses this information to allocate resources optimally across different clouds, considering factors such as cost and network latency. The authors also propose a feedback mechanism to adjust the workload factoring strategy dynamically based on changes in workload and resource availability.



*Figure 1: Architecture of ARIMA Model*

In [15] proposes a deep learning-based autoscaling algorithm for Kubernetes, which is a popular open-source container orchestration platform. The proposed algorithm utilizes bidirectional long short-term memory (BiLSTM) neural networks to predict the future resource utilization of a Kubernetes cluster and dynamically adjust the number of replicas of containerized applications to optimize resource usage. The authors first trained the BiLSTM model on a dataset of historical resource utilization data collected from a Kubernetes cluster. The model was trained to predict future resource utilization based on past utilization patterns. Once the model was trained, it was deployed as a microservice within the Kubernetes cluster itself. The model continuously receives real-time resource utilization data from the Kubernetes cluster and predicts future utilization. The predicted future utilization is then used to

dynamically adjust the number of replicas of containerized applications to optimize resource usage.so by, evaluating the proposed algorithm using a real-world dataset collected from a Kubernetes cluster running various microservices. The evaluation results showed that the proposed algorithm achieved better performance in terms of resource utilization and response time compared to existing Kubernetes autoscaling algorithms such as the default Horizontal Pod Autoscaler (HPA) and the Predictive Horizontal Pod Autoscaler (PHPA).



*Figure 2: Architecture Prometheus matrices gathering for prediction.*

In publication [2], proposes a new auto-scaling algorithm based on reinforcement learning (RL) for containerized cloud applications. The authors argue that existing.

auto-scaling approaches are often reactive and do not consider future resource usage, which leads to performance degradation and unnecessary resource consumption. To address these issues, the authors propose A-SARSA, an RL-based auto-scaling.

algorithm that takes both current and predicted future resource usage into account. A- SARSA uses the state-action-reward-state-action (SARSA) algorithm to learn an optimal policy that determines when to add or remove containers based on predicted future resource usage. The authors evaluate A-SARSA using several real-world benchmarks and show that it outperforms existing auto-scaling algorithms in terms of both resource utilization and application performance. A-SARSA achieves higher resource utilization by scaling proactively based on predicted future usage, and

improves application performance by avoiding under-provisioning and over- provisioning of resources. Overall, the paper proposes an innovative approach to auto-scaling containerized cloud applications that leverages RL to learn and optimize resource allocation policies. The results suggest that A-SARSA can improve both resource utilization and application performance, making it a promising direction for future research in this area.

Diagram

Description automatically generated

*Figure 3: Architecture of DScaler*

DScaler [1] was a horizontal autoscaler for microservice based application developed based on deep reinforcement learning. DScaler was developed for the usage of large scale and more complex microservice architecture applications to fix the scaling issues. DScaler uses a combination of deep neural networks and one of reinforcement learning algorithm called Q-learning algorithm to learn a policy for scaling microservices based system metrices. The proposed DScaler algorithm was tested with large e-commerce applications and with some rule based approaches.as a response DScaler has a good performance in resource utilization and response time.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Paper | Virtualization Technology | Monitored Metrics | Method | Technique |
| Zhang et al. [3] | VM | Request rate | Reactive | ARIMA |
| Li and Xia [4] | Container | CPU | Proactive | ARIMA |
| Rodrigo et al. [5] | VM | Request rate | Proactive | ARIMA |
| Xuehai Tang et al.[6] | Container | CPU | Proactive | Bi-LSTM |
| Ming Yan et al. [7] | Container | CPU, Memory | Hybrid | Bi-LSTM |
| Imdoukh et al. [8] | Container | Request rate | Proactive | LSTM |
| Laszlo Toka et al.[9] | Container | Request rate | Proactive | LSTM |
| Arabnejad et al. [11] | VM | CPU | Reactive | RL |
| Fabiana et al. [12] | Container | CPU | Reactive | RL |
| Prachimutita et al.[10] | VM | Request rate | Proactive | ANN, RNN |

*Table 1: existing research on scalability, metrics and technology used.*

Most of the research is based on the VM and container-based prediction for scale the resource load .so there are certain limitations on selecting the matrices for start the predictions. and all research is scaled based on the microservice architecture and deployed in and cloud-based cluster so there will be a cost that will be associated according to the resource load. So, through this component, first a prediction was made and according to the prediction by using an reinforcement algorithm, the predicted result will be optimized.

# RESEARCH PROBLEM

Resource management and scaling are critical issues for microservices deployed on Kubernetes clusters. As the number of microservices and the volume of traffic

increases, it becomes challenging to manage the resources effectively and ensure optimal performance. Traditional approaches to resource management, such as manual scaling, are not efficient and can lead to over-provisioning or under-

provisioning of resources, which can impact the performance and cost of the system.

To address these challenges, there is a need for a resource management framework that can dynamically manage the resources of microservices based on the workload and resource demands. This framework should utilize modern technologies, such as machine learning algorithms, to predict the resource load of microservices and automate the deployment strategy based on the prediction.

Therefore, the research problem to be addressed in this research paper is: How to develop a machine learning-based resource management framework for

microservices deployed on Kubernetes clusters? How to utilize machine learning algorithms to predict the resource load of microservices through time series data queried and propose a reinforcement learning algorithm to automate the optimized deployment strategy based on the prediction? How to evaluate the performance of

the framework under different workloads and resource demands to ensure scalability, efficiency, and cost-effectiveness?

# OBJECTIVES

## 3.1 Main Objective

this component aims to develop an optimal solution to Kubernetes cluster to allocate the resource requirements according to the demand that changes with time and complexity of the microservices.so as a first step set up the Istio service mesh in the Kubernetes cluster and get the time series matrices and logs from monitoring tools

like Prometheus, Grafana and Kiali. By using prediction models and reinforcement learning, an autoscaling approach can be developed.

## 3.2 Specific Objectives

1. Configure the Kubernetes cluster.
2. Configure the Istio service mesh.
3. Get the time series matrices of microservices using an API of Prometheus, Grafana and Kiali
4. Store those gathered time series data to a persistent database, so that can be use in the machine learning model to make the predictions.
5. Identify and apply a machine learning algorithm that can be used to predict the resource load of microservices through the time series data queried from the cluster.
6. Propose a reinforcement learning algorithm to automate the optimized deployment strategy based on the prediction.
7. Configure Kubernetes cluster health check.
8. Perform concurrent performance test with large loads (Gatling, JMeter, Locust).
9. Automating performance test and make as a stage in CI pipeline execution.
10. Propose a way to perform throttling, error injection and chaos testing.

# METHODOLOGY

## 4.1 Requirement Gathering

Requirements are gathered for this component by referring several research papers , articles , publishes done in recent years and also by having a conversation with

industry expertise. the knowledge to start the development and the configurations are gathered by following some Udemy courses alongside by referring to the documentations and developing some practical’s for test purpose to get the clear understanding on the tools and configurations.

## Past Research analysis

Past research analysis are done by analyzing past research publishes on the area of Kubernetes, setting up Istio service mesh , Prometheus, Grafana and Kiali , existing prediction models that are used on resource management of Kubernetes , problems on storage allocation on Kubernetes , performance test with large loads by using frame works , tools that can be used on chaos testing.

## Software solution and System analysis

**4.2.1 Solution System analysis introduction**

A picture containing text, white, bunch, screenshot

Description automatically generated

*Figure 4: System Overview Diagram*

in this component a helm chart was created with a set of Kubernetes yaml files, which define the resource requirement to run the application, such as deployment, services, and configmaps. So, the created helm chart was stored in a helm hub. So, from the repository the chart was retrieved and deployed to the Kubernetes cluster. By enabling the relevant configurations for Prometheus, Grafana and Kiali it can be used to query the data and stored in a persistence database for make the predictions.

When it comes to testing the performance of a Kubernetes cluster, tools like Gatling, JMeter, and Locust can be used to simulate various scenarios and test the clusters’ ability to handle different loads. Automating performance tests and making them part

of CI pipelines is essential to guarantee performance and quality expectations and reduce downtimes. Datadog can be integrated with Gatling, JMeter and Locust to provide real-time visibility into the performance of your applications and

infrastructure during the load testing.

Chaos monkey and Kube-monkey are open-source tools that simulate random failures in Kubernetes clusters.

Chaos testing is a technique that is used to test the resiliency of a Kubernetes cluster by deliberately injecting failure scenarios**.**

### The steps that are followed in this component as follows,

1. create a Kubernetes cluster using a cloud provider such as Azure AKS using Free tier.
2. Configure Istio service mesh, Prometheus, Grafana and Kiali in cloud environment and deployed to AKS cluster.
3. Gather the data and logs from the metrics through a metrics server and store them in a persistent database that can used to make the prediction using a machine learning model.
4. Develop the prediction server by using some popular and accurate time series prediction models like ARIMA and LSTM the prediction can be done, and the limit can be updated in the Kubernetes Vertical Pod Scaling (VPA).
5. Cluster Health Check (CHC), would, based on a criterion, automatically detect all deployed services and pull their Health Check Status over /healthz API endpoint. Using a rolling time window, the CHC service would calculate the overall availability of the cluster and provide the status of the cluster health on demand – either success (HTTP 200) of a failure (HTTP 4xx-5xx).
6. Generate a status report for CHC.
7. Configure Gatling and JMeter to run the performance tests on the Kubernetes cluster.
8. Create an automation script to automate the execution of the performance tests, including commands to start the Gatling or JMeter test and to generate test reports.
9. Modify the existing pipeline to include the automation scripts for the performance tests.
10. Generate a test reports using Gatling and JMeter after each test run, including metrices such as response time, through put, error rate and with graphs and charts.
11. Configure and setup Datadog agent on the where Gatling or JMeter is running. The agent collects and sends metrics and logs to Datadog.
12. Analyses the resource performance through the Datadog dashboard.
13. Create alerts based on the metrics and logs in the Datadog dashboard and analyze them to identify performance issues.

**4.2.2 Matric Server**

![A diagram of a diagram of a server

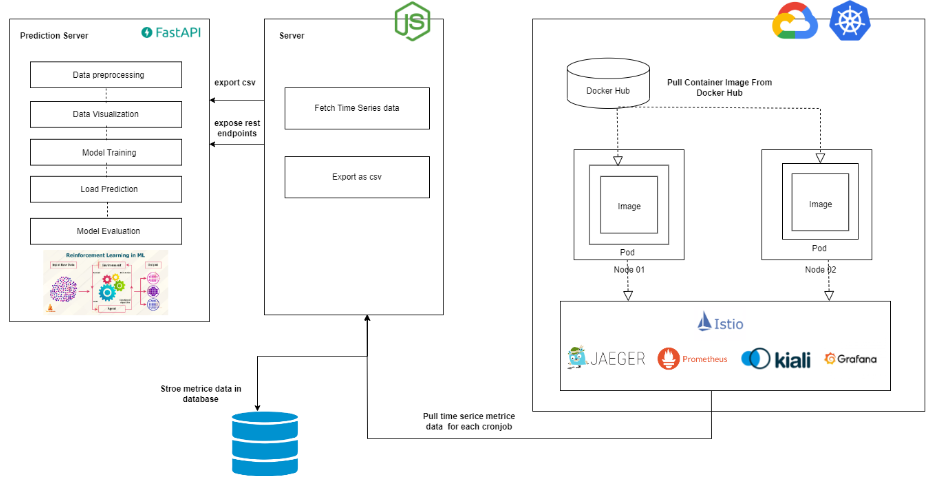
Description automatically generated](data:image/jpeg;base64,/9j/4AAQSkZJRgABAQEAYABgAAD/4TN6RXhpZgAATU0AKgAAAAgABgALAAIAAAAmAAAIYgESAAMAAAABAAEAAAExAAIAAAAmAAAIiAEyAAIAAAAUAAAIrodpAAQAAAABAAAIwuocAAcAAAgMAAAAVgAAEUYc6gAAAAgAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAFdpbmRvd3MgUGhvdG8gRWRpdG9yIDEwLjAuMTAwMTEuMTYzODQAV2luZG93cyBQaG90byBFZGl0b3IgMTAuMC4xMDAxMS4xNjM4NAAyMDIzOjA2OjI2IDIzOjQ2OjM1AAAGkAMAAgAAABQAABEckAQAAgAAABQAABEwkpEAAgAAAAM4MAAAkpIAAgAAAAM4MAAAoAEAAwAAAAEAAQAA6hwABwAACAwAAAkQAAAAABzqAAAACAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAMjAyMzowNjoyNiAyMzo0NTozOAAyMDIzOjA2OjI2IDIzOjQ1OjM4AAAAAAYBAwADAAAAAQAGAAABGgAFAAAAAQAAEZQBGwAFAAAAAQAAEZwBKAADAAAAAQACAAACAQAEAAAAAQAAEaQCAgAEAAAAAQAAIc4AAAAAAAAAYAAAAAEAAABgAAAAAf/Y/9sAQwAIBgYHBgUIBwcHCQkICgwUDQwLCwwZEhMPFB0aHx4dGhwcICQuJyAiLCMcHCg3KSwwMTQ0NB8nOT04MjwuMzQy/9sAQwEJCQkMCwwYDQ0YMiEcITIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIy/8AAEQgAxQEAAwEhAAIRAQMRAf/EAB8AAAEFAQEBAQEBAAAAAAAAAAABAgMEBQYHCAkKC//EALUQAAIBAwMCBAMFBQQEAAABfQECAwAEEQUSITFBBhNRYQcicRQygZGhCCNCscEVUtHwJDNicoIJChYXGBkaJSYnKCkqNDU2Nzg5OkNERUZHSElKU1RVVldYWVpjZGVmZ2hpanN0dXZ3eHl6g4SFhoeIiYqSk5SVlpeYmZqio6Slpqeoqaqys7S1tre4ubrCw8TFxsfIycrS09TV1tfY2drh4uPk5ebn6Onq8fLz9PX29/j5+v/EAB8BAAMBAQEBAQEBAQEAAAAAAAABAgMEBQYHCAkKC//EALURAAIBAgQEAwQHBQQEAAECdwABAgMRBAUhMQYSQVEHYXETIjKBCBRCkaGxwQkjM1LwFWJy0QoWJDThJfEXGBkaJicoKSo1Njc4OTpDREVGR0hJSlNUVVZXWFlaY2RlZmdoaWpzdHV2d3h5eoKDhIWGh4iJipKTlJWWl5iZmqKjpKWmp6ipqrKztLW2t7i5usLDxMXGx8jJytLT1NXW19jZ2uLj5OXm5+jp6vLz9PX29/j5+v/aAAwDAQACEQMRAD8A9/ooAKpPq+nRSvFJfW6yIdrKZACp9DQAn9taX/0ELb/v6KP7a0v/AKCFt/39FAB/bWl/9BC2/wC/oo/trS/+ghbf9/RQAf21pf8A0ELb/v6KP7a0v/oIW3/f0UAH9taX/wBBC2/7+ij+2tL/AOghbf8Af0UAH9taX/0ELb/v6KP7a0v/AKCFt/39FAB/bWl/9BC2/wC/oo/trS/+ghbf9/RQAf21pf8A0ELb/v6KP7a0v/oIW3/f0UAH9taX/wBBC2/7+ij+2tL/AOghbf8Af0UAH9taX/0ELb/v6KP7a0v/AKCFt/39FAB/bWl/9BC2/wC/oo/trS/+ghbf9/RQAf21pf8A0ELb/v6KP7a0v/oIW3/f0UAIdb0sD/j/ALc+wkBJrlpvEGvDzr4SWMMEbfJZyZ3yL/vf3sdga5amIisRTw6dnI66EKSpzq1U2ltbudbpuoQ6pp8N5bnMci5weqnuD7g1brsnBwk4vdHFGSkk0FFSUFFABVHTfvXv/X038hQBeooAKKACigAooAKKACigAooAKKACigAooAKKAIbu3W7tZIGJAcYyO1YQ0/xMi7E1WzCgYH+j84/Os6dKCxKrT1VrNff/AJk1XVcFGDsWfC9kLDRvJ373E8okfGNzBypOO33a2q6K1T2lRz7ipx5IqPYKKzLCigAqjpv3r3/r6b+QoAvUUAFFABRQAVwV3rWsapql/FYXYtre0laNNkeS5XqSTwRnPFduDpwk3KorpGFeUkko7s0NK8WmPRr+41cqJLC48mVo0K5U4w2D75/KtBvFdkkKMY5TIxXESDJwzRgH0/5bIfxrHE01TquMdi6UnKCbGr4x0eQEwzPLhsfIueisx+mAjdfSmTeM9LiDKvmvMmA0IUBwSeARn8fSsDQuR6/a3Nok9orTGSbyETIB3Yzz6cevI6YqAeK7BDdLcJLBJbMiSIy5ILKGHTjvj8KAI4/GekvM8bOyENheh3DYj7hg9P3iity2nW6tYrhFZVkUOoYYOD0oAlooAKKACigAooAoaR/x5Sf9fVx/6Oer9ABRQAUUAFUdN+9e/wDX038hQBeooAKKACigArk9S8HPJqU15pd3HZ/aCGmjaEOrMO4B6Zzz/wDXrpwuI9jJ3V09zKtT9otN0aej+G7LStM+ylFnd3Ek0rDmR85BP07Cnnw1oxYt9gjBIA4JGMFW4545ROn90egrKtVdWo5vqVCChFRRKuhaWgIWyjCnPy84GVKnA6DIZhx60PoWmSO7taJmT7+CRu9zg9fQ9RWZY8aRYLbtAtuAjOH4Y7gwGAQ2cjA4GDwOKgHhvRwwZbCJCMY2ZXoAB09ABj0oAcvh/SkbctminvgnnhQc885CL1/uj0rRjRYo1jRQqIAqgdgKAHUUAFFABRQAUUAUNI/48pP+vq4/9HPV+gAooAKKACqOm/evf+vpv5CgB+qSyQ6TeSxOUkSB2RgAdpCkg88fnUK6fdFQf7Zvun9yD/43QAv9n3X/AEGb7/viD/43R/Z91/0Gb7/viD/43QAf2fdf9Bm+/wC+IP8A43R/Z91/0Gb7/viD/wCN0AH9n3X/AEGb7/viD/43Uc1tLAEMmt3q73CL8kPJPT/lnQlcCT+z7r/oM33/AHxB/wDG6P7Puv8AoM33/fEH/wAboAP7Puv+gzff98Qf/G6P7Puv+gzff98Qf/G6AD+z7r/oM33/AHxB/wDG6P7Puv8AoM33/fEH/wAboAP7Puv+gzff98Qf/G6P7Puv+gzff98Qf/G6AD+z7r/oM33/AHxB/wDG6P7Puv8AoM33/fEH/wAboAP7Puv+gzff98Qf/G6P7Puv+gzff98Qf/G6AD+z7r/oM33/AHxB/wDG6P7Puv8AoM33/fEH/wAboAS1FxDqktvLeTXEfkq481UBBLEfwqPStGgChpH/AB5Sf9fVx/6Oer9ABRQAUUAFUdN+9e/9fTfyFAC6x/yBL/8A69pP/QTVxfuL9KAFooAKKACuW8YWmsX4toNNACKwk3ZP3wQRn0HWtaLippy2IqJuNka2p299daSkduwjutyMxV+mCCcGuai0nxSl357zkgoA4E2CwBXgZJ2tgNnnv17jIsv6lpuuzNay2kyLepbhfNkfMSvg5JUYOeR0yD0IAqNNO8TGEML0q5ZU2u/3UOQ56nLDO4ZJ6YoAii07xVHbQRm5xIpAkYOCpG0YwMjGGyT656HoH6vpHiOfVZZtPvFjgeRJAHkOBtQDGPQkkn6CgBh0vxNM0CSXb+WQPNPmbSuHJ4IY84wOc8fxDoeqsVuEsoUum3ThcOc9TQBYooAKKACigCiv/Iek/wCvVP8A0JqvUAUNI/48pP8Ar6uP/Rz1foAKKACigAqjpv3r3/r6b+QoAXWP+QJf/wDXtJ/6Caluby2sbT7RdTJDEoGXc4FOMXJ2W5MpKMXKWyObm8e6fNqcOm6Oo1C7k6nzPLjT0BYg8n0AP8s7Om6x9suZbO6tZLK+iXe0EjBtyf30YcMuePUHqBWlSjOn8asZUsRTq6wd0WRqdg1lJei9tzaxbvMmEg2Lt+9k9BjvVhHWRFdGDKwyrA5BFZWNroq2mradqDbbO+trhtm/EUoY7clc8dsgj6iqur+ILTSZYbdlea6nIEcEWNx7ZPoKipNU43Zth6TrzUIjNI8S2mrXElqI5be7jGWhlGDj1BHXrWzSp1FUjzIeIoSoT5Jf0gorQxGySJFG0kjBUUZYnoBWVYeJNN1OVY7WV2ZiQN0bKOPqOahzSko9WJtLc16KsYVXur61sl3XM6RA/wB4/hSlJRV2VCEpvlirskE8LRCUSoY26NuGD+NSU07iaadmICD0INLQIor/AMh6T/r1T/0JqvUAUNI/48pP+vq4/wDRz1foAKKACigAqjpv3r3/AK+m/kKAF1j/AJAl/wD9e0n/AKCawPHej3usaFClipkkikDmMHlhjFdGEqRp1oylscuOpSq4ecIbtHmGhaNf6f4ns7K/heyleczI83yhk3ZyCeDjpj6eor1kXUOseKbaWwdZoNPimWe4jOU3vtxGCOp43HHTC55NdeYVYVHHld7I4MqozpRmpq12jyGDVdWsNI1W1naWGxnS/wDsIRNyzSByZN+R1Cgkdq1pPFWoHxDp5t9QuY/L1C1t54JJcYibaG/dhcBDn7xPfiuRxTO5SaL3wimnF5FbPkRJpIYKVAw32iUHnrXT6nGbDx9DqFypaGaIJCxztDdNvAPP+NceMWz7NHqZW/ij1cWiOOBtU8fC7s1KCCLFxIM4U4IC8gc/57V1a2dzGu1Ls/8AAhn0/wDr/p6VjQi3zSXVs6cXUjHkhJXaiv6+4Y1leMT/AKewXBGNvsR1+pz+FOFrf7cG9BOQfuenb8a15J9zl9pS/lJdQTzLCVNpYkcADOTXJWUFxa6qoW1dWLiYjBL7QAD1PIz/ADFdVNQer6HlYrnUlypu9vzNLW7PX727WXTZY7eNYyinzirNu2nJG0gEFSPoe1Zc1n4ttokV72WeWQ4HlN8qvkYP3OFxu4PHuOhyO02dE0/W7e6Emp37TRhCNu8EFiev3Rx6DtU+pWEsupRXaxecFj2BVwGXkHIJ9a58Spun7iu+xtQmoTuzOv8AQry4s7YJbWtztD5huH+VdxyG+6QSOnT3qvPovinyy0eqbpW2AKkvlomI0GQNp43iQ49xzWlJNQXNvYio05torv4d8UWyKljqO3cqszG4wFc58zjYd2TjByMVONH8VxXNoBqZlijniLM1xg+WJcyBhs+clOByMVoQdKv/ACHpP+vVP/Qmq9QBQ0j/AI8pP+vq4/8ARz1foAKKACigAqjpv3r3/r6b+QoAdqys+jXyIpZmt5AFAySdppi6vabQCLnp/wA+sv8A8TQBDc3elXsXlXds88ec7JbKRh+RWpItRsII1jhimjjUYCpaSAD8NtAWHf2vaelz/wCAsv8A8TR/a9p6XH/gLL/8TQAv9r2npc/+Asv/AMTTH1OxlQpIk7qeoa0kIP8A47QNNrVBHqVhCmyKOZFHZbSQD/0Gn/2vaelz/wCAsv8A8TQlYG23dh/a9p6XP/gLL/8AE0w63YrIsZNwHYEqPssvOOv8NAhJ9Q065haGeGWWJxhkezkZSPcFaSO/02Jy8cEqMQAWWzkBIHQfdoAl/te09Ln/AMBZf/iaP7XtPS5/8BZf/iaAD+17T0uf/AWX/wCJo/te09Ln/wABZf8A4mgA/te09Ln/AMBZf/iaP7XtPS5/8BZf/iaAD+17T0uf/AWX/wCJo/te09Ln/wABZf8A4mgCO1mW51eWeNZPLECpueNk53McfMB61pUAUNI/48pP+vq4/wDRz1foAKKACigAqjpv3r3/AK+m/kKAL1FABRQAUUAFFABRQAVRuP8AkM2P/XOX/wBloAvUUAFFABRQAUUAFFABRQBQ0j/jyk/6+rj/ANHPV+gAooAKKACsyOLUbaa48mG1kjllMgLzMp5A4wEPp60AS+Zqv/PrZf8AgS3/AMRR5mq/8+tl/wCBLf8AxFAB5mq/8+tl/wCBLf8AxFZniDX73w9od1ql1Z2rxwLnZHcOWY9AAPL6k0AXLK/1G/sobuG2szHMgdc3Dg4Pr+7qfzNV/wCfWy/8CW/+IoAPM1X/AJ9bL/wJb/4ijzNV/wCfWy/8CW/+IoAPM1X/AJ9bL/wJb/4ijzNV/wCfWy/8CW/+IoAPM1X/AJ9bL/wJb/4iqNxJqn9r2WbWzzslx/pDf7P+xQBe8zVf+fWy/wDAlv8A4ijzNV/59bL/AMCW/wDiKADzNV/59bL/AMCW/wDiKPM1X/n1sv8AwJb/AOIoAPM1X/n1sv8AwJb/AOIo8zVf+fWy/wDAlv8A4igA8zVf+fWy/wDAlv8A4ijzNV/59bL/AMCW/wDiKADzNV/59bL/AMCW/wDiKPM1X/n1sv8AwJb/AOIoAPM1X/n1sv8AwJb/AOIo8zVf+fWy/wDAlv8A4igCTT7eS2tNkpXzGkkkbacgF3ZsA98ZxVqgAooAKKACigAooAKoa2obRbzIBxExGfpQBeACjCgAegpaACigAooAK5/VNcsbDxTpNhcSOtxcrIsKCNjvJ29wMcYJPtQB0FFABRQAUUAFFABRQAUUAFFABRQAUUAFFABRQAVzWqOzz3qyGeSPzREI0kkChfLU9ERu5PWgB2nXUxvbZt84WWUwukryMCNjMCN6Lg5Xt610dABRQAVT1a7ksNIu7uKPzJIYmdU/vEDpQBzWoeNbjTR9nm04i+UOzx7xgAIWB4J68Dqe9Fz4lQ628cmkrNcWc5ihkD427ii5yR33DPtnrQBYh8bWtxp1zerayokGzIlO04YE8jBPbtnII98QyePIRcTRR2Fw3ku4c45wrbenXJPP0I9wADR0XxRBrV35EcLRkxGUZbJIBA9OBzxkg+w4zvUAFFABRQAUUAFFAB0rOutf0mzcpPfwK46qGyR+AqowlJ2iiZTjBXkysvizQmOBqCfirD+laVrfWl6m61uIph/sMDVSpTiryRMK0Ju0WWKKzNAooAKKACvPL7XNViuLiU+HV1C3mk8xWSB5duCIyMAHnCFvy9aAK+n+I9Umm3y+GTpS2kokBe1dFlJGwjOAOkhI+lel0AFFABXLa1rHiK01KWPT9L+0QIjbP3bEM2EKkkf8DGB6c8c0ARQav4mltrhX00RyRtDscxN8wMoEnHfCZPFLa6z4o2Itxo6FgY0ZgrDcSBuPXgcnnoMUAVj4m8QzXMVpHpaQXEiowV0YkDC7yRngAlh79q1/7W1n7XDGNKPlvZiVzg/LLgnZnp1AHrQBSk1vxIqAx6OrlmIX5GGRhOTzx95+v9zHWui0+W5nsIpLuIRXBHzoOgNAFqigAooAKKACigDkviFqz6ToCMsrRLNKI2dewrzJJEkG9XDA9wc16eES9nc8rGt+0JB1q3YXc1rexS20hWVWHKn+ftXTJJqzOaMuV3Pa6K8M94KKACq98Lk2UotCouCvyFumaAORSDxtD5MRlidCVBbepI+TnPH97PTrx0rL1Oz1TRNLm1DVI9N8mJd007Qo2AAxJ5GcnCL3yWoAs2+kazPaW93p8WlSxyL5sUphQEAgFTwv1/HFbCweKII5AJllleb5WYjaEAY8jHGcKPxoArvZ+M5LNh9rhSbYyhQwx1bBzjrgr+VdbAHW3jWQ5kCgMfU45oAkrI16HVLhLRNLlMLiYvK+RjYEYY5zySVxx2zQBlwaX4nj0+9jbVf9IkdGic4bb82WxkcZXimDSfFu+Ldri7Q48zbGoyu4E444OMigC5b6frD/ANlzXF5I0sLMZ/n2hlJOMqOCcY+hzVRNK8UQRRRx6tuCEEl8EkbUyCSMnkPjv83egCzYWPiOO5t5LzURIgkJljG3bt2px0z1D/nXSUAFFABRQAUUAFVbPUbO/wDNFpcxzGFzHKqtkow6gjqDQBw3xhUt4Ugx/wA/K/yNeO2jMvQkH2NephP4Z5uL+M2bdmYfMxP1NbNnIqyoo5bIworpexwM9zorwj6EKKACigArO12CG40S7SeJJUEZba6hhkcg4NAFy2gitraOGCJIo0GFRFAA+gFS0AFFABRQAVh32sWEHinTtOluVW7kjk2QnOWzjp+RoA3KKACqt7dm1WJUiMksz+XGucZO0tyewwppN2Qm7IyTqU0T+YdTt3kzhrfYQi84xvxkH3PHtWxZ3Qu4PM2MjBirKexHWs4Su7XuRCV3a9yxRWpoZ2tQxT2cUU0aSRtcwhkdQQf3i9jVSS20VLiSFdDilaMgMY7NSASM+noaAE+zaT/0L3/kklGg6PpGlzXL6bpBs5Z3Mk0rRbWkJPr6e3QdqAOZ+MIz4Ug/6+V/ka8ctTkLnn6816mE/hnm4v4zYtlUjkZ+tbdjsjdMADLDAA610PY4Ge50V4Z9CFFAFTU5ZINJvJYnKSJA7IwAO0hSQeeKiWwuioP9sX3T+5D/APG6AKt8RpqxNd69eRLK+xCUh5bBOP8AV+xqrdT2L2ExuPElyYNuJBiHoQfSPPY/kaAJmubdHCHxJcgkkDiDqMZH+r96WWeOC8e1k1+8WVF3tlIcAe58ugBRPAxIHiWfIAJ/1HAP/bOnwbblwkHiG6kcjcFVYc4/790AWv7Puf8AoMX3/fEP/wAbo/s+5/6DF9/3xD/8boAP7Puf+gxff98Q/wDxuqkvhxJtSg1GTULpruBGSKUxw7lVsZA/d+1AFv8As+5/6DF9/wB8Q/8Axuj+z7n/AKDF9/3xD/8AG6AD+z7n/oMX3/fEP/xuqOo6deq1pOmoX04hm3sFSHcoKMuVGzk/N054J74qZbEy2OXutNmhgFy2rTQySXCp5Sqol+aQDkBctwc4NdxpcciWztLGY2kkaQI3UAnjNY0lZmVJWZdoroNyjqv/AB7wf9fUP/oxaSy/5COpf9dU/wDRa0AX6KAPPvi/G8nhONkUsEuFLY7CvGrY5AzXp4T+Gebi/jNi2UHHX/vo1u6bFmeMIuXLADuTXS9jherse2yzxwgGRsZ6cVF9vth/y0/8dNeJY9+5YR1kQOhBVhkEUtIZR1gj+xb8ZGfs0hx/wE1dX7i/SgCjquj2ms26wXis0akkAHHVSv8AI1lt4K0xoTHvuANpQHfyFO7I6d97e/5UAQv4F09p1YSyLEFcbABnLEZO76DHTPPWtDUvDNjqs3mXLTHgYAboR0Prxjp09qAIX8H6VLAsMqSOgcuw3Y3ErtOcdj/Orml6Fa6S7vAXZ3UKzNjkDp0FAGnRQAUUAFFABRQBl6xDAlss/loshubfc+BnHnJ3q/8AaYP+e0f/AH2KLWC1g+0wf894/wDvsUxr+0X71zEP+BigChf39pcLbxQ3EckhuYiFVsnhxmrFl/yEdS/66p/6LWgC/RQBS1DT0v4WjfBVhgqwyDXCXnwxs5ZS1tiIk9I24/KtKdWVN+6Z1KUai94hh+G/lOQ12cDkgkcCuo0fwnbacwkXaXH8ROTVzxE5qzM4YanB3Ro6tPHt8tZU8xBkruGV9zWGt2eQ0sLg8cyY/wDZawvYKk7S3Og0ceXZFdxKqeMnoK88uPjr4asvEerWV1K32OyVVhkhjZ2uJckOF7ADjkkZps3Wxb8L6nqfivTte8V3ttJZ2dxavb6fbOefKUElz7k/yr0hfuL9KQxaKACigDO86/mvbiKCS2SOIqB5kbMTkZ7MKrzX13BdrayXdr5zLuwtpIwA9SQ2APrQAz+1m8x4/wC19K3Iiuw2H7rFgD9/uVb8qJNUnjsPt32+ykts8yRW0kgH12scUALFqcszKq6npodukbxMr/8AfJfP6VMLq7aOOQajppSU7Y2CHDn0Hz80ARnUZgUH9qaYS5IXCE5x1/j7VJFc3c7qkOoabIzLvASMklemfv8ASgBbmbU7SISvLaOodQVELA4Jx13VqUAZfiAkaSSITMftEH7oY+f98ny88c9OeKx9Cv59Ss5Zb7wfLZzLMyeSUhyoHTksM59QMUAamIv+hbl/75t//i6esu37vh+4H08j/wCOUASi9nX7ujXg+jQ//HKdp6zGe8nlgeDzpFKo5UtgKBztJHUetAF6igArlLvwvcpdXV7FrJt45PMkcMDtXc6tnO4Ywq4/DPrkArTWFpf6qJ7bxBYvJMq7IywfzSuzqA3I/dngetbnh/SrvSLZ4Lm7S5DMXBCFSCe3JPFAHCa/LPb+PdRNrK8TyLHu8sAs+I1wBnvURvdT8qM/b7lGeUxjegxnnA6ewrm5pXdmfOSq1vaTUZNJN/ma/gjV767165tJ7tpoFhZgGA6hlGf1NXrb4ZeHIPFl9rrafbSm6VT5EkQZI5ATlwD65/Stabbjdnq5dUnUo803d3Z0urgLoV+AAALaQADt8pq6v3F+laHcLRQAUUAUbP8A5CF//vp/6DUd7olnqE801wm55IPIzgZUc8g9jzQBmp4OtI5PMW4kDmMRsSAcjMh79/3rVcfSbCx0e8glnENrKMySOwUKNoHXoOlAHOT6Z4RuYWSTV4iLt8fLMAx3OjfUA+UOa2YNC06/022W3vfOijl8zz4ZAdxyMjcD7CgCAeBNN+xC1aWcoI/L3budoAA5+gFX9O8N2+nXsd1HNIXVWUjsxJJJPr1NAFzV/wDjwP8Avp/6EKvUAZ+tMFsEZiAourcknoP3yVcW4gf7s0bfRgaAJOvSigAooAKKACormBbm1mt2JCyoyEj0IxQBhweE7eHUYr43MrTKUL9g5ThePoAK6GgDJvfDOj6jdvdXdkskz43PvYZwMDofSq//AAhnh7/oHL/38f8AxqHTi3exxywGHnJylHV+pb07w/pelTtPY2iwyMuwsGY8dccn2FadUopKyOilShSjywVkUtY/5Al//wBe0n/oJq4v3F+lM0FooAKKAMwSy2l/dsbS4lWQqVaNQR0x61L/AGi//QPvf++F/wDiqAD+0X/6B97/AN8L/wDFVWvpRf2pglsb9QWVwyKoKspDAjnsQKAML/hGbHzFk8rW/MUEK+5Ny59D1H9M1saWsekWhtrex1BkLlyZFTOT16EUAXv7Rf8A6B97/wB8L/8AFUf2i/8A0D73/vhf/iqAK95cS3kAgSxulZnU7nVQBgg+ta1AGdrShtPRWAKm6twQRwf3yVM2l6e/3rG2b6wr/hQBX0m3htpdSjgijijF1nbGoUf6uPsK06ACigAooAKKACigAooAKKAIbu3+12U9sW2iaNo92M4yMZqsINUAx9uteP8Ap1b/AOLoAXydU/5/rX/wFb/4ujydU/5/rX/wFb/4ugA8nVP+f61/8BW/+Lo8nVP+f61/8BW/+LoAPJ1T/n+tf/AVv/i6PJ1T/n+tf/AVv/i6ADydU/5/rX/wFb/4ujydU/5/rX/wFb/4ugA8nVP+f61/8BW/+Lo8nVP+f61/8BW/+LoAPJ1T/n+tf/AVv/i6PJ1T/n+tf/AVv/i6ADydU/5/rX/wFb/4ujydU/5/rX/wFb/4ugCOWxv7kRpPewGNZY5CEtipOxwwGd5xyPStKgCjp/8Ax9al/wBfX/tOOr1ABRQAUUAQ3F1BaR+ZPKsa9Bk9fp61BBq9jcyiKO4G89FZSufpkDNAF2igAooAKKACigAooAKKACigAooAKKACigAooAKKAKOn/wDH1qX/AF9f+046vUAFRNcQpOkDSosrglULYLAdcDvQBLTJporeJpZnVI16sxwBQBk27o6TazeIxG4iBCvKRg4BA9W6/QgdquXRtrkxWlxGx89SUOOhAz17GspVYxlZ+X4uxcabav6/hqVvtN/pkK/bYkuLdOGniY7wPVkx+ZB/CtUEMoYHIIyCK1IFooAKKACigAooAKKACigAooAKKACigAooAKKAKcul2M0zyyWyNI5yzevGP5Cm/wBjad/z6p+ZoAT+xtO/59U/M1n3vg3Rb/U7G/ntiZLIs0Kq5Chjjk469O/FAG/WZqQVr7TlmwYDKcg9C+Pl/rQBNqsD3GnSpGpZ+CFBxnBzUc8Us99YMInCR7nduBt4wBXLVhJzdlvy/g9TopySirvv+Q7U3vlEIs4o3Ut++MjgfL0wM9Tzn8Kfpy3iW+29KFwBgoMDoP611HOXKKACigAooAKKACigAooAKKACigAooAKKACigAooAKKACori3iuoGhmXcjdRQBR/sqccLql2FHQEg/ril/sy4/wCgrdf+O/4UAKmkqZEe6up7kI25Ukb5QfXA61o0Af/Z/+Ex6Gh0dHA6Ly9ucy5hZG9iZS5jb20veGFwLzEuMC8APD94cGFja2V0IGJlZ2luPSfvu78nIGlkPSdXNU0wTXBDZWhpSHpyZVN6TlRjemtjOWQnPz4NCjx4OnhtcG1ldGEgeG1sbnM6eD0iYWRvYmU6bnM6bWV0YS8iPjxyZGY6UkRGIHhtbG5zOnJkZj0iaHR0cDovL3d3dy53My5vcmcvMTk5OS8wMi8yMi1yZGYtc3ludGF4LW5zIyI+PHJkZjpEZXNjcmlwdGlvbiByZGY6YWJvdXQ9InV1aWQ6ZmFmNWJkZDUtYmEzZC0xMWRhLWFkMzEtZDMzZDc1MTgyZjFiIiB4bWxuczp4bXA9Imh0dHA6Ly9ucy5hZG9iZS5jb20veGFwLzEuMC8iPjx4bXA6Q3JlYXRvclRvb2w+V2luZG93cyBQaG90byBFZGl0b3IgMTAuMC4xMDAxMS4xNjM4NDwveG1wOkNyZWF0b3JUb29sPjx4bXA6Q3JlYXRlRGF0ZT4yMDIzLTA2LTI2VDIzOjQ1OjM4LjgwMDwveG1wOkNyZWF0ZURhdGU+PC9yZGY6RGVzY3JpcHRpb24+PC9yZGY6UkRGPjwveDp4bXBtZXRhPg0KICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgPD94cGFja2V0IGVuZD0ndyc/Pv/bAEMAAwICAwICAwMDAwQDAwQFCAUFBAQFCgcHBggMCgwMCwoLCw0OEhANDhEOCwsQFhARExQVFRUMDxcYFhQYEhQVFP/bAEMBAwQEBQQFCQUFCRQNCw0UFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFP/AABEIAbECNQMBIgACEQEDEQH/xAAfAAABBQEBAQEBAQAAAAAAAAAAAQIDBAUGBwgJCgv/xAC1EAACAQMDAgQDBQUEBAAAAX0BAgMABBEFEiExQQYTUWEHInEUMoGRoQgjQrHBFVLR8CQzYnKCCQoWFxgZGiUmJygpKjQ1Njc4OTpDREVGR0hJSlNUVVZXWFlaY2RlZmdoaWpzdHV2d3h5eoOEhYaHiImKkpOUlZaXmJmaoqOkpaanqKmqsrO0tba3uLm6wsPExcbHyMnK0tPU1dbX2Nna4eLj5OXm5+jp6vHy8/T19vf4+fr/xAAfAQADAQEBAQEBAQEBAAAAAAAAAQIDBAUGBwgJCgv/xAC1EQACAQIEBAMEBwUEBAABAncAAQIDEQQFITEGEkFRB2FxEyIygQgUQpGhscEJIzNS8BVictEKFiQ04SXxFxgZGiYnKCkqNTY3ODk6Q0RFRkdISUpTVFVWV1hZWmNkZWZnaGlqc3R1dnd4eXqCg4SFhoeIiYqSk5SVlpeYmZqio6Slpqeoqaqys7S1tre4ubrCw8TFxsfIycrS09TV1tfY2dri4+Tl5ufo6ery8/T19vf4+fr/2gAMAwEAAhEDEQA/AP1TooooAKKKKACiiigAoorx3Sf2qvAmv6XZanplp441LTbyFLm1vLT4fa/LDPE6hkkR1sirKykEMCQQQRQB7FRXlP8Aw0t4R/6A/wAQP/DceIf/AJBo/wCGlvCP/QH+IH/huPEP/wAg0AerUV5T/wANLeEf+gP8QP8Aw3HiH/5Bo/4aW8I/9Af4gf8AhuPEP/yDQB6tRXlP/DS3hH/oD/ED/wANx4h/+QaP+GlvCP8A0B/iB/4bjxD/APINAHq1FeU/8NLeEf8AoD/ED/w3HiH/AOQaP+GlvCP/AEB/iB/4bjxD/wDINAHq1FeU/wDDS3hH/oD/ABA/8Nx4h/8AkGj/AIaW8I/9Af4gf+G48Q//ACDQB6tRXlP/AA0t4R/6A/xA/wDDceIf/kGj/hpbwj/0B/iB/wCG48Q//INAHq1FeU/8NLeEf+gP8QP/AA3HiH/5Bo/4aW8I/wDQH+IH/huPEP8A8g0AerUV5T/w0t4R/wCgP8QP/DceIf8A5Bo/4aW8I/8AQH+IH/huPEP/AMg0AerUV5T/AMNLeEf+gP8AED/w3HiH/wCQaP8Ahpbwj/0B/iB/4bjxD/8AINAHq1FeU/8ADS3hH/oD/ED/AMNx4h/+QaP+GlvCP/QH+IH/AIbjxD/8g0AerUV5T/w0t4R/6A/xA/8ADceIf/kGj/hpbwj/ANAf4gf+G48Q/wDyDQB6tRXlP/DS3hH/AKA/xA/8Nx4h/wDkGj/hpbwj/wBAf4gf+G48Q/8AyDQB6tRXlP8Aw0t4R/6A/wAQP/DceIf/AJBo/wCGlvCP/QH+IH/huPEP/wAg0AerUV5T/wANLeEf+gP8QP8Aw3HiH/5Bo/4aW8I/9Af4gf8AhuPEP/yDQB6tRXlP/DS3hH/oD/ED/wANx4h/+QaP+GlvCP8A0B/iB/4bjxD/APINAHq1FeU/8NLeEf8AoD/ED/w3HiH/AOQaP+GlvCP/AEB/iB/4bjxD/wDINAHq1FeU/wDDS3hH/oD/ABA/8Nx4h/8AkGj/AIaW8I/9Af4gf+G48Q//ACDQB6tRXlP/AA0t4R/6A/xA/wDDceIf/kGj/hpbwj/0B/iB/wCG48Q//INAHq1FeU/8NLeEf+gP8QP/AA3HiH/5Bo/4aW8I/wDQH+IH/huPEP8A8g0AerUV5T/w0t4R/wCgP8QP/DceIf8A5Bo/4aW8I/8AQH+IH/huPEP/AMg0AerUV5T/AMNLeEf+gP8AED/w3HiH/wCQaP8Ahpbwj/0B/iB/4bjxD/8AINAHq1FeU/8ADS3hH/oD/ED/AMNx4h/+QaP+GlvCP/QH+IH/AIbjxD/8g0AerUV5T/w0t4R/6A/xA/8ADceIf/kGj/hpbwj/ANAf4gf+G48Q/wDyDQB6tRXlP/DS3hH/AKA/xA/8Nx4h/wDkGj/hpbwj/wBAf4gf+G48Q/8AyDQB6tRXlP8Aw0t4R/6A/wAQP/DceIf/AJBo/wCGlvCP/QH+IH/huPEP/wAg0AerVjeJPEWneEdFvNX1i8jsNNtE8ye4k6KM4AAHJJJAAAJJIABJArgv+GlvCP8A0B/iB/4bjxD/APINeafGLxZZ/GrR7Gw0691vTvDwMjXdtcWM2nXFzKrgKskNxGsqKhVjtdQGJVsYVSfns8zzC5DhHisVK13ZLe77fqz2MpwCzHFwozdobyfZdfm9l576CeK/26fD2m3Yi8P+Hr3W41eRZLi6mWzRgpAV4/ldmDcn5ghAxkZJA9R+Fv7QnhD4szTW+lXctjqMb7V07UtkVxMNpbfGqswcAK2dpJG0kgAgnx/4Z+H/AIe+AtHu49X8EyeICgDz38ekTa1dEb8Ji1iikYAGTBaJBwFLDClh5J8QZND8A+PNO134bQ+IvDkCoskS61oWp6bJFOuVcRm+gjM0bKV3KDJgswYBWUHp4LqT4wqRnh8RHklzLWLVpJX5dvveqttc9niqvkOT4WdPDYap7SLVpcytJN2bkru3kkk3pex+hlFch8NfiDp/xQ8J2mv6cGijlZo5beRlZ4ZFOCjbScHoRnBKspwM4rr692rSnQqSpVFaSdmvM+Kp1I1IqcHdMKKKKyNAooooAKKKKACiiigAooooAKKKKACiiigAryn9k3/k1j4N/wDYmaN/6Qw16tXlP7Jv/JrHwb/7EzRv/SGGgD1aiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAr5xs/Fmm/Cfxve3fiaUmCOR7dryaNjIGZuJguCzFlBJ25OGY8jr9HVwvxE+EPhf4pLbHX7Bp57VSsNzFK8UiKxBYcHBB29GBxk4wTmvkc9yOWa18JiqdTknQnzLS6a0uu3Ra2duh108dWwuGr06EVKU42V20r+dtbeXXucxH+0x8KYbiaaLxBGk8+3zZV025DvgYG4+Vk4HAz0rkPjJ+0j4A1v4cazpWmXH/CQ3moQG1jt2tZY1jYg7ZmaRAPkIDDGTuC4xyy9CP2P/AIdg5FtqH/gY3+FS237JPw5t51kksLy5UZzFLeuFPHfaQffgjpX6VkMssw0FVzFS9pGV0oW5WtHrfW7le+qPjprOa8HCoqaurddtutzmf2K9DvrHwPrOqTz3Asr+8Vba1kRljHlrhpkJOG3FghIHWHBJxgfSFeV/suXU19+zP8I7q5le5uJ/CGkySzSsWd2ayhJYk8kkkkk8kmvVK5cyxrzHGVMU1bme3lsvwPaweHWEoQop3sFFFFeadgUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAV5T+yb/wAmsfBv/sTNG/8ASGGvVq8p/ZN/5NY+Df8A2Jmjf+kMNAHq1FFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAVwHxL+NXhD4S2Zk8RatFBdGPzIdOg/eXUww5XbGOQGKModtqZ4LCvNv21viBqfgb4YWEeia1JpGp6jqUcRNrP5Vy8Co7uUYEMAHEQZl7MFPDYPhnwj+A9npOueC/Fq/8ACQ6/HdanZ3FjY2ljFbTWcRuE8u8vlMkrRwMIrll2KyusakyxlsD7LLckw9bCLH42o1BtpRS1k1/eekTwcXmFSFZ4ehC7Vrt9L+W7Ps/wL8SvC/xL003vhnWrbV4o8eYkTFZYcllHmRsA6ZKNjcBuAyMjmusr82fFXwTi8F+FYvHVjceJvDX2by3GkX1oj6nYXBGIJndJEKW7zRTqs7JGwKptjbduH2V+y/40uvHnwS8O6hqWoLqeqRLJaXcrSB5A8cjKolOSd5jEbEtydwY/ezU5xkdHBUPrmDqudPm5WmtYuzdm9ntutNu9ysDmE8RU9hXhyytffRr03R65RXi2pftXeD9M8HavrzWesXEtl4nuvCFro1vbxyX+q6lbztA0VpEJMOGZWYFmXCqWbaAcegR/Ebw9Do+p3+p6tYaINHhjk1qDUL+3VtHZ4llEd2yyMkTBHVslipBBDFSCfjz3DqaK8ug/aP8AAA+JWu+BNS8RWHh/xJpmo22lRWesX1vbPqU89tBOi2iNJvl4uYlICg7zgA5BPS698U/BfhbxRpvhnWvGGgaP4j1PZ9h0e/1OGC8u97lE8qFnDvudWUbQcsCByMUAdZRXEa78avh54Z16fQ9Y8eeGdJ1q2jkln02+1i3huIkSETuzxs4ZVWEiQkjAQ7j8vNW/CPxU8F/EK81G08KeL9B8TXenMq3sGj6nDdvaklgBKsbkoSVYDdjlT6GgDrKK8s1X9pj4Y2Hw98X+M9P8baD4k0fwtZtd6n/YWrWt08fysY4vlk2rJIyFI1Zl3NgA1Lov7SHw01TQfCWpXHjrw1pT+KLWK60yzvtato5rjeduyNS/7xhJmMhc/MpXqMUAenUVzvjrx1ofwz8J6h4m8S6gumaJYKrXFwY3kI3MERVRFZnZmZVVVBLFgACSBXA6X+0ba/8ACRaJo3ibwT4q8BXWuw3Eujv4gjsmS+eGLzZIVa2uZvKlEe5gkwjLBXAyVYAA9gorz/4X/Gzwr8WPBuj+IdKvksk1LSIddGm6jNFHeW9nKXCSzRq7bFJjkG7JUlGAJwasap8bvh3os2hRah498MWEmvQx3Okpc6xbRtqMUhAjktwzjzVckbWTIORjOaAO4oryrwz+0l4E8UfED4i+E4Nbs7a98BrHJq9xdXkCxKhi8yaRSHJEcOQkjOFCOGU4xk9Dpvxk8A654UuvFOn+OPDd94YtJ1trjWrXV7eSyhmLIojeZXKK5MkYCkgkuox8wyAdpRWJ4T8YaD480SDWvDWt6d4i0eYsItQ0m7jubeQqxVgskbFSQwIODwQQa26ACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKAPKf2Tf8Ak1j4N/8AYmaN/wCkMNerV5T+yb/yax8G/wDsTNG/9IYa9WoAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACvKf2Tf+TWPg3/2Jmjf+kMNerV5T+yb/wAmsfBv/sTNG/8ASGGgD1aiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooA+Pf28PDeuXk3hXXJtN+3eDNL3pdSWUhS5jlldQwdirKisqRqj7WCsWDA7kBr/BXxJ9t8MzeKPDPhiy1WW28Lx6Xr1vps87a5JdW6rbwRRuJF+zxSxLHMHhRjmCQcuqGvrfW9Fs/EWj32lahF59hfW8ltcQ7iu+N1KuuVIIypIyCCOxr4x8S/sN+NUkvdI8PeI9Jfwsbw3dtHqDyRXDsUCr53lwkOyAsqncQNzsqoZGFfpGU5lg8Vl8cvxc1ScNm72lFu7TSa1vrutbPofK43C16OJeJoxc1LppdNbb9OnXqupJ+0lr1lo7+IbvxN4f02HWPEwsLayjsZXtNVbS1eO5ma6UNNEHV4orZXIBZo3IBRAtej/sM+GfEnhv4YX8mr2cVjpGp3i3+mh9wuJQ0aq8jA8CNgkezgE/MeVZSeG8C/sR+IZdY0e38d61pl94Y0sySR2emySNNJuZWMPmNGrJEW3MQGOCz7QrSM4+z6xznMsLRwCyzCzVS7TlLWy5bKKim9HZK/Ra230vAYWrPEvF1o8ttEtLu+9+++n9X/Prw38E/iD8PfEniv446Tp+u69r3h34geJZo/BN7a4S70G5vHa4k06JkDG6kG2ZJlYGZUWL5l2qdD48XGuzeGf2oNL07wR4w1i6+Ilhp2peGhY+G751uom0q3t5VkbydttLG1vIWhnKSn5VVSzKD960V+en058QfErTbhfFX7V/h2X4Z+I9b1bx8un2GgX9poNxNaanIdHt7eMNdhPJgjtrhi5mldApZyrMyMF5m++Eev+Hrz4teFPiPqfxP1mTxXJaukvgrwxZapbeIrZrKG3VGvJrCc200MkMi/v7iFUBSVSu4uf0FooA+ONe+G2uJ4f8A2xjH4f1q6v8AX/DcFlps1xatJdas0fh0RKiMikTuJmdT5e4eYzAcnFZfxu+CHi3xQbfRPBWiXGm3UvwX1Dw9BMkHkQCYXWnGOwaVgEjZ41nRVYgAM54AY19t0UAfCl34HHj/AMK+JbnR4vi5qviGL4bavotra+JPCdlollaLcWwCacRHYWj3DiRE2LD5saFGIK7xuy/FWm39v4Pg1PwxoPjq08Vah4P0exvfDevfD2XWPD/io2kVwiWNxG8QmsWSV2Vnlktk2skgWVTvr7/ooA8a/aE8G+J/iB8H9KfTNKjvPFOk6ro/iH+wVuUWO8ktLuG4ltfMdlTDBHVWYhdwUkjqIf8Ahel549utN0fwn4B8VzxXtvPLq994j0a90OLSohF8qj7TbqbmZnYKsUO4fK7M6gLu9rooA/Pv4K6X4n+Efhvw1NrfgbxZI+q/B6x0C2t7HQru4lTU4Lm8ZrSdY42+yswu4iHn8tOGywCtirN4N1zwf8JfDI0jw/4ys/G9z8PPDdhqPhjVvAp1/wAO+J2tbOVUsLlQhksZUllKSNLLbKoZWKyYZh+h1FAHwD8X/h7468TeJPjXt8O69FJf6t4S8S3FnpemfaU1Kxsre3W+gtZZ4XtZ5o3UssD7jIYQPLYHBufEr4bnx74F+KGoaRB8UvG2o6x/wjlhe3HizwxBpkWoRwatDIVjtI7K1nlaKJ5S0rwlRGdochCF+8qKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigDyn9k3/k1j4N/9iZo3/pDDXq1eU/sm/wDJrHwb/wCxM0b/ANIYa9WoAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACvKf2Tf+TWPg3/ANiZo3/pDDXq1eU/sm/8msfBv/sTNG/9IYaAPVqKKKACivMf2htb1rQfhvFJoGsXHh/UrzxBoOlDUrSGCWaCK71eztZmRZ45Iy3lTSAFkYAkHHFQf8Kb8Xf9F2+IH/gD4e/+VVAHqtFeVf8ACm/F3/RdviB/4A+Hv/lVR/wpvxd/0Xb4gf8AgD4e/wDlVQB6rRXlX/Cm/F3/AEXb4gf+APh7/wCVVH/Cm/F3/RdviB/4A+Hv/lVQB6rRXlX/AApvxd/0Xb4gf+APh7/5VUf8Kb8Xf9F2+IH/AIA+Hv8A5VUAeq0V5V/wpvxd/wBF2+IH/gD4e/8AlVR/wpvxd/0Xb4gf+APh7/5VUAeq0V5V/wAKb8Xf9F2+IH/gD4e/+VVH/Cm/F3/RdviB/wCAPh7/AOVVAHqtFeVf8Kb8Xf8ARdviB/4A+Hv/AJVUf8Kb8Xf9F2+IH/gD4e/+VVAHqtFeVf8ACm/F3/RdviB/4A+Hv/lVR/wpvxd/0Xb4gf8AgD4e/wDlVQB6rRXlX/Cm/F3/AEXb4gf+APh7/wCVVH/Cm/F3/RdviB/4A+Hv/lVQB6rRXlX/AApvxd/0Xb4gf+APh7/5VUf8Kb8Xf9F2+IH/AIA+Hv8A5VUAeq0V5V/wpvxd/wBF2+IH/gD4e/8AlVR/wpvxd/0Xb4gf+APh7/5VUAeq0V5V/wAKb8Xf9F2+IH/gD4e/+VVH/Cm/F3/RdviB/wCAPh7/AOVVAHqtFeVf8Kb8Xf8ARdviB/4A+Hv/AJVUf8Kb8Xf9F2+IH/gD4e/+VVAHqtFeVf8ACm/F3/RdviB/4A+Hv/lVR/wpvxd/0Xb4gf8AgD4e/wDlVQB6rRXlX/Cm/F3/AEXb4gf+APh7/wCVVH/Cm/F3/RdviB/4A+Hv/lVQB6rRXlX/AApvxd/0Xb4gf+APh7/5VUf8Kb8Xf9F2+IH/AIA+Hv8A5VUAeq0V5V/wpvxd/wBF2+IH/gD4e/8AlVR/wpvxd/0Xb4gf+APh7/5VUAeq0V5V/wAKb8Xf9F2+IH/gD4e/+VVH/Cm/F3/RdviB/wCAPh7/AOVVAHqtFeVf8Kb8Xf8ARdviB/4A+Hv/AJVUf8Kb8Xf9F2+IH/gD4e/+VVAHqtFeVf8ACm/F3/RdviB/4A+Hv/lVR/wpvxd/0Xb4gf8AgD4e/wDlVQB6rRXlX/Cm/F3/AEXb4gf+APh7/wCVVH/Cm/F3/RdviB/4A+Hv/lVQB6rRXlX/AApvxd/0Xb4gf+APh7/5VUf8Kb8Xf9F2+IH/AIA+Hv8A5VUAeq0V5V/wpvxd/wBF2+IH/gD4e/8AlVR/wpvxd/0Xb4gf+APh7/5VUAeq0V5V/wAKb8Xf9F2+IH/gD4e/+VVH/Cm/F3/RdviB/wCAPh7/AOVVAHqtFeT/AAN1DX/7W+J2ha94n1Dxb/wjviaPT7LUNUgtIrnyH0nTrsowtYIY2xLdS4OwHBAJOK9YoAKKKKAPKf2Tf+TWPg3/ANiZo3/pDDXq1eU/sm/8msfBv/sTNG/9IYa9WoAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACvKf2Tf+TWPg3/2Jmjf+kMNerV5T+yb/AMmsfBv/ALEzRv8A0hhoA9WooooA8q/aW/5J1pH/AGOfhP8A9SHTq9Vryr9pb/knWkf9jn4T/wDUh06vVaACiiigAooooAKKKKACiiigAooooAKKKKACvPfjp44uvht8M9R8S2oLSafcWUjxrtzLEbuFZYwWBA3IzLuxkbsjBANehV8l/tbfFL+1ofGngWyfNtpWgwXt/wAfeuX1Gx8pOVBG2Ji2VYhvOwRlK9DAUHiMTCFrq6v6XX/DHLiansqUpX16ep9U2N9Bqdjb3drPHdWlxGssU8LhkkRgCrKw4IIIII4INW6+cf2I/iLJ4u+Fc2hXly09/wCHpxbru3s4tXBaHczEg4IlQAEbVjUYHBP0dWGJoSwtaVGW6f8AwxpRqKtTVRdQor4X0n9o7x34G8Fefcarb6XoD654qNx428UaNqOvWVtLBrTW9pYzG2lRrSMpIcTSt5arEFVSRxW+LPxa8dWfxA8deIrfxRo2paHa/BRfEMel6Ld3r6e87STgvbXcF1CxLSpkXMaxu0QjUBWXfXMbH3jRXyhN+0N8RdS8G/EHxp4Xt/DMfhT4cxpFfaVq8F1Nf6zJb2UN5feVdLOFtR5cwjiLx3BZ0LOQGAHrHxql8XeOP2fdWufhlJPF4j1OxtrmwVLhLS4kgZ43mijmbKwzPAZUSQkBHZW3Lt3AA9Xor4ptPj1ovwJ1Hx7rdwnxK8OnRvD1xP8A8K7+Ik91fNqlxBLDsvbDU5bm7Ty83CwSJDI4+ZXKDZl9uH9p74k6DoHiy41fRV1t7HwdqOvQarD4E13QLHTtQtYTILW4N/j7RFLn5ZI3ib9y4ZBvUqAfXdFfMej/ALQnjXwDq3gy6+KV34Zbwz4r8L32uQzaDptxay6fd2tvFdyWrvLcyrKv2Y3LCQLGWMByqg151p/7VHj6w8QRHU9E0lfGPiLwz4ZOm2Rur6OxtL3Vr+4ihE8LTMgWGMo0jRRrJI0bLvClNgB9w0V8x/FL44fEz4OtofhvXf8AhHdW8Q+JtY+zaNrmh+HtUuoo7OK2M93JNpVu81xJKhRkVY59rCVXZowjg+hfs/8AxO8RfEjStei8T6Nd2N7o+oG2h1V9Av8ARbbVrdkV0nhtb5RNEV3NE6lnAaMsrlWAAB61RXw78Nfj54u8HRfHiy1DVL7xBqUmta3P4NivmafbcrrM+mx2KZy3lLNJpoCgEKLggAAAHK+Df7TPjDQ/hv8ADrwZHrN7rXiq9PiLUtU8T6v4c1XxPMLS21m5tYVFpYESMzMUUO0iRxJFtAYsigA+96K+O/EX7WHxD0zwj4d1TVtFg+G1k9tqCap4p8S+ENYu9M+2QXUMFupgVoJ7GC5WUzrNc8IAY/mZS1Tat8RvihrXxs1K68G+J/CotI/hXpniCaOZ7nWNFmuzdX+fsgingysu0D7VydkSAxtkbQD6+orkfhL44PxO+Ffg7xj9l+w/8JDo1nq32Xdv8nz4El2bsDO3fjOOcV11ABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQB5V8G/+Si/Hb/sc7b/ANR7Rq9Vryr4N/8AJRfjt/2Odt/6j2jV6rQAUUUUAeU/sm/8msfBv/sTNG/9IYa9Wryn9k3/AJNY+Df/AGJmjf8ApDDXq1ABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAV5T+yb/AMmsfBv/ALEzRv8A0hhr1avKf2Tf+TWPg3/2Jmjf+kMNAHq1FFFAHlX7S3/JOtI/7HPwn/6kOnV6rXlX7S3/ACTrSP8Asc/Cf/qQ6dXqtABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABXzJ4y/Z98R2vjDX77wz4e8DarZ+JhPFqM2sf2hHIqyytO29RcOpw0cREkQVg5BVUCkj6borqw+JqYZtw6mNSlGqkpHjXwR/Z/0/4U3k2rPZaPBrNxZpbP/Y63qpGGKtKm64uZRINyphtiHC8jnA9loorOtWqYibqVXdlU6caceWCsjxax/Zng8P26x+F/iH428JubnUrmZ9MvLWRJze3bXcqtBcW0sAKSOwSVY1lVSVLsCc4mo/sUeB7nSk0qw1XX9D0s+DpPA09nYz27Lc6exkYM7TQO4lV5XcMrKC33lYZB+hKKxNDwzXv2SvDusSa3bW/ibxTovh3xFFbReIvDmmXkMdjrPlRrCzTboWliaaFI4ZTbSQ+YiANzkn0H4j/CzQfif4KbwzqkU9pZo8M1nc6XMbW50+4hZXgntpF5ikjdVZSOOMEMpKnsqKAPHf8AhmjQ/EFzqk/jzxBr3xNkvtIn0JV8SNaxJaWc5BuEhSyt7dVaXZHulIMgESBWUAg17f8AZlim8M6voGvfEjx54q02/wDD9x4bSHVL+2Rba2miMTuFgt4lmm24xLcLKwIyD8zbvaqKAPIfjB+zH4S+Nnwv0HwF4im1NdF0ee2nt5bOdI7hvJiaHa7FCCrxPIjgKNyuwBGaPGv7MvhLx54o17X7+bU7fUdV03T9OjeynSH+znsbh7i0urUhMpNHK+4MxZTtUFSpYN69RQB4vqn7Mtlr9nHLrHjzxpq3ie11SDV9N8UXF7bJeaZNFGY1W2hjt1tERo3lWRTbkSiRvM3FUK9z8P8AwHJ4Esb6O48T+IPFt7fXBuZ9R8Q3aSyFiAoSOKJI4IUAAAWKNATliCxJPX0UAeKL+yf4M/t3w/qslzq09zonijUvFcCvcRhJLq9nNxLFIFjG6JZvLkReCGhiJZtpzX0/9kvw74e0nwtF4c8S+JPDWteG7jU5bHxDp8tq960N/cSXF1bSrLbvBLCZZFZVeIlTEhVgdxb3OigDyJvgBcw2elrpnxU+Iel6haQT29xqS6tBeSX4lkWRmmhu4JoFYMvytDFGUVii7U+WneCf2Z/CHw8mY6G2pW0H/CK23hAW7zq6LaQy3EiyAlCxlZ7qQsxJU/LhRg59booA5v4e+C7H4a+AfDXhDTJbifTdA0y10m1lu2VpnigiWJGcqqgsVQEkADOcAdK6SiigAooooAKKKKACiiigAooooAKKKKACiiigAooooA8q+Df/ACUX47f9jnbf+o9o1eq15V8G/wDkovx2/wCxztv/AFHtGr1WgAooooA8p/ZN/wCTWPg3/wBiZo3/AKQw16tXlP7Jv/JrHwb/AOxM0b/0hhr1agAooooAKKKKACiiigAooooAKKKKACiiigAooooAK8p/ZN/5NY+Df/YmaN/6Qw16tXlP7Jv/ACax8G/+xM0b/wBIYaAPVqKKKAPKv2lv+SdaR/2OfhP/ANSHTq9Vryr9pb/knWkf9jn4T/8AUh06vVaAKV9fQaXZz3d3PHbWsEbSyzTOESNFBLMzHgAAZJPAxXITfG74eW8c8kvjrw5FDAu6SSTVYFUKBkncXxgDr6V8k/tpfErVNa+I03hFZJbbSNHjiLQJKdtxNJGkpkZeAdqsqgHOMMQRuIr5A+IH/E31Hw94cfi11O5aW4OTh4oVEjR4GD8xx8wII296+/wnCyqYGOLrVGpStZL+80o3fndeh+W47jR0cylgqFJOMG1KTb+ym5WXlZ+p9oH9uDxp8a/iJ/wjfwa0/wAMaNorvNb2niXx41ykeqSAoqC3iiUNESTIV80MHwoOx/3Z7lrP9rT4dW/2qO/8DfFq1SWKWaxmhbS9RkTeqyQ27DZAvyksHlJI5+VsBT8V6fK+ky20lk5s5LZlaBrf920TLgqVIxtIwMY6YGK/U/4X+KW8b/Dvw3rsksc1xe2MUtw0S7V87aBKAOwDhhj2rDPsljk9OlOn7yejv3/4J1cMcQzz6pWp1lyyjrG38u2vmu9upyvwA/aB0P4/eGLi8sba50PxFpcotNc8NakpS90q65BjkUgEqSrbXwMhSCFZXRfV6+Wf2htGt/hP+0R8IPivoltFDqPiDXYfA2vW8JMTalBeIRBJIwypMDQ7vu7n2xqWCouLP/BSF2X9j/xoA2A01grAdx9tgOPzAP4V8f7NSlHl2l+B977VwhNz1cfx0ufTtFfH37Mvx70L4O/sD+BfGXjzV5/sttDc2UCk+bc3Jju7iOC2gViNzCONVVcgKqZJVVJHG/tL6p8OPiV+0R+zbr9x491yxn1B7LUtK02CwM9q8UlzFJbysWlja1eZgY2cLIT5abggjywqL5nF7a6+gSxCUFJb6aX7n3nRXgnx0/bX+Ff7P2q3OjeIdYuL7xHBHFLJoekWxuLhFk+7uYlY0bbhtrurbSpAIYZT4GftsfCv9oDVrbRfD2r3Fj4knjllj0XV7Y29w6x/eKsC0TnbltquW2qxIAVsZ+zny83Loa+2p83JzK/Y98or8hv2XtS03WPgT+1deeM/FOsaLaXsOmyXutWG6e9knknu9qkFgZfOlZY3VmUOsrqzqGLL+gf7Edj4d039lvwHB4V1a81vRPs87R3t/ai2maRrmZpkaIMwUpKZEwGYYQYZhhjrVo+yvr/VrmFDE+2tpa6b387Hu1eNfGT9pHQfhLeDTBBJrevbFkeygkWNIVJGBLIc7WKksFCscAEhQwJ7/wCIPiRvCPgfX9aiMImsLGa4hFxny2kVCUU4IOC20YBBOeK/Pf4WeHX+JXxX0PTtRm+2DUL7zr17uZg06LmSYM4+YsyqwBzklhyOtfJZtmFXDShh8P8AHPr21sfpfDGR4fMIVsdjr+xpK7S66XevZL/h0e5t+1H8SrHSNB1268OaHcaVqz3TwW9rHOZ5IbYr5z5EjBAAX+ZgcbGJXaAT638Hv2jvD3xakSw2HRvEOHb+zZpN4kC85ikwA/y8kYDDDcEKTXWXXww0S+0mHS7qKW4so7u9uzEX27vtQuFlTKgELi6kAxggBeSRmvz01Bb/AOGPxCvYtNv2XUNC1GSGG9RApLRSMgbYcjB28qcggkHIrzcRiMZlMoTqT54S3T9Fe3zvbp3Pey/AZTxPSrUcPS9lVhdxavZpuVrrVbct9L9ux+n9FcU3xEiPhnwvq9vpF/qEniARfZbGzaISgvbvcfMZJEQAJG2Tu6gAA5q3pPxC0DVJLa3Op21jqMwlI027njS5UxO6TDZuO7Y8UqllLKfLYgkDNfXqvTbsn/X9M/LpYOvFNuD0v+F/8nrto+x1VFc8fHHhz+yRqv8Ab+l/2Zgt9u+2x+RgOqE792MBmVTz1YDqRUcnxA8MQ6e+oSeJNJjsEuPsj3TX0QiWYLuMRbdgPt525zjnFV7Wn/MvvI+r13tB722e/Y6WiuKv/it4V0+TVY31m0kn0u5itr6FZ0D25eSJN7BmGI0aVdzdBhh1UituTxZokbaWH1mwU6pg2Aa5QfawQDmLn95wyn5c/eHqKSrU5aKS/rT8xywteKTlBq/k+1/yd/Q2qKwrzxZomn6fDf3WsafbWM0P2mO6lukWN4iVAkVicFcyRjcDjLr/AHhWhY3kGpWcFzbTR3FtMiyRyxOHSRGGVZWHBBByCOCDWilFuyepjKnOK5pRaRdrzj4rfHbwr8Ho4U1y5mm1CdPNh02xQSXDpu27sEhVGc4LMu7awXJBFehyMwjbYAWxwCcDNfHPjjwavijxBeajq3g9vEVxbxSDU9butRksFjkjnEEcGIy6mRoyrAqu3apBO773HjPrXsJSwivNW6N7+iOGviI0ZRjJpXvq9tPx+7s7ntlj+0t4G8RNDBoevWlzfMPNe3un+yusYZEZtsu0kbpY0DLldzqM5IB9YhMhhTzVVJMDcqtuAPcA4GR74H0r5A8L+C/Ek3hCfxLfeHvD2k6bBesLG10mBROsEcxRpZXOQyT+XGCiqrbYYXZgxZI/pT4XTW0/gPSZLS1FnBsYGMMz5cOwdizEsSzBmLMSSWJJJya6cPRxKw8a2IVm9H0s/S7fe9/+G5aOYQrYl4ZWfuqSavqu+1rbWs2djRXBfGT4oH4R+E7LWzpn9rfada0zSPI8/wAnb9rvIrbzN21s7PO3bcfNtxkZyMWf9p34dWviyHw/NrF8l5NrP/CPRXh0S/8A7OfUt5T7It95H2Yy7wyFRLkMpU/MCK1PVPV6K8yX9o34et40bwsNclOoLqf9iNdDTbr+zRqG3d9jN/5X2X7RyF8nzd+/5Nu75ap6l+1J8NNJ8Yx+GbnX7hdTl1iPw9FKmk3sllLqTsiizS8WE27TKXAZFkLJtfcF2NtAPWaK8yX9o34et40bwsNclOoLqf8AYjXQ026/s0aht3fYzf8AlfZftHIXyfN37/k27vlq94M+N/hD4ieKtb8P+HrvUNSv9Fu7iw1CZdHvUs7e5t3CTQNdtCIDKrEHYHLFSGAKkGgDv6QkDk9qWvmXxRpPifxt+0NrXh618Y3+leHo7WO5uorS5kTyoQkW9EU4UOzMDuGQAxJydynz8Zi/qkYPl5nJ8qXm79/Q9bLsCsfOalUUFCLk203omu3XXQ+g9C8R6X4nsWvNJ1C31K1DtGZraQSJuU4IyP8AOCD0IrXr5Vj8OeENBt9DPh+HxT4YE13E9hrGpBjpmpTxuGhe4QPkK+0kHbH8rlsAA49H+NXxe8SfD7wN4XGgeFxqPj7xbqNtoWlaZcSM9pZ3k0UkrzXbxAubeCOGZ3MaliI8DaCWXPBYx4m8ZpXVtndfkn/n0bNMywNPCNSoybi7/ErNfc2n522tqkex0V88+EfCfxw8G+Pbe/1/40eGfHGgXV2RqPh+98OR6SdPt2Dtus5opXd3VvLULPuDIWJYMAT1GsftX/CvQ/itH8NrvxSf+E3eZYf7It9Ou52UtAlwHZ44mRYxFIrmRmCKoclhsbb6h4p69RXgXh/9u34E+KvGA8L6V8Q7K+1qS8jsIIY7W52XU0lylsiQSmMRzbpZFw0bMu3c+disw5LwH+2RaXvjr4ov428ReH/DPhPwfcanF9iGkaiLiKG0vEtBcTahIFtpGlk3bbaBGc+bEoZmV1oA+qqK8I8H/tv/AAS8f2eq3WheOob+PS9NvNXvY/7Pu45YLO1SN55mjeJXChZkK/LlzuCbijBbnwn/AGyPg38cfFR8N+C/G1vq2t+U86Wctpc2rTIpIYxGaNBLjDEhCThWOMAkAHtdFfNurf8ABRT9nbRr7ULa4+JdnI1i5SWa0sLy5gdgyqRFNHCyTcsp/ds3GW6AkfSK9KAPK/g3/wAlF+O3/Y523/qPaNXqteVfBv8A5KL8dv8Asc7b/wBR7Rq9VoAKKKKAPKf2Tf8Ak1j4N/8AYmaN/wCkMNerV5T+yb/yax8G/wDsTNG/9IYa9WoAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACvKf2Tf+TWPg3/2Jmjf+kMNerV5T+yb/wAmsfBv/sTNG/8ASGGgD1aiiigDyr9pb/knWkf9jn4T/wDUh06vVa8q/aW/5J1pH/Y5+E//AFIdOr1WgD4N/bT+Gup6P8RpPFyxyXOkaxHGDOkZ2280cax+WzDIGQqsCcZywAO0mvkDx9/xKNS8PeI35tdMuWjuRjhIplCNJxknacfKASd3bFftJfWMGqWc9pdwRXNrPG0UsMyB0kRhhlZTwQQSCDwQa5Cb4I/Dy4jnjl8C+HJoZ1KyRyaVAysCMEYK4wR19a+/wnFKp4GOErU25RtZr+6043XlZep+WY7gt1syljcPVSjNtyi0/tJqVn53fofl/p8L6pLbR2SNeSXJVYEt/wB40pYgKFC53E5GMdcjFfqf8L/CzeCfh34c0KSKOGeysYorhIzuXztoMpB7guWOfevktv2IfGXwV+IP/CS/BrUPDOs6Mkk1xaeGvHi3Lx6XISjIbeWFt0pDCTb5pUJlSS7/ALwd093+1n8RIPssdj4G+E1q8sUU19LM+qajGm9WkmgQboG+UFQkoBPPzLkMMM+zqOcU6UKa5VHV37/8DudfDHDtTIalapW96UtI2/l3+9/p9x+0LrMHxY/aI+D/AMKNEuop9R8P67F4516eEGVtNgs1JgjkUYAM7Tbfvbk3RsVIdc2P+Ck3/Jn/AIx/672H/pZDXpXwB/Z+0L4BeGbizsbi51vxDqkv2vXfEupOXvdVujuJkkYkkKCzbUycBiSWZmZpf2kPguv7QXwd1vwKdX/sJtRe3ddQFt9o8ryp45f9XvXdkIV+8Mbs84wfkIzjGcLbL/M+7lTnKlUv8Uk/y0Pxs8MT3a3nwcb4vW2sXPweWSddOigkaGE2v2t/tbRNGrEsJiTKFxKyBFVlBiZfqr9sdtHb9tf9nk+HjZHQfsehHTzpxT7Kbb+0pPK8nZ8vl7du3b8u3GOK+qdS/Yq8Nax+yjpnwUutVuJotNjM9nrzQKZYb4yPKZ1jzgKXllUx7s+W5Xfn56840n/gmzaaTf8Awhvj8QLie78BThpt2lKI9QiXUZL1ERRLmBgZpELFpAcqwVcEHt+sU5S5m7Wv/wAOeasLWjFwSvezv1uraHF6z8YfD+r/ALXXjFfgz8CovGvxa06K4hvPEGq6wLaKGSB0t5pI7eRvLCjbEnmq0btvYdGYv5V8GX8XSf8ABT7Rbjx1oWleGfFtxJPcajpeibfs8DyaRI45DuC7Kys7Bmyzsc819KeJv+Cfep2fxq174jfDr4uar8PtR1qS4muoo9NF02+dy8yhhNGDEWKsEZW2lVbJKqRf8N/8E+7Xwf8AHLwh8SdO+I2vXF9o6h9SfVk+232rzkyh3kuXfCK0UiQ7VjJCR/e3MXEKrSSdnurdbmjo15STa2lfpbf77+p8FfCX/k0f9qT/AK7eGf8A06PX6Qf8E6P+TNvh7/3EP/Tjc1wnhT/gnFpnhf4cfF3wYvji6uNN8dSWL20505Vm077JcSTwhz5hWfLMgbAiyFbG3cNv0B+zp8Gz8Afg14f8BHV/7d/sk3H/ABMPs32bzfNuZZ/9Xvfbjzdv3jnbnjOBGIrQqRai+t/wsXhMPUpTi5LaLX/k1zpPiR4bbxj4C8Q6JEkck99YywwCQkKJSp8skjoA20/hXxb+zatza/EG70KXwpp2r6nuWeMaq72s9hPbPuBSQRuyHOQy7RuKgEgAg/fdfP8A8ZP2U9H+IV/ea3ol0NE1+4JkkQrutLl8N8zKBuRmJXLrkcElSWJr43NMHVq1KeKoK8obrTVfNNH6nw7m+Hw1Cvl2Mlywq2tLX3ZedmnbbZ9Ox6b/AGx49/6FTQv/AAoJf/kOvir46afJdfEux8H6Z4W03QbqySCySz02RZjczyhGVmnaNHkZleNfnzgqxz8zE+833wd8Uah4QbQv+EG+HcOsNbNDN4iWIIHyrLujiS3Vo5QCrbs7QwJCgEKOg+Bn7NunfCWVtW1C4j1nxI6BUnEe2K0BUB1iBJJJOQZDglcAKuWDcWKw+JzHlouLUd23bRdV8Kd9vLQ9XLMdgMhVTFqUZTs1GMebV9JN88kla+6Ute+j7WL4cxL4d8F6TNqExj8OGL97AXhe42WkluMMjho8+ZuOCeF2nIJNO1D4cW0kPiGGya30621PRotIhtorRTDbeWbkhtgIVlP2nlMAYQgkhuO4or6b6vS7f1a35H579exN78/5fzc222+v4bHByfDua8m8QS3Wowyvq8dzE/l2exIxLBbQ8guS2BbZ5PO/HAUZi1z4YpqmpSXsWp3Fk8lw8wjt5ri3TDW0MIBME0TMQ0CEEnG0uu3JDD0Gik8NSkmnHfUqOYYmDTjO1lbZbfd5HnejfC+TRdJk06PULcWb3mn3giitXURtapZoEUtKx2kWnG4lhvGS23LUrf4NpB4i/tRtUaUG48827efGm4ajNeKSsc6q5Xz3UeYrjcqsABuRvUaKn6nQ093b1L/tPGK9qm++i7W7dkeZr8JWktdE0+41OO40vSNPXTraJIZoJii3NvIjtNHMpDqlrEAVC/PufGCIx3Wi2t1ZaRY299ef2hfRQRxz3bRiMzyBQGk2jIXcQTtHAzitKitadCnTfNBGFbF18RFQqyuk29lu9/v69+uyEbpXz7rmtSaB4w1jTJtQki0u7+2LnG2Ai4dC7uoRmZ0MLxxsCAPMkLKwKhfeb9VazuFe3N2pjbNuApMvH3cMQvPTkgc8kCvI/A0OoeK/EWpWmqfBz/hDvDlrFF9nvdcv7OW6nkYMWjigtXnVI02ryZVHz/KpwwHfQqxpy99Xi/vT6NenmfPZhha2Jpp4eSjOL67NPdPrZ+WvmjxXSrCTw9odlZXWsReJ9cWzWzv9fbS4baW4j8zzxErKu9YlkZmEe7aCN3zFmY+7fAnVo49D/s5pJZJXmklQ7TsjACgx5Pf+Ljj5+uSRWDonwp8WX/xF03VNX0Tw3oPha3huI7nSNN1RtQa7kOPIlLS2ETJt+bciuQTsIOFYSerN4F8NySWUjeH9MkksbkXlq72cbNBOFZBKhK5VwrMNwwcMRnBNd2JxVKph1RpJ3vdt2173PDy/KMXh8d9bxEotWslG9kuiSfRdjxP47eH/AIn/ABG8VQ+DIfCtvc+C317w/rFl4ntbmGMWUVrdpcXsd3HJMJHkJt18ryYipEwDMpUmvJfF3wq+NXjDWtEvNc8M+J9c13RPG1hqvmJ4nsrPw2mk2mredFFYWMMyGWXyGQlr2IyARNiUssaN90UV5J9ifEvg39m/WvCsUngrXfh54r8aW8XieW9t/EH/AAsS5stAms3vTexXE1kt7uWeJmAMSWbI0sQbfhi4528bWPB/gT4W/D2HRbbXdE0/4uWf2XxvZ6nZz2N/GdXlumWJEmNwt2haSOWNogq+RcHzG+UN9+Vx2m/CHwHonjC58W2Hgrw7Y+KrhpHm1220mCO+lZ/vlp1QOxbJySee9AHyt4N/Zv1rwrFJ4K134eeK/GlvF4nlvbfxB/wsS5stAms3vTexXE1kt7uWeJmAMSWbI0sQbfhi498/Zz8C614B0vx5Frdh9gm1XxtresWoEscnnWtxdM8MpKMcbkKnDYYdCAeK9eooAQjIIrwG88Gx/CHxxFqNt5+vWWtJNa3NtcLvuGVmQv8AMOHJYg8gZ5B67q9/pMDuK8LNsr/tKnH2c+SpBpxla9tVfS6Tulaz0PSwWOng3JWvCStJbX7a9LPXQ8K1fVG+IHheHwVoFvez7kjSW51YIhjiiZSGJTq2QoJI6DgEnNafx8+HHjXxz4Q8MX3grWNN0v4geFtXt9a0+TUUb+z7mQRS288E4VWcRSQ3E6/LhgSuGBGR6x9jh+1fafJT7Rs8vzto3bc5xnrjPOPWrdRleXV8JOdbFVvaTlZaLlSjG9la711d3d79hYrFxrwjSpw5Ypt93d2u7/JfcfDXxS+EPj39qJV0vUv2dvCfwv1XWxEPEXjzXrjStWv4LdWWOSGxaKOSR5zAFCTSCMIFZV2kI49Tl/Z91LUrX9pnTLTS7DQx49tItK0S/YRhHt10KCzQOI8uscc3nAKwBA3FVw2T9JUV9AecfEXi3wh8a/itZ/A/wzffBrT/AAVZeAvFei67qd/Y65ZnT5IrKcQGLT7dHaQRmGVpwsuwosLRje23d1PxI/Z98Z+JvgzoQ0yxgk8U+D/idqHj+x0K6uUjTWIl1i/uYbXzwWWFpoblGVnBCttVwuWZfrSigD4g+JHgn4w/HTxp4s1u8+D9v4Lt3+GWveGLKSbV7C41C/vbqKFo4ZZIpSqwiUSLFk4U+c7FPNCjqpf2dPE2ueB/2ZtHuLO40TUvCvh+50fW9W0+6hW90Uz+HpbJ2glDElhcGLBj3DcityFyPrWigD85Pix8L/jz4k/Y8svgFYfAWye60xLfTh4jsvEenLZzR2ssLpewRyOkoknKTFw6oyhyxLM7Kv6NL0paKAPKvg3/AMlF+O3/AGOdt/6j2jV6rXlXwb/5KL8dv+xztv8A1HtGr1WgAooooA8p/ZN/5NY+Df8A2Jmjf+kMNerV5T+yb/yax8G/+xM0b/0hhr1agAooooAKKKKACiiigAooooAKKKKACiiigAooooAK8p/ZN/5NY+Df/YmaN/6Qw16tXlP7Jv8Ayax8G/8AsTNG/wDSGGgD1aiiigDyr9pb/knWkf8AY5+E/wD1IdOr1WuR+Jvw9t/ih4Rk0G41TUNE/wBNstQg1HS/K+0W89pdw3cLqJo5IziWBMh0YEZGOa5j/hTfi7/ou3xA/wDAHw9/8qqAPVaK8q/4U34u/wCi7fED/wAAfD3/AMqqP+FN+Lv+i7fED/wB8Pf/ACqoA9Voryr/AIU34u/6Lt8QP/AHw9/8qqP+FN+Lv+i7fED/AMAfD3/yqoA9Voryr/hTfi7/AKLt8QP/AAB8Pf8Ayqo/4U34u/6Lt8QP/AHw9/8AKqgD1WivKv8AhTfi7/ou3xA/8AfD3/yqo/4U34u/6Lt8QP8AwB8Pf/KqgD1WivKv+FN+Lv8Aou3xA/8AAHw9/wDKqj/hTfi7/ou3xA/8AfD3/wAqqAPVaK8q/wCFN+Lv+i7fED/wB8Pf/Kqj/hTfi7/ou3xA/wDAHw9/8qqAPVaK8q/4U34u/wCi7fED/wAAfD3/AMqqP+FN+Lv+i7fED/wB8Pf/ACqoA9Voryr/AIU34u/6Lt8QP/AHw9/8qqP+FN+Lv+i7fED/AMAfD3/yqoA9Voryr/hTfi7/AKLt8QP/AAB8Pf8Ayqo/4U34u/6Lt8QP/AHw9/8AKqgD1WivKv8AhTfi7/ou3xA/8AfD3/yqo/4U34u/6Lt8QP8AwB8Pf/KqgD1WivKv+FN+Lv8Aou3xA/8AAHw9/wDKqj/hTfi7/ou3xA/8AfD3/wAqqAPVaK+c9J8I+PL/AOM/irwi/wAb/HA03StA0jVYZV0/w/5zS3VzqUUgY/2XgqFsoioCggs+ScgL2/8Awpvxd/0Xb4gf+APh7/5VUAeq0V5V/wAKb8Xf9F2+IH/gD4e/+VVH/Cm/F3/RdviB/wCAPh7/AOVVAHqtFeVf8Kb8Xf8ARdviB/4A+Hv/AJVUf8Kb8Xf9F2+IH/gD4e/+VVAHqtFeVf8ACm/F3/RdviB/4A+Hv/lVR/wpvxd/0Xb4gf8AgD4e/wDlVQB6rRXlX/Cm/F3/AEXb4gf+APh7/wCVVH/Cm/F3/RdviB/4A+Hv/lVQB6rRXlX/AApvxd/0Xb4gf+APh7/5VUf8Kb8Xf9F2+IH/AIA+Hv8A5VUAeq0V5V/wpvxd/wBF2+IH/gD4e/8AlVR/wpvxd/0Xb4gf+APh7/5VUAeq0V5V/wAKb8Xf9F2+IH/gD4e/+VVH/Cm/F3/RdviB/wCAPh7/AOVVAHqtFeVf8Kb8Xf8ARdviB/4A+Hv/AJVUf8Kb8Xf9F2+IH/gD4e/+VVAHqtFeVf8ACm/F3/RdviB/4A+Hv/lVR/wpvxd/0Xb4gf8AgD4e/wDlVQB6rRXlX/Cm/F3/AEXb4gf+APh7/wCVVH/Cm/F3/RdviB/4A+Hv/lVQB6rRXlX/AApvxd/0Xb4gf+APh7/5VUf8Kb8Xf9F2+IH/AIA+Hv8A5VUAHwb/AOSi/Hb/ALHO2/8AUe0avVa4f4Y/DFPhrH4jkk8R6x4p1LX9TGq32pa0LVZnlFrb2qqq20EMaqIrWIABM5ySTmu4oAKKKKAPKf2Tf+TWPg3/ANiZo3/pDDXq1eU/sm/8msfBv/sTNG/9IYa9WoAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACvKf2Tf+TWPg3/2Jmjf+kMNerV5T+yb/AMmsfBv/ALEzRv8A0hhoA9WooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooA8p8N/wDJ0/xD/wCxM8Nf+l2u16tXlPhv/k6f4h/9iZ4a/wDS7Xa9WoAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKAPKf2Tf8Ak1j4N/8AYmaN/wCkMNerV5T+yb/yax8G/wDsTNG/9IYa9WoAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACvKf2Tf+TWPg3/2Jmjf+kMNerV5V/wyd8EP+iN/D/8A8Jex/wDjVAHqtFeVf8MnfBD/AKI38P8A/wAJex/+NUf8MnfBD/ojfw//APCXsf8A41QB6rRXlX/DJ3wQ/wCiN/D/AP8ACXsf/jVH/DJ3wQ/6I38P/wDwl7H/AONUAeq0V5V/wyd8EP8Aojfw/wD/AAl7H/41R/wyd8EP+iN/D/8A8Jex/wDjVAHqtFeVf8MnfBD/AKI38P8A/wAJex/+NUf8MnfBD/ojfw//APCXsf8A41QB6rTcjOO/pXlbfsofA9QSfg58PwO5Phex/wDjVfmx8G/jZ8E/HH/BQDXPDlx8LvATfDHXgnh7Qc+G7HyEuYWbyLlf3eP9IdpVyACwkgVuIxQB+vtFeVf8MnfBD/ojfw//APCXsf8A41R/wyd8EP8Aojfw/wD/AAl7H/41QB6rRXlX/DJ3wQ/6I38P/wDwl7H/AONUf8MnfBD/AKI38P8A/wAJex/+NUAeq0V5V/wyd8EP+iN/D/8A8Jex/wDjVH/DJ3wQ/wCiN/D/AP8ACXsf/jVAHqtFeVf8MnfBD/ojfw//APCXsf8A41R/wyd8EP8Aojfw/wD/AAl7H/41QB6rRXlX/DJ3wQ/6I38P/wDwl7H/AONUf8MnfBD/AKI38P8A/wAJex/+NUAeq0V5V/wyd8EP+iN/D/8A8Jex/wDjVH/DJ3wQ/wCiN/D/AP8ACXsf/jVAHqtFeVf8MnfBD/ojfw//APCXsf8A41R/wyd8EP8Aojfw/wD/AAl7H/41QB6rRXlX/DJ3wQ/6I38P/wDwl7H/AONUf8MnfBD/AKI38P8A/wAJex/+NUAJ4b/5On+If/YmeGv/AEu12vVq+XdB/Zl+D8v7SPjrTJPhR4IfTbbwn4fuYbNvDlmYYpZLzWlkkVPLwGdYYgzAZIjQHO0Y9N/4ZO+CH/RG/h//AOEvY/8AxqgD1WivKv8Ahk74If8ARG/h/wD+EvY//GqP+GTvgh/0Rv4f/wDhL2P/AMaoA9Voryr/AIZO+CH/AERv4f8A/hL2P/xqj/hk74If9Eb+H/8A4S9j/wDGqAPVaK8q/wCGTvgh/wBEb+H/AP4S9j/8ao/4ZO+CH/RG/h//AOEvY/8AxqgD1WivKv8Ahk74If8ARG/h/wD+EvY//GqP+GTvgh/0Rv4f/wDhL2P/AMaoA9Voryr/AIZO+CH/AERv4f8A/hL2P/xqj/hk74If9Eb+H/8A4S9j/wDGqAPVaK8q/wCGTvgh/wBEb+H/AP4S9j/8ao/4ZO+CH/RG/h//AOEvY/8AxqgD1WivKv8Ahk74If8ARG/h/wD+EvY//GqP+GTvgh/0Rv4f/wDhL2P/AMaoA9Voryr/AIZO+CH/AERv4f8A/hL2P/xqj/hk74If9Eb+H/8A4S9j/wDGqAPVaK8q/wCGTvgh/wBEb+H/AP4S9j/8ao/4ZO+CH/RG/h//AOEvY/8AxqgD1WivKv8Ahk74If8ARG/h/wD+EvY//GqP+GTvgh/0Rv4f/wDhL2P/AMaoA9Voryr/AIZO+CH/AERv4f8A/hL2P/xqj/hk74If9Eb+H/8A4S9j/wDGqAPVaK8q/wCGTvgh/wBEb+H/AP4S9j/8ao/4ZO+CH/RG/h//AOEvY/8AxqgD1WivKv8Ahk74If8ARG/h/wD+EvY//GqP+GTvgh/0Rv4f/wDhL2P/AMaoAT9k3/k1j4N/9iZo3/pDDXq1UdJ0my0DS7LTNMs7fTtNs4UtrWztIlihgiQBUjRFAVVVQAFAAAAAq9QAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQBR1bS7bXNKu9OvYvOsruFoJ4txXfGwKsuQQRkEjg14bpfwt8Haj8dvHXhSfwtpB8NW3g3wwbbSUso1trZlv8AXijRIqgRsp+ZWTBU8gg819AV5T4b/wCTp/iH/wBiZ4a/9LtdoA9UVcAflzzTqKKACiiigAooooAKKKKACiiigAooooAKKKKACiiqOr6gNJ0u8vjb3F2LaF5jBapvlkCqW2ouRljjAHc4FAHm/hv/AJOn+If/AGJnhr/0u12vVq+QvDf7YPwfm+OniXxNYeN7DVrLXvDHhnTtJtbDdLe3t4b7WgLZLYDzRLmaEFWVdvmoWKhgT9dqSQDgjPagB1FFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQB5O37TPgo3l/b21v4v1P7De3GnzXGl+B9bvbfz7eZ4JkWaGzaN9ssbqSjEZU81wGi/GzRbP49+MfEUvh/x+mi6h4Z0PT7a6/4V7rx8ye3utWkmTaLLcNq3UByQAd+ASQwG/8AB7xLqXhX4S6ldaZ4T1jxhO/jnxVEbHRJbKOZFOv6kTKxu7iCPaCoUgMWywwpGSOi/wCFyeLv+iE/ED/wP8Pf/LWgBbP9pjwHPrmj6TeT694futYu1sNPk8SeFtV0e3ubplZkgSe7toozKwVgqbtzEYUE8V6rXl8d9N8aNJ13wf4y+FXibQPDmqadNbXcmu3emGC4jcBGhU2V9NKrlXYhgFA2Ehg20Gf9m3XL/wATfs6/C3WdVu5b/VNR8K6Vd3d3O26SeaS0id3YnqzMxJPqaAPSaKKKACiiigAooooAKKKKACiiigAor59/aM+O2p/Cv4kfDzw5B4w8FeAtK8RWWrXV1rvjW1aaBJLU2YiijxeWqhn+0yE5Yn5BgdaT4LftPDxpJY6Nr1sl5q14NcubHXNDg2aXqtjp11HB9rhV5nZVl89CgDSKdjESFSpIB9B0V8Z+KP24tamuhqHhXSrEeH73RvCmsWK6zbP9rC6rqTW0qyiOfaCsQUqFJAbJJYcV3/x9/aV1v4QfEi80SHRWk8P23gLVvFc+spaR3TW89rNCq5ha8gaSMB9rINrM00OHVRKyAHkvwt/YA0/wP+3x4r+Jxtox4Qgtl1nRbXadkeo3TSLMoH3cQlJXCgAKJ4Mcoa+56+dte/bO0DwtcaxFdeEPF+q2uhXOmWGrazpthbG1gnv4LeW1AQ3PmsHN1GmEVyrHDHDKzbNr+1dodxplyh8KeJ4vFkXiH/hF08GNHaNqs18LdLlghW5Nv5S27mYytMECo3zbsKQD3CivMrL9oDwvdfBnWviVKL+x0fQ4LyTVbC5gAvrGW0Lrc20sQYgSo0bLgMVPBVirBjm3X7RdroXhnWtY8SeCfFvhZ9ONnHa6ffW1rcTavNduYraCye1uJoZZXlATZ5ilC6M+xGDEA9forwG+/bA0KxuNN0o+C/FsvjC812Tw7J4TSKyGoWt0tqbpTKWuhB5TwYdZUlZCG5IKsFraD+1lZXmntDbaHrvjjxLLq2tWsOh+G9Lht7pbTT7hoZrh1mvTH5asYow5lVpXlXZCpJVQD6Horw5v2sPD2rmB/CPhnxP4/s20Oz8RXNz4btraRrWyumlELNBLPHPI5+zzZihjkkUptKhiFPuNABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUV5f8ePi5qHwj0fwvNpGg2viHVPEPiKz8PW1teai1jCklxvCyPKsMzBVKDICE4PtggHmvhfTv7T+B95F/wAI74x8T7fiB4pb7J4I17+x7xf+J7qg3vN9ttN0Qzgp5jZZkO07crkf8IkP+iR/tCf+HRP/AM0Vc78Jf2hvEPhG1bwnZWXw+1rV9R8d6/pcVnD4p1Zbo6jJfT391bCAaIz7LZbpRJcYEQAVmZAwA0W/bzkX4Z6b8QjF8Mv+EOv9W/sOHUx401Uqt7tZ/JlT+wt8J2IzZkVRtKtnDKSAeo/BHQxpviy6lHgb4p+GM2Tp9r8beMjrFk/zxnYkP9q3e2U4yH8sYVXG8btrbn7J/wDyaz8G/wDsTNG/9IYa8Ztf24tQvPFGo+HYdO+Hr6zYeJLfwjc23/CW6wNmqzvOkNtuOg7SWa1nG8EoNnLAEZ639nDxX4s8Aaz4W+Avi/w9o1pfeG/AtldRa1omtTXsd3HAyWXzRSWkBiZmRmwC4AIGT1oA+jqKKKACiiigAooooAKKKKACiivlb47ftI+NbCPxbbeBdChg0jwx4l0LQNT8VTXyGZbi5urJp4orRoHWSMQ3SRNIZFZWmJVSE3AA9t8R/DH+3/i94K8c/wBpfZ/+Eb0/VLD7B5G77R9sNqd/mbxt2fZem1t2/qu3nn/iz8GdY8ceNvDnizw14pt/C+taVp+oaRK93pR1CO4tLvyC4VfOi2So1ujK5LKCSGRhxXlfjD9s7UNK+IV14JtNF8L/ANp3c+q6dp6QeMoJ9VgmtrSeeGa8sEt3NtDKYeCXd0DrvjDHZXNTftw+I/hj8B/AXiXxbpXhPUtVvPBVj4ivpNS8ZW+l3eqPIgLLY2n2bdNMyBZWQLHErSeVG8hUkAG9b/sF/Z9D0vTv+E5z9h0Pwzovmf2Rjf8A2RfG683Hn8ednbtydnXLdK9B/aA/ZrufjVrNxfWfieLQBfeEdX8IXsc+mG83wXpgdZoiJo/LeOWBCQwcOpZQEOHEXx0+KHjHwz8QfgrZeEzarpHiTUNQj1WO8mEbPDHpk9wgX9xIcr5bSjay7miSMnbIzJ5B4G/bc8ZeF/ghoevePfB9vqOo3Hw8l8Y2moQawqPqxtjbpOs0KWwS1ZzcLIuwyLtOCFbKgA9W1L9lD+0ND8dad/wlPl/8JRrHh7VvN/s7P2b+y109fLx5vz+b9gzuyuzzejbfmo+Pv2NdO8c6n4m1a51LSb/UtQ8VReKLG31/QU1LTYGXT4bGS3ubV5VFwjRxswZWiZHKFT8hDL4o/ak8WeG7fUxH8N7PU7/w14cTxT4rt7fxEQmn2csk3kRWztaj7TcmG3nlaNhCilAqyPuBq5J+1vayaJ4m1Wx8O/bLLSfFWieG4HN9s+1x6iti63OPLOzYL8fu+d3l/eG7gA6rwP8AA2fwF8Fb/wAEaTren6Rql4ly/wDa2ieGrGxtYZ5SSGSwVDEUUbV2yF2ZVwzkncPNIP2G9LvPCXjLStRufDFlJrjadPbWPhjwoum6Ha3FlK80M0umvcTJcNI7hJtzr5kSKg2Eb60If2sNd1bWNCey8CxJ4K8SeINU8OaT4mXWQ0xksorxnnktTBhVd7KURjzGJVSzbPlVqv7Kv7R3jLx/p/w10Tx5oNvb6j4m8Gf8JFaeIINRSV79oWt45zNbJAiW7MbhJFCM67Tg7WyoALPgn9jeHwr4l8Ja6NV8O2F1ofiGbXZLPwt4PttFspUayltUt0jikLjb5zOZJpJmJyBtUgDE1X9g7T7u3t7lNU8O6trkGq63eLJ4t8Jx6xp/2fUbo3LR/ZHnXbNC4j2TrIMgSBkIcBeg+Kn7Xw+F/wAVrfwrc6P4cns5dY0zSVSbxfBFrc/2t40+0w6WImdoY2kAJeSNmCMyqU2se9+KXxd1rwv4u0jwb4N8K2/jDxhqGnXes/Yr/Vf7Mto7O3aJHYz+TKTK8s8KIgTB3OzOgUbgDzHxp+xa/izw5pPh4a/4Z/six0mLTIZ7/wADWR1DS5AzGa60qe2e3WxlkLKwHlyqjorKByG+nooxHGqgsQqgfMxY8epPJPuea+a2/a+1hW8fTt8O2htPDurWXhqwt5taUX+o6xeC0+zWzwiExwoGuwJJRLJtCEqsucBmofta+JrHxFa+DB8OrKT4ht4oTw1c6X/wkZFhEJbCS+hu1u/su54mijOVMKupDgKxChgD6corz74H/FCT4wfD2DxDcaUNCv1vr3Tb3TVuvtK29xa3UttKqzbE8xS0JYNtGQwyAcivQaACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACuF+JnxZ0H4X6clxqszPcynEFjbrvmmI6kDICqM8sxA7cnAPct0r86/i94pm8WfFDxXPcXf2o2mqXWnoOgjSCZ41QDtjbzjqdx6k16uXYOOMquM3otWePmmMlgqKlBat29D0Dxd+1l4v1mV00YW2gWu4FfKQTTY2gEMzggjOSCqqRwO2T5zefEzxfqglF14n1iZJDkxG+lCH/AICGwB7Yrl6fH90/WvuKeEoUV7kEj4KpjMRWd5zb/rtsaY8SavxnVb049bhuf1qeDxVrVvIskWr30bryGS5cEfiDWPT17Vv7OHY5+eV73O80n4yeOdHmElv4p1KRmXkXU32hfwWXcB+VeueCP2uLlZ4rbxTp0bwswU31gCrIMdWjJO7nkkMOOimvm9f4fpUqAYORXJWwGGrq0oL1WjPQoY7E0HeM389Ufo3omu2HiTTYdR025jvbGYZjmhOQecEexBBBB5BBBwRRXz7+x5rF1dWfirTS+6ytJbeaMbsYkkEgcflHGfxor4HE4f6vWlSvex+gYTEPE0I1bbn0rRRRXIdoUUUUAFFFFABRRRQAUUUUAFFFFABRXE/Gj4ir8IvhL4w8bNZNqP8Awj+lXOpLZqxXz2jjZlQsAdoJABbBwCTg4r5U8WfHb4pfB34neJ5vFV/pXiPWbrwn4dg0rT9BtL7+y4Lu+1O7t/tDWayzyysjE58gmSdIY1UKW+UA+4a8A/a+tb5tF+GGo2mk6tq8Gj+PtJ1O+TRtNuL+eG1iMpkl8qBHkKqCM7VPUDuK8+X9p34qXlx4Z8OW+i6fYa3q3jIeHo/EXiDwrqmlWF7YvYTXS3UFldSRzrJG0TRtE0jKxj+V1EgKcbrX7WfifQ/Hll4h19JJn8HaH41g1nRtFuJrfTdXutMntUjmSNmfaGBIBfeYi0gywGWALHxQ8I+EvGPxk8bfEjw7ceMvDHiDxJ4Jk8JfaP8AhU3iOWW2uJJAraisi26EzC3CwrjBXYp3EAqfKtb/AGW/CeoeBtT8C2/jn4mf8Idcavp2qWlrdfCHWTcWn2fSp9OlXfDYxJI8sb27eYUBzbgsHZ2c/R2pftJfEf4Y+Mrfw947XwlqEUMWiaxqOs6JaXFvb2+m3109jMpWS4kKvDc+U4m3FWi3kopUkYWvftPfF+Pw/q2q2Fr4NsP7H+Gdv8Rru1v9Pu5nmWWW9YWalblPLYQW0YLkMBIH+VlcbADwnSP2ZdI8Oatd65p3jrx1deILnxboHjSS81b4PeI50m1HT4bvz2lVYkJW4ubx5cKylFGwFvvD6e+Euq6x44/auXxLdafrEttB8Pl0u71m68I6loNnLeDUTIY4Y75N33GDYDvxnng4xNe/ay8e+AYPFun6/p2hatrv2Lw7c6E+i6bemGF9WuJrcLcQrJLNceQ0W7ECq8wG1VjZhtkt/wBpb4q6pd+GvDVpo+n6breq+MDoMfiPxB4T1XTLC9sW06e8F1BY3Msc6yo0LRNG0jKxTIcCQFAD69or48+CPxg+IHxS/aK8GnV9Y0+ysYfDPiW01jRtPtrhbO6urHW4rMXEKtcEIzDymUursi+emW80MnS6z+0z4o0Pxh428PXS+GNOn8Jprd/fXWptPDbpYxQWcumyOys7LvN+gkZVbcLacIgYBaAPp6iviZf2wfiJY6P8QIGh0LVtX0RPDM2nXl14W1bQIZP7T1IWcyyWt5IZmVRuZJkYKcj5ThlG7eftDfFvwl4p8RWety+C9R0zwr4y0Pw1qT2Gk3dvNqEGpC0YTQh7t1t3iF2uVYyhyG5TA3AH15RXhX7IXiLVvEvw98T3GsaneatcQ+NNftIpr6d5nSGPUZljjVmJIRVUKqjgAAAACvdaACiiigArxDx5+yj4d8eeKdW1Z/EXiXRbPWb/AE/VdW0PS7mBbHUL2yZGguJFkhd1bEMKsI3VGEKEqWG6vb6KAPAbP9jjwxZa1o91/wAJP4ol0nR9Zv8AW9O0Bri1Wxt5b37R9qQ7bdZZVcXUoDSyO6DAR1BYNiyfsJ+GJPCM/hpfHHjWDS7zw1a+FNREM9gkmoWFr5wtUkkFoGUxLMU/dGMSKiiUS5Yt4N4E+Onjqb4FW11p/jKw+HyeDfhXY+L4bS00ewjttbuZXuA0ckRi2xWy/ZEhCWwhYPck7j8iixpXxK8X+F/Enxp8faR4nfSLuHxh4aM3gNrW2ddQe+s9Ohkild42mVmVisZiZdrQuWVwcKAfYvjT4MaZ43k8DzXOr6rY3/hG8+12d7ZmASThrWS1mimDxMhSWKZwxRVYEgoyEVwutfsX+Cdd8C6D4TuNU19NO0bwhc+C7eWO4hEzWU/2ffI5MJBlH2WPDBQvzNlTkYyf2mviZ450b4jeEPBfhKTXNKttR0nUtYuNT8ORaTJeu9uYUSFf7VkS3ES+eZZdu6UhEC7F3tXl3hP9o74seJPGXhPw5d39nFr+vW2j+KRp+jC0vrb+y4rW8/tGKCeISq6z3NlEqsHkZBfIFfgGgD6A+Jn7Mfh/4lajNdya94i8Pm/0ZfD2tQ6JdQxJrWnK5Zbe48yJyoG+Zd8Jik2zyKWIKhc3xF+yF4X17xJeahB4h8SaLpl5qml61c+HdMntk0+W808QLbyENbtIPktokZVkCsFDbQwDD5en/ao+M9j8IdQ8Yvqd1E+ueBr/AMRwPqbeH0j0y7QpJE2mQQTy3M9sokaF1u42dSkTMysXSvbNG8dfETw78bvDPwz1Hxvea82t/YPEEeqXmm2aS/2ctveG/t9sMKIF+0QWaBiC6reLhmZSaAING/ZT8WyfGnRtXvL6LQ/h3omt6tr1loNn4ga/T7Rexzo3lQHToDbhmuZZmDTzhCWSMKrs1eveBf2d/Dnw9vPh9cade6pNJ4J8OzeGNO+0yxMJbWQ25Z5tsY3SD7LHgrtHzN8pyMeC/tb+CFl8feLfihfp8MPFOi+B/BsJl8LeNbI6jPuNxPO5Cl1W0eZUSKKUrKWZGXbjNdt+0pFqXjDxH+zr/Z9/feENQv8AxW0ouRbxSXVkG0e+Z1CSq0YkC7lG9WCtyVYDBAOi8Vfsj+H/ABV4i1nUW8VeKdLsdW1+z8UXOjafParatqVsIBHMS1s0rLi2jzE8jRZJYIGClev+J/wVtPiRq+m61a+JNe8F+I7C0udOj1rw3LBHctaXGwzQN58MqFS0UThgodGjBRly2fnTwb8Y/iD4+/4Q3wXffESTwjdOfFzXvjRNPsTcag2k6h9lgjEcsTW8Y8qQTyhYwSIflaMFs87p37W3xA1b4P8AxB8T32sW+ialafCfTvE+loLWFY01CWW/ia6iEiMXWRobcqjFlG5VAyxyAfRy/sr+D18K+NdAN5rklv4p1K31iW7m1Ay3djeW8VulvPBO6ly8bWsMgaYylnBLlwxUpoP7Lfh3R/EOmeIrvXNf13xLaeIG8SXGs6nNbme/ujZSWSrMscKRrEkEm1UiWMAop5+bd4zZ/Ez4oeIX+Ld5Z+Pmgm0rxRp3hzRtFZNKtd0csdhNMtrLcQ7ZL11knSBZnMe6TDK3ylfb/wBnHxdqviXRfFVjreq6/qWq6Hrj6fPF4nsLC3v7LNtbzLBK9gzWs5xMHEkW3CyqrLuQswB2Hwz+Gum/Cvw/daPpM93cWtxqd9qrPeurOJbq5kuJFBVVG0PKwUYyAACSck9fRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAjV+NXxS8X6j4W/aA+JZtJQYX8TakXgflSftUgzjscd/pnOMV+yjfdr8Ufjt/yXz4lf8AYzan/wClUlfS5H/Fn6fqfPZ0lKlFPv8Aodzo/wAUtOvkAuY5LOTocjev4EdfqQK6W18R6dcgeXdxNkZxuGR+FeCafXQWQyuDX2Z8TKkk9D2oXlvgEzxjjuwFKNQtl6zxnH91gTXmNgSyrnmugsgMZoOdqx141iEkCNJJsDB2KR/6FirEEdzfNiR1tYiOVibe5/4EQAPoAT7isbT+g+hro9P+6KDGU2lofTv7HtulvB4pjij2xj7IQo6DPnE/Uk5JPU55opf2RZAq+KQT2tP/AGtRX59mV/rc/l+SP0XJ/wDcad/P82fSNFFFeQfQBRRRQAUUUUAFFFFABRRRQAUUUUAUtU0yz1zTbvTdRtINQ068ie3ubS6iWSKeJwVdHRgQyspIIIIIJBryvRf2S/hb4f0vWdPtvDtxPb6xYQaXeNqOsX17K1tA263RJZpneIwtho2RlaMqpQqVBHsNFAHgHij9jXwVrreFobFr7S7TS/EbeJNQlbUb241HVLj7JLbozag1x9pR0MkbLJ5jFREFUAHI7PSv2cvhxow0lbfwtbOmmWF9pcEd1LLcLJb3rK14syyOwuDMyhneYOzEsSfmbPplFAHg+v8A7Ifg9vhx4k8I+GUbRYfEsNtp+r6hqst1rN1Lp8Xyi2ilubhmjCpuWP5ikRYsEJr5I/4as8EeP/8Agob4n+G+o2NvJ8PPEXhz/hWckscjQpcSo8zABkK7VaSe4t1MZyd8bAiv0lvbVb61ntmeWJZo2jMkEjRuoIIJVlIKsM8EHIPIr5gtP2XfhPdfGTW/BJ8BaND4as/CelXMFpb2/kvDOb2+/fpMhEizYQAyqwcgYLGgD2jV/gR4F17+2v7Q0BLr+2NKtdGvC9xNlrW1d3tlUh/3bxvKzrKm2RW2sGyqkU/Dv7Onw/8AC93p13Y6LM9/Yas+uRahe6nd3d3JfNbvbmaaeWVpJj5MjIBKzAAjAGBj0eFWjjVWdpWAALtjLe5wAMn2AFTUAeXxfs3+ALS9tr2y0u/0u9thqix3Wl61fWc2NRl868BeKdWYPLiRQxIjdVaPYVBFf/hn/R9b8UfEvVvFi2XiKHxtZW+jz2aWP2dY9OhSRVhZw7M8haaVjKCn/LMBV2An1iigDyG0/ZV+GdtJq8z6Jf31zq/2E6jd6jruoXk92bO5W5tWllmnZ3aOVVKlmJ2gJkp8tdDq3wR8Fa5da7c32i+fNrmq2Gt6g32qZfOvLIQC1lwHAXYLaH5VwrbPmDbmz3tFAHOeCfh/oXw60y707w7Y/wBnWd3f3OpzR+dJJuuLiVpZny7MRudmbaCAM4AA4ro6KKACiiigArI8TaPca9od7p9prN9oFxcR7F1LTRCbiA5+8nnRyR5xkfMjDnpnBGvRQB5ton7PPw80nwj4J8OXXhLSdftPBtrHa6JPrljDeXFnsVB5iO6Eo7GJGZl25Kg9hjYk+EPgWbxgnix/BXh1/FSTeemuNpUBvVkKLGXE+zeG2Kq7s5woHQAV2NFAHOeNfh74W+JWlxaZ4v8ADWkeKtNinFzHZ61YRXkKShWVZFSVWAYK7AMBnDMO5qzb+EdCstXttWt9F0+DVLWy/s2C+itY1nhtNyt9nVwu5YtyI2wHblQcZAraooA4iL4LfD2C48QzxeA/DMc3iMONakTR7cNqgdizi5OzM25mLHfuySSeTU9t8N7C3+JUvjR7y9uL9dL/ALHs7OQxra2Nu0iSSiJVRW3SNHEWLs3+qQLtGQea+GMjy/GD4xgszJHqenRqpPA/4ltuxwO33q9VoA5PxJ8LPBXjTXdM1zxB4Q0HXNZ0wq1jqOp6ZDcXFqVbepikdCyEN8w2kYPPWtrUNB03WLrTrm+060vbrTpjc2U1zAsj2spRkMkTEEoxR3XcuDhmHQkVpUUAcbrnwg8B+JtAXQdY8E+HdW0Rb2TUhpl9pME1sLp2dnnETIV81jLIS+NxMjkn5jlvin4O+AvHE1lN4k8D+HNflsrdrS1k1TSbe5a3hZSrRIXQlUIJBUYBBxiu0ooA5e6+GfhDUdJ1jTLvwpolzpmtSCbVLKbTYWhv5FVFVp0K7ZWCxxgFgSAijsMXPCPgvw/4A0WPRvDGhab4c0mN2dNP0mzjtYFZjlmEcahQSeSQOTW5RQAUUUUAFFea+P8A42aN4Ivho9tb3PiPxRIG8nRNJQyzZEZcGTGdi4AJ4LYbcFYZNVfCPx1sNT1xfDnibS7vwZ4nY7Y7DUiDFcEu6DyJsASAlQAcAMThd2Ca5frVFT5ObX9e19r+W5zfWaSnyc2v9aX7+R6pRRRXUdIUUUUAFFFFABRRRQAUV4R8WPh74W+JP7SXw40zxf4a0jxVpsXhPxJcx2etWEV5CkovNEUSKkisAwV2AYDOGI7mui/4ZO+CH/RG/h//AOEvY/8AxqgD1WivKv8Ahk74If8ARG/h/wD+EvY//GqP+GTvgh/0Rv4f/wDhL2P/AMaoA9Vrg/jV8Spfg58N9Y8af2Hd+IrHRUF3qFlp7qLlbQH99LErYVmjXLlWZQVRvmyADjf8MnfBD/ojfw//APCXsf8A41XCfGn9in4e+NPhtq/h7wX8O/h34V1zVEFmNefwraM+nxOcSzRKkalpQm4Jhk2swbcNuCAd78Ef2lvhv+0VpH2/wJ4otNXkRA9xp7N5V5bDOP3kDYYDOQGwVJBwx616nXzP+zf/AME+/hH+zb9n1DTdF/4STxXFhj4h1wLPOj9cwpjZDg5wVXdg4LNX0xQAjD5a/F/48RR33x4+IqqyRXS+JNQUh2CJIv2hwOTgKw4yTgEAknd979oGr8U/j4Afj78SM8j/AISTUT/5MvX0mR/xp+n6nz2cfw4+pgWitbzLFKjRSsu8RyKVcqejbTzj3rfsvu1iadcy+QYd++FjloZAHjY+6ng/iK3LLyWYM9sjHPAR3jUfRUZV/SvstT46Rt6f91a6Kzrm7WIMw2u8XoEIIH/fQJ/Wtu0sRIm155mBzkiQof8Ax3FO7OSVjprDjAPWrtt4jtyVSyDalMwG1bb5kB6fM/3V54OTkehrEsdCsWCebG10yncGuJGkI/76JrrtLiSJVCIqBegVQMUanLOx9K/sb2cqr4tuLpwbmb7HkRgFY1Hn7UB74yTnuWPbABV/9kX/AFfin/t1/wDa1Ffn+ZP/AGufy/JH6Pk+uBp/P82fR1FFFeQfQBRRRQAUUUUAFFFFABRRRQAUVznjzx7o3w18Nza9r81xBp0U1vbk2lnNeTPLPOkEMaQwI8kjPLLGoVVJywrjP+GlvCP/AEB/iB/4bjxD/wDINAHq1FeU/wDDS3hH/oD/ABA/8Nx4h/8AkGj/AIaW8I/9Af4gf+G48Q//ACDQB6tRXlP/AA0t4R/6A/xA/wDDceIf/kGj/hpbwj/0B/iB/wCG48Q//INAHq1eU6N837U3jH/Y8GaJ/wCPX2rf/E0f8NLeEf8AoD/ED/w3HiH/AOQa8+0j42aNb/Hzxd4kl8PeP00W98M6Lp1rdf8ACvNfO+eC61SSZNostw2rcW5yQAd+ASVYAA+l6K8p/wCGlvCP/QH+IH/huPEP/wAg0f8ADS3hH/oD/ED/AMNx4h/+QaAPVqK8p/4aW8I/9Af4gf8AhuPEP/yDR/w0t4R/6A/xA/8ADceIf/kGgD1aivKf+GlvCP8A0B/iB/4bjxD/APINH/DS3hH/AKA/xA/8Nx4h/wDkGgD1aivKf+GlvCP/AEB/iB/4bjxD/wDINH/DS3hH/oD/ABA/8Nx4h/8AkGgD1aivKf8Ahpbwj/0B/iB/4bjxD/8AINH/AA0t4R/6A/xA/wDDceIf/kGgD1aivKf+GlvCP/QH+IH/AIbjxD/8g0f8NLeEf+gP8QP/AA3HiH/5BoA9Woryn/hpbwj/ANAf4gf+G48Q/wDyDR/w0t4R/wCgP8QP/DceIf8A5BoA9WrnfHnjXT/hz4P1fxPq63J0nSYGu7x7WEzPFAvMkuxfmZUXc7BQW2qcAnAPF/8ADS3hH/oD/ED/AMNx4h/+Qahuv2jPBl5by28+h+PJoJVKSRyfDfxAyupGCCDY4IIOCDQB5f8ACX9qL4YX3xb8exaV4usdfv8AxZ4jsYdDsdHf7TPeAaRZbnCr/q0QrKGaTaqmNwSGUivqyvzM/Yz+Cfgn9nH4+fEvxpd6L46uLBbprHwip+HviCR4bKVVlkkb/QSVcBlgBO1j5cpxtdSft3/hpbwj/wBAf4gf+G48Q/8AyDQB6tRXlP8Aw0t4R/6A/wAQP/DceIf/AJBo/wCGlvCP/QH+IH/huPEP/wAg0AerUV5T/wANLeEf+gP8QP8Aw3HiH/5Bo/4aW8I/9Af4gf8AhuPEP/yDQB6tRXlP/DS3hH/oD/ED/wANx4h/+QaP+GlvCP8A0B/iB/4bjxD/APINAHq1eL+INQ8VfEz4i+IfBulasvhbQdDNm9/f2RY6hdGVDJsibgRLgYJ5IKg/MGKjS/4aW8I/9Af4gf8AhuPEP/yDXnng343eHdM+LXxG1ebTvF72mqf2b5Edv4L1me5TyoGRvOt0tDLBknK+aq7xkruAJrixPvOnC+knZ+lm/wBDkxF5OEL6N6/c3+h6T4D0/wCH3wpvr7wro8kFjqlrYrqGoy3JPmNAoAM0sxAUAbs7QQF3EhQCawPFfxh+DPxAhGg67q9pqMDzqq+db3CIj5wGEwQBB2LhgME5OCa8X8dfEDwn4k+KHjW81O88SeHtD1Pw+liLyTwzqUVzBIJYGVpbZrcSpEzJtLOqq24KGy65p6l+0ZrPgjwKuheJfAWo+IvBFjpZhvJovBviDT1hgij5lmnuLNVj2hd5cKAuC25cZHmOtO0qUILlV1s31tqk7r1s/Q87207OlCC5VdbN9eqT09bHstj/AG78Eb7wpDp2vr4t8B67qNppltFqEu+6tBKhCNDMo2tFhQ2MAYACgZLV79XzRdapYax8J/grc6XDeQWP/CWafFHDfzedLHseZCm/+JQykKcD5QvA6D6Xruwb3Ufh0a+avp5HZhHulto18108gooor0j0AooooAKKKKAPKfEn/J0/w8/7EzxL/wCl2hV1HxN+IVv8L/CMmvXGl6hrf+m2WnwadpflfaLie7u4bSFFM0kcYzLOmS7qAMnPFcv4k/5On+Hn/YmeJf8A0u0Kl/aW/wCSdaR/2OfhP/1IdOoAP+FyeLv+iE/ED/wO8Pf/AC1o/wCFyeLv+iE/ED/wO8Pf/LWvVaKAPKv+FyeLv+iE/ED/AMDvD3/y1o/4XJ4u/wCiE/ED/wADvD3/AMta9VooA840D4oeJdY1i0srz4Q+M9Dtpn2vqOoXeiNBAP7ziHUZJCP91GPPSvR6KKAGtX4sftAQsPj38RJACY5PEWolWHIJW5cMPYgkZHXDKejDP7Tv0r8V/jhcPbfHz4lqu1o28S6jujZQVbFzJjIPcZOD1Gcgg19Jkf8AFn6Hz+cfw4+pzmnc4Irfsu31rFs/s8jAqXtiWLMuPMQ56BcsGUD1Ysa2rWOSMKQUn3HhIWOVHuWVR+tfZ3PjZG9ZfeFdDZ/d/CuasHnYKRaP/wB/oT/7PWzZ38oYp/Z1yTyMhosf+h0XRySR1On9vpXRWBAxk1x9rLqLLtitreLK/LJLMXwfdVHP/fQrds9Ge+IN/dSXCcH7NF+7hzgcEDLMM84ZiPbrRr0OaVup9T/sc6gNQ/4S94MNbKbREm6iRh5+4r7DgZ7kH0oq3+yGiiHxQoT5VFoFCjAA/fcUV8BmX+9z+X5I/Rsn/wBxp28/zZ9I0UUV459AFFFFABRRRQAUUUUAFFFFAHlX7S3/ACTrSP8Asc/Cf/qQ6dXqteVftLf8k60j/sc/Cf8A6kOnV6rQAUUUUAY+teKtG8NzabFq+r2GlTapcrZWMd7dJC13cMCVhiDEF3IUkKuSQDxxUsOv6Zca3c6NFqNnLq9rBHcz6ek6tcQwyMyxyNHncqsUcBiMEowGcGvmX9uvwLB8TL74JeGZ7y50z+0PGZji1CzYpNaTDTb14Z4yCPnjkVHHOCUAORkV4RefGvxPqnj/AMfSaveSeC/EsNr4P8HeNtTsZPIOlK+r3sF3dQTMoWJZYZElim58tblGDBlyAD9JKzbfXtMvNYvdJh1G1n1WxjimurGOdWnt45d/lNIgO5Vfy5NpIAbY2M4OPiz4s+Ltc+E3j/xX8PvDnjDXk8B3R8L/ANt6zqGtzX134WGoahLa3Xl31w7ywebbxwsvmOTCZDKmzcDXIfEzVNR+DPxE+Pun+CvEepNFM/gbSNR1PVvEM8lxodpdT3qzbtQmW4mhBWVSJmEjQi5DIAFQAA/QDXPEWleF7EXus6pZ6RZmaOAXF9cLDGZZHCRpuYgbmdlVRnJLADJIrUr4N+JF58RvhVofiTT5fFlno9s+seF5YdBsPH9/4g1bT2l1WKGd2nuoIbhLa4i2r5bM65STbgOwrpfh5r+teCvjF4Sv/FOvax4r0zxRrup6fpfjDwv40bUdE1SSSW7aKzutJlOy0MEUAXdaBwskOHkwzggH1Z4y8e+GfhxpKap4s8RaT4Y0x5RAt7rV9FaQmQgkIHkZQWIViBnJCn0q34c8TaR4x0W11nQNVsdc0e7Ba31DTrlLi3mUEqSkiEqwBBGQTyCK8Y+OmijXfjP8NZPDvjPS/DPxO0uy1e40PTvEWky3thqtvJFClygKyQkSoVgbMcpdUaQlGVjj51vvH3i+GbT/AALZ+H9H8J6vrPxIksPGepeGfFl7p+k6vqjaZHdiK3v1gkuLIzyqiyRRoJN8bIXBmdiAfoPRXw1r0PjmxtPCfgzV/HfkWU/xTstOaz8L+N73U9TsbGXT7mabTrvUGjt7lgzqXXfmRUdMOfLQibxX4g8SeBfipqX7P1h4i8SNL4u8RaPqug6ndazdXOpW2htGz6okV27tIoiOnTKCzFh9tX72d1AH2/RXwR4D8QfFHxpDcfESbxHpHhrV7fxxPp13Nr3xEvraytFivjbHSX0X7GbQF7cKqHzDMzyJMJAzAD239lbS7rW9Y+JvinV/EHiHWNRi8b6/pNpDfa1dyWVpZx3hWOGO1MnkDbtOH2b1DFQwUBQAfRVFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAV4ZrGraj8Hfip4r8Uarolxe+EfEH2ASatYt5jaeYoWjJlhA3bc5JYcAFQNzHaPc6K561J1EmnZp3X3W/Uwq03Us07Nar7rfqfNXgyDRfjV8cPG2rw2o1rwbNo0emyzzRtEskjNCwUK2HBBiYhgAVKAgjKkw/Gv4QD4dfAnxtqmmeNfGFzLpHh+7ew0q61bdazypAwgtjEqLuV22R+WpBbcFHJFev/EnwZr/AIkis7zwz4ouvDesWCymFeJLS53LgLNGQQcEDDYO3JIUnGPDf2hfhT4gvvgN8UPFvjrxLJqes6Z4X1a80zT9JZ7ay06aO2uHjkQjazsvyEMwDDbglwAR5n1dxbg6fNJtu+y117307annewabi4c0m277LXXvfTtr5G7a3lr8TrnwdoPw78Pyw+C/DGtWuoSa3OzQwfu97NFGjjzJGO85Y87mG4ANur6RqpZ2cGnWsNrawx29rCixRQxKFREAwFUDgAAYAHQVbr0aFF0k3J3b+S06JdvvO+jRdJNt3b+S+S7BRRRXUdIUUUUAFFFFAHlPiT/k6f4ef9iZ4l/9LtCpf2lv+SdaR/2OfhP/ANSHTqTxJ/ydP8PP+xM8S/8ApdoVL+0t/wAk60j/ALHPwn/6kOnUAeq0UUUAFFFFABRRRQA1ulfip8fI3j+P3xJDqyk+JNRIBGMg3LkH6EEGv2V1i+eBTtJA9q+M/wBqT9mq0+K2sXHibRp4tL8TyKqTmdT5F4FAVS5UFlYKqjcA2VUKV53D2MrxUMLWbqbNWPJzLDzxFJez3TPhbTzlvwroLL7tJrngDxH4FlkGu6Nc6fHGoZp2CvCoJIH71SyZ46bsjIyBmjT2EiKVIYN0IOQa+8hUhUXNB3R8PUhKm2pqzN3T/urXQ2PGa56w4210Nj3ycZrQ4JG9YdAfY10en/dFczaTwwywxPKiSSnbGjMAzE9Ao7/hXpHgv4eeIfFTp5Fk2m2rfeu9SjaPb8207Yjh2I+8AQqsAcMKwq4ilRV6krE08NWxEuWlFs96/ZHkWNfFILYP+if+1qK9C+GvhzT/AAH4eGnWIklDN5ss8zZeRzwScAAAAAAAcAdzkkr88xmIjiK8qkVoz9JwOEeGw0KU3qv+HPUaKK8x1p2fVr0sxYiVhknPAYgD8AMVxpXPUlLlPTqK8koquQj2nket0V454X8df8I54xutNvHjj0u6dWMsny+TJ5agNnH3ThVOeBwcgA59jqZRcSoyUtgoooqSwoorzb9oj42aV+zv8G/E3j/V1E8Ok2xa3sy5Q3dyxCQwBgrFd8jKpbaQoJYjCmgCn+0t/wAk60j/ALHPwn/6kOnV6rXlX7S3/JOtI/7HPwn/AOpDp1eq0AFFFFAGZqnh/Stbu9OudR0uz1C4024+1WM11bpI9rNtZPMiZgSj7XZdy4OGIzgmqVz4D8NXl3rd1P4d0qe61y1Wy1WeSxiaTULdVZVinYrmVArMoV8gBmGME10FFAHJ+H/hX4L8J+Fr/wAM6H4P0DRvDmoeZ9s0fT9Mhgs7nzECSeZCqBH3IAp3A5AAOQMVB4f+DvgDwjpOp6Vofgfw3oul6pb/AGO/sdP0m3ghvIP3n7qVEQCRP303ysCP3j8fMc9nRQB4l41/Zf8ADureHdK8NeDtN8OfD/w0uu2OuatY6R4fije/a1uIp41VonjSNi0IVndJcqcADrXe6X8I/AuieMLrxZp3gvw7p/im6MjT65a6VBHfTF/vl51QOxbvknPeuwooA57xl4C8M/EfSU0vxZ4d0nxPpiSidbLWrGK7hEgBAcJIrAMAzAHGQGPrVX/hVXgr/hBv+EK/4Q/Qf+ENxt/4R3+zIP7Px5nm4+z7PLx5nz/d+9z15rq6KAOV0f4W+C/D2k6ZpWleENB03S9Lu/t9hZWemwxQWlzhh50SKoWOTDuN6gH5m55NaVx4T0S78R2niGfRrCfX7OB7e21SS2RrqCJzl40lI3KrEDKggHHNbFFAHHXPwh8C3XjZfGc3grw7N4wVldfEEmlQNfhlQICLgp5gIQBQc8AY6cVvaPoGmeH47tNL06001Lq5kvLhbOBYhNPI26SZgoG52YksxySTkkmtOigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAryn9rL/k1j4yf9iZrP8A6QzV6tXlP7WX/JrHxk/7EzWf/SGagD1aivKf+Fs+OZP9X8EvFSf9fGraMv8A6DfNR/wsv4kSf6v4O3qf9fHiCxX/ANBdqAPVqK8q/wCE8+K0n+r+FOnp/wBfHiqNf/QbdqT/AIS34yyf6v4aeE0/6+PGs6/+g6a1AHq1FeU/298b5P8AmR/h/b/9zlfS/wDuKWj7d8b5f+YL4Atv+4xfTf8AtqlAHq1FeU7fjfJ/y0+H9t/2zvpf/Zlo/s343y/8zF8P7b/uAX03/t6lAB4k/wCTp/h5/wBiZ4l/9LtCpf2lv+SdaR/2OfhP/wBSHTqZ4Y+G/jhvippfjPxl4q8P6r/ZejX+kWthoWgT2H/H1PZSvJJJLez7tv2FVChV++SScAU/9pb/AJJ1pH/Y5+E//Uh06gD1WiiigAooooAKKKKAKt5YpdoQeDiuR1rwQbsEqC30ruKKAPB9c+FD3BY+Uc/hXmeufszaVqd01zPotnLcEkmZ7ZWfP+9jNe833x48Laf8erH4Q3n2u38U32ijXLSSSJRaTxmSZDCr7t3nAW8z7SoBRGIJwQINH/aA+H2ofDy28bazrWn+DvD9zqd5pEVz4ou7exV7i2uZ7d1DM+0lmtpWUBslRkgEEClJxd0yXFS0aPm6X9lDTS5YWUkZJziJ5EH5KQK09O/Ze0yEASaZHcKRyLlTL/6Fmvpn4jfELw78M/Al14s1JJL3T4/JS2g02MTT300zrHbwQLkBnlkkRFyQuXBZlUFhxLftOeE9DsPFr+K9B1zwTrPhqC1u7rQdWtYZ724guZPKtZbYWks8c6yzhoVCOWEi7WVcqW19vV25397Mvq9K9+Rfcjm/C/wOttDi8qy06G0jJyUt4lQZx6KPevStF+H72oBKsBXOz/tN6PoK+KIPFHhPxH4O1fQfD1x4o/sjVVspJ7+xgD+c9s9tczQuylQpRpFYGRCQFYNXq3h7WoPEmg6bq9skkdvf20d1EsoAcLIgcBgCQDgjOCRnuaxbb1ZskloiO00COCPaTk0Vr0UhhXk3iaRI5tWkkuvsMatMzXW5B5Kgtl8sCo29fmBHHIIqf42ftAeGPgLZ6XP4hj1C6l1KSRLe206FZJGCBS7ksyqFBdBy2SXGAQCRk2fiSPxdokGv6XA3lanbrfWsN2wjbEq70Vyu4KcMASN2OcZ7uMk24p6nDUr05ylRjL3luu1/69TjP7d0b/oqX/kzpv8A8ZrU0vxp4c0+3aOfxtYam7MWEt1e2quowBtHlhVxxnkZ5PPTEv2rxn/0DNB/8GU3/wAj0favGf8A0DNB/wDBlN/8j1mrx1/R/wCZ4cJVKb5k3841H+chVtNA8Yyy3djqMN6ykLK9jcrIAccBsbgDge1eyeDVMPhqwiMjSiJDGrNjO0EhRwAOAAPw55rzHRpdVkhkOrW1nbyhvkFlcPKpXHcsikHPsa9T8J/8gG1/4F/6G1buTcVc93DNyipPf0a/B6mxXzNqP7cXhe5/a+8MfAPw3bDX9UujeLrmrxz4t9MlhtJZ1tkAU+bLmLEmCFjyFyz71jf/AMFBfj94w/Zu/Zz1HxX4J0r7brM15Dpv9pSRGWLR0lVx9sdNpVtrKkah8L5k0ZbcB5b/AIQ/D34weMPhZ8SLPx/4a1yWz8Y20k80eqzxx3UnmTRvHK7CZXV2ZZXBLAnLE9cGsztP6XvEfibSPB2i3Ws6/q1joej2gDXGoalcpb28KkhQXkchVBJAySOSBX5R/Ez46v8A8FKf2yPhx8N/CenS3Xwj8N6r/aV6bz7QkWqW8TK1zdXEaj90jRq0EG9QwNx8zIZyifG2p+Nvjp+2p46tdDudV8SfEjW7qdZoNJjdja2zHbEZVgXbBbINyhpNqKASWIyTX66/8E6v2IX/AGU/A97rXilLO5+JHiFF+2NAqyDTLYYZbNJeSxLYaVlIVmVANwiV2APcf2lv+SdaR/2OfhP/ANSHTq9Vryr9pb/knWkf9jn4T/8AUh06vVaACiiigAooooAKKKKACiiigAooooA4H4u/EDVfh/o+jyaJotnruq6tq1vpNta6hqLWMAaXd8zyrDMwA2ngRkn2rH/4ST43/wDRPPh//wCF5ff/ACmpfj7/AM05/wCxz03/ANqV6rQB5T/wknxv/wCiefD/AP8AC8vv/lNR/wAJJ8b/APonnw//APC8vv8A5TV6tXyDo/7UmseIPFnjfTL/AOMvwh+H0+i+KL/QrPQ/EWmvJfyQwS7I5WJ1aAsX/wBmMAkHHoAD2j/hJPjf/wBE8+H/AP4Xl9/8pqP+Ek+N/wD0Tz4f/wDheX3/AMpqy/E37S+meDb/AF+3n8O+I/EWm+FBCninxRoljD/ZulSMiyS7kkuBPJ5cTrNItuk5jRwCSwIHJad+1tPpvxF+Mln4k0C8i8M+EbjS7PRZdNjhuLjVJ7uGMwwRolwzSzXEkymLCqoQr5hjYNQB6D/wknxv/wCiefD/AP8AC8vv/lNR/wAJJ8b/APonnw//APC8vv8A5TVw3xF/ao1nwq3hKOD4deJNO1S/8Y2vhy+0LWLOE3dxDPYXVxE9nNFdG1ctJCiFzMUTEokCYyPVvhV8VbD4r6Xq9xa6XqWhX+japNo+p6TrCRLc2d1EEcoxikkjYFJI3VkdlKupznIABhf8JJ8b/wDonnw//wDC8vv/AJTUf8JJ8b/+iefD/wD8Ly+/+U1cR4g8XfGnS/jj4e8BQeLfAZg1rSdR1eO+k8HXpa3W2mtkETKNVAcsLoEsCuNn3Tu4peGv2xoZJ7HRH8OeIfHHie/vtdgt7fw1o9vZqU0y6WCbK3F+yr94FWaUb8YCqzLGQD0T/hJPjf8A9E8+H/8A4Xl9/wDKaj/hJPjf/wBE8+H/AP4Xl9/8pq5Pwv8AtreBvEFjZ6rfab4h8LeHNQ8PXXifT9d1u0iS2vrO2MQufKSOV5t6GZBsaNS+QY96spbsvAPx1t/GPia08O6t4Q8TeA9Zv9OOq6baeJoLZDf26sqymM288wV4zJFvhlKSKJFOwgMVAIf+Ek+N/wD0Tz4f/wDheX3/AMpqP+Ek+N//AETz4f8A/heX3/ymqPxr+0NZeE/GWseHNP8ACHirxleaHa2l7rLeGrWC4/s6K5aRYS0TzpNMzCGVttvHKwC8jJANXxh+01ovhDWfEtuvhnxNruieFTGniTxJpNrDJYaOzIsjrJvmWaYxQsksgt4piiuM/NlQAXv+Ek+N/wD0Tz4f/wDheX3/AMpqP+Ek+N//AETz4f8A/heX3/ymrkPiN+1pYeHW+J1jonh7Xr5/Aumi/wBT8RQ2lpc6ZaiWx+2QOI2voJbgMp27Y8HcPmZFZXOlqX7VeiaPHqc6eGvE2v6H4fgtpfEniXSbGE2GkNLEsziRXuBNIYonSaVbdJzGrjJLZAAN3/hJPjf/ANE8+H//AIXl9/8AKaj/AIST43/9E8+H/wD4Xl9/8pqr2f7R2maz40u9F0Twn4n8RaVY6zFoF74m0i3t7jT7W8kiilCuon+0eWqzw7plhMS7slwFYj16gDyn/hJPjf8A9E8+H/8A4Xl9/wDKan+CPiR4wv8A4kXHg/xh4V0PQrldJGrQXOh6/NqSOvn+UUYS2VuVOecjcCPSvU68q/5um/7kz/2+oA9VooooAK8p/ay/5NY+Mn/Ymaz/AOkM1erV5T+1kB/wyz8ZO/8AxRusn/yRmoA4741ftxfDT4O+ILbwpDfyeMvH99dJY2fhTw4VnunuXYIkcjlhHCSzKCrMGw2QrV79atO1tC1zHHHclFMqROXVWx8wViASAc4JAJHYdK+LvjZ/wSz+GfjXxFbeLfANtZ+BvE9ndR3g0+WyW80O9dGD+XPZN8oRsBSsZVcFso2a98tf2U/gxJawNc/Bf4ewzsgMscfhqykVGxyoYwqWAOQDgZ64HSgD16ivKv8Ahk74If8ARG/h/wD+EvY//GqP+GTvgh/0Rv4f/wDhL2P/AMaoA9Voryr/AIZO+CH/AERv4f8A/hL2P/xqj/hk74If9Eb+H/8A4S9j/wDGqAPVaK8q/wCGTvgh/wBEb+H/AP4S9j/8ao/4ZO+CH/RG/h//AOEvY/8AxqgD1WvKv2lv+SdaR/2OfhP/ANSHTqP+GTvgh/0Rv4f/APhL2P8A8aqfSf2Zfg/oOq2ep6Z8KPBGnalZTJc215aeHLOKaCVGDJIjrGCrKQCGBBBAIoA9OooooAKKKKACiiigAooooA+bPit+z5rPxP8Ajb4o1tAdIi/4RLSY/D3iJHjZ7HWrS/vrhHEeS+FE0W7ICvHK6ZOWA81+Gnwh+J/w98N/DPxRr/w4XxbrOmW3iyy1XwfZ6lZF7WXU9Sa7huIpJ5VheNo08l8uXCzrhGIcL9u0UAfOWi/BnxP4T/Y38OfD290HQfHWq6dptrb6n4c1Rt1tfwK6vNaQSsyqkqplYZH+UPHGW2gll88b4DfETWdP16Tw9pGveGPD+jXWg6t4Y8H+PPE41SSbUNPu1uJT5y3F39nt5YlSFUErYdd+1ABu7HwL+1v4p134b+BfiR4j+HumaL4B8U3dpafbtL8Ry395pxupfIgkngeyhQRGZo0ZllYr5ikKwzi7ZftVeIm1aK7vPAumQ+DpfHk/gJNQt/ETy6j9oW9ks0uDaNZonltIisyrOzKjFsNtIIB558UvCvxd+O3irxHcav8AC3UfDml2fg/xLo3hhm1DTnnubq/s4VVbwR3cixLmJkVkZgzMC5iAAb6G+CviDxDP4b0nQNd+HniLwi+l6Zbwte6tdaZLBPJGiIVj+y3kz54LAsqjA6g4Fen0UAFFFFAHyL+3xpdrqi+BluofNCm+I5II/wCPfuOa8j8F/EX4h3Men+H/AA5eTTpawx28FvDZwv5UShUXczIcKPlBZjj1Ne0ft2f8yR/2/f8AtvXh/wANGS+0fxVodtLFa65qlrFFYzyzLD5gWVTJbqxI+aUEKF6NtwcCvBrSl9acVJr09D8RzirWjn9anTqyinb4W03amml6t6LzZ1WpeLvjHpOt2GkXMs6ahfZ+zRra2rrJj72GVCvyjlsn5RycDmm634z+L/h29jtNRumtppLeW6QGG0ZWjiUtIQyqVJVVJKg7sY45Gc7RvDeueD9a8Pabd3WmvqbPqBi0S7aKZEZ7dVVJSHA/f/KqqWwODyWZaseJ7iX4Xax4Yv7fSpNAuILy4um0Nr+K6CxskKMyuFLKsqq6/MWwUJXFPmmouUpSVmuvp/XR+XbF1MRGlOrVrVoKMop3m3yp8t76Wb1dleL0Xuv3rZkfxq+Icunz3yayDawSRwyP9lthtZwxUYKZOQjHgcbee1el/s6/Hbxn4i+Juj+HdV1OO/0q7SdWia2iQoVieUMrIoOdy45JGGbjOCPNviwbLwzYWHhTSJLeTT/Ol1aWS2uPOVzK7C3BJ5UrAE6HDb888Mbf7LP/ACXjwx/29f8ApLNWaqVI1owc29VfVmGGxmOw+aUMNLEzl70FL3m1dtXXok7eqZ976tpNlr+l3umanZ2+o6beQvbXVndxLLDPE4KvG6MCrKykgqQQQSDX5beHf+Cdj/CP/goh4U2eB18VfA7VjqF1Aby0/tOysR9gnxa3nmIwQpOyCJpc7wYyHaRXC/qvRX0J+9mD4R8D+Hfh/o6aR4W8P6X4a0lXaVbDR7KO1gDt95gkaqoJ7nGTW9RRQB5V+0t/yTrSP+xz8J/+pDp1eq15V+0t/wAk60j/ALHPwn/6kOnV6rQAUUUUAFFFFABRRRQAUUUUAFFFFAHlXx9/5pz/ANjnpv8A7Ur1WvKvj7/zTn/sc9N/9qV6rQAV5h8Dfhpqnwyt/HEeqT2k7a54u1PXrb7G7sEguJA0avuVcOAPmAyAejGvT6KAPkjxR+yJd2/xG8faxpXw/wDhT46tPFmoJq0Op+PrFpb3Rrh0WO4QILaQ3cJMayonnW5DO65wQ9W/Ev7LPi2+8b+PdV0q48O2tvf6j4d8SaFI0k0axahpMcCJazWyxEJbSiJl3pKzRhhhGxivqyigDwfxB8Pvif8AErXPAmq+J4/CWiw+G/GEGtJpmk3l1dstqmn3dvJm6khjEsrS3KEJ5MSoqNl3JAHW/CH4b6n4A1z4nXmoz2s0XifxXJrtmLV2Zo4GsbO3CyZVQH3WznALDBXnJIHpdFAHmevfDXU9U/aD8IeOoZ7RdI0fw/qmlXELu4uGluZrN4yoClSoFs+SWBBK4BySPLvgp+zL4o+HHxU0zxLqd/pM9hay+KneO0mlaUjU9St7q3wGjUfKkLB+eGKgbhlh6j8VP2gPB/whkls9avbmTVU02bVTZ2On3V4YLWPgz3LQRSC1gLZHnTbU+VzuOxsc1pv7XHgqPwB4R8Qa+NS02+1zwvaeLLjS9K0m/wBZbTrSeIPvne1t38uMESKJJAgby3IHysAAeTX37Dmt+LfhH8LvAmva1plrB4d8Eat4Z1K8sjJMRdXP2JoJoUZFEsavaMzK5QkbQOpI9D+BfwL1TwN4ssdW1H4YfCDwQ9rpxt5tQ8E2Je/vbhgod1kNtbi2iO1iYsTFtwG8bctet/2ufC118XpvCqpMvhqLwZH4z/4Sxre5Fm1s+5w28wiNYhCm/wA4vtLMYxh0Za7DwH+0B4J+JXid/DujXupxa4NPGrLp+s6Ff6VLLZlxH58a3cEXmx7mVSyZALDPUUAeYftK/s++KPjB4gubnS/D/gqW8Syii0LxnJqF7pHiHw7cATBpEmto3a6jUyiRYfMgXcGVg+4MOV1D9jnUNH8Y+Mb+z8EfCz4kr4muodSGvfEKwMl/pl0Ykiuv3S20n2qFjGJljE1vtaSRc4w9fYVFAHzV4q/Zp8Rax4X/AGiNKsrrRLc/ELTbay0ZU8yGG2aPSo7M+agRvKTehIVDJhcd+K4tv2Mb/wAO+IvFE9h4A+E/j2LxFJbXy6r46smmu9HuvJSG5RUFs5u4CYllRPOtyGd1JAIavsmigD5U8efsz+LfFHxGs9a03RfA3hrU7HUrSSz+IHhu5vdJ1iHT4xb+ZZz2caNHdhkgaEebc+WI3XEQKkN9V0UUAFeVf83Tf9yZ/wC31eq15V/zdN/3Jn/t9QB6rRRRQAV5T+1l/wAmsfGT/sTNZ/8ASGavVq8p/ay/5NY+Mn/Ymaz/AOkM1AHq1Fc1D8SvCNx/qvFWiSf7mown+TVdh8X6Fcf6rW9Ok/3buM/yagDYoqrDqdncf6q6gk/3JFP8jVnryKAFooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKAPkf4c/s5/E+H4JfD34P+LbfwjY+E/D9zYT6rquk6zdXtzqcVpcLcrbrbyWUCxCSWOIM/msVVWAUlgVyrP9kLxXb+KNUubbwz4B0fUrvx5deJ4/iNY303/CQwWUuovdG3VBYrlmhY27K1y0e13BVhwfs2igAooooAKKKKAPKPjj8DYPjRZ6Qr6vJpF1pryFJBAJkZJAu5Su5ecohB3YxuBByCPJ/wDhhQ/9Dv8A+Un/AO319X0VzTw1KpLmktT57F8P5bjqzxGIpXm93eS206NHyh/wwof+h3/8pP8A9vo/4YUP/Q7/APlJ/wDt9fV9FZ/U6H8v4s4/9VMm/wCfH/k0/wD5I+UP+GFD/wBDv/5Sf/t9db8J/wBlWD4Z+NrLxHL4kk1WSzSURW6WYgG90KbmYu2QFZuAAc4OcAg/QVFVHC0YtSUdfmbUeGsqw9WNalRtKLuvek9V6uwUUUV1n04UUUUAeVftLf8AJOtI/wCxz8J/+pDp1eq15V+0t/yTrSP+xz8J/wDqQ6dXqtABRRRQAUUUUAFFFFABRRRQAUUUUAeVfH3/AJpz/wBjnpv/ALUr1WvOPjZ4R8ReLNH8PyeFoNLu9V0fXLXVltdYvZLOCZYt+5DLHDMyk7uD5bDiqH/CSfG//onnw/8A/C8vv/lNQB6tRXlP/CSfG/8A6J58P/8AwvL7/wCU1H/CSfG//onnw/8A/C8vv/lNQB6tRXlP/CSfG/8A6J58P/8AwvL7/wCU1H/CSfG//onnw/8A/C8vv/lNQB6tRXlP/CSfG/8A6J58P/8AwvL7/wCU1H/CSfG//onnw/8A/C8vv/lNQBwnxC8H+OPCHxN+JeueHPBs3jyx8e+HLXTofs+oWtudLvbVLiNUuBcSJm1kW6D7ofMdSkv7o7lJ8uh+EXxkt/Afg/wZqWheJbjQbDwJoum2em+FvElppFrb6zHZPb3TatcRzJdyRIzREC2aWJliyYpGAz9Gf8JJ8b/+iefD/wD8Ly+/+U1H/CSfG/8A6J58P/8AwvL7/wCU1AHzh4c+Efxp8E6b4dvfD3hP7D4pt/g1Y+EILm5vLGWHTdYs3kdTIDKwbd8pjZUlQOU8zChiOv8Agv8ADHx1pf7S2meNdU8MeMbfRJfDGpaVf6l408T2uoX322a7tLkOLa3ne3t4GEbIi2qoMod0UYVC3sH/AAknxv8A+iefD/8A8Ly+/wDlNR/wknxv/wCiefD/AP8AC8vv/lNQB6tRXlP/AAknxv8A+iefD/8A8Ly+/wDlNR/wknxv/wCiefD/AP8AC8vv/lNQB6tRXlP/AAknxv8A+iefD/8A8Ly+/wDlNR/wknxv/wCiefD/AP8AC8vv/lNQB6tRXlP/AAknxv8A+iefD/8A8Ly+/wDlNR/wknxv/wCiefD/AP8AC8vv/lNQB6tXlX/N03/cmf8At9Sf8JJ8b/8Aonnw/wD/AAvL7/5TVB4J8M+Pr74uXPjDxho/hvQrZdDGkwW+h65cam7t9o80uxlsrcKMcYG4k+lAHrtFFFABXlP7WX/JrHxk/wCxM1n/ANIZq9Wryn9rL/k1j4yf9iZrP/pDNQB0c3wZ+H9x/rfAvhqT/f0i3P8ANKpTfs+/C24/1vw18IS5/v6Dan+cdegUUAeZTfsw/By4/wBb8JvA0n+/4bs2/nHWV+yrpNloPwnu9M0yzt9O0yz8WeKLa1s7SJYoYIk8QagqRoigBVVQAAAAAABXsVeVfs0/8k61f/sc/Fn/AKkOo0Aeq0UUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAeVftLf8k60j/sc/Cf/AKkOnV6rWN4o8K6J440G60TxHo2n+INFutvn6bqlqlzby7WDrvjcFTtZVYZHBUEcgVwf/DJ3wQ/6I38P/wDwl7H/AONUAeq0V5V/wyd8EP8Aojfw/wD/AAl7H/41R/wyd8EP+iN/D/8A8Jex/wDjVAHqtFeVf8MnfBD/AKI38P8A/wAJex/+NUf8MnfBD/ojfw//APCXsf8A41QB6rRXlX/DJ3wQ/wCiN/D/AP8ACXsf/jVH/DJ3wQ/6I38P/wDwl7H/AONUAeq0V5V/wyd8EP8Aojfw/wD/AAl7H/41R/wyd8EP+iN/D/8A8Jex/wDjVAHqtFeVf8MnfBD/AKI38P8A/wAJex/+NUf8MnfBD/ojfw//APCXsf8A41QB6rRXlX/DJ3wQ/wCiN/D/AP8ACXsf/jVH/DJ3wQ/6I38P/wDwl7H/AONUAeq0V5V/wyd8EP8Aojfw/wD/AAl7H/41R/wyd8EP+iN/D/8A8Jex/wDjVAHqtFeVf8MnfBD/AKI38P8A/wAJex/+NUf8MnfBD/ojfw//APCXsf8A41QB6rRXlX/DJ3wQ/wCiN/D/AP8ACXsf/jVH/DJ3wQ/6I38P/wDwl7H/AONUAeq0V5V/wyd8EP8Aojfw/wD/AAl7H/41R/wyd8EP+iN/D/8A8Jex/wDjVAHqtFeVf8MnfBD/AKI38P8A/wAJex/+NUf8MnfBD/ojfw//APCXsf8A41QB6rRXlX/DJ3wQ/wCiN/D/AP8ACXsf/jVH/DJ3wQ/6I38P/wDwl7H/AONUAeq0V5V/wyd8EP8Aojfw/wD/AAl7H/41R/wyd8EP+iN/D/8A8Jex/wDjVAHqtFeVf8MnfBD/AKI38P8A/wAJex/+NUf8MnfBD/ojfw//APCXsf8A41QB6rRXlX/DJ3wQ/wCiN/D/AP8ACXsf/jVH/DJ3wQ/6I38P/wDwl7H/AONUAeq0V5V/wyd8EP8Aojfw/wD/AAl7H/41R/wyd8EP+iN/D/8A8Jex/wDjVAHqteU/tZf8msfGT/sTNZ/9IZqX/hk74If9Eb+H/wD4S9j/APGqP+GTvgh/0Rv4f/8AhL2P/wAaoA9VooooAK8q/Zp/5J1q/wD2Ofiz/wBSHUa9Vryr9mn/AJJ1q/8A2Ofiz/1IdRoA9VooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACvKv2af+Sdav/wBjn4s/9SHUa9Vryr9mn/knWr/9jn4s/wDUh1GgD1WiiigAooooAKKKKACiiigAooooAKKKKAOI+KPxPsvhvptov2WTWPEGqS/ZtH0O1YCe/nwOATwka5BeRvlQHnJKg8Dp/wACte+JlsL/AOLviK61F7hfn8J6HcyWmkQrncquFIeZlba25m4I25YKCWfAu2T4qeLNb+MGoolwl3JLpXhgMnNtp0Mjo0gHBDyybydw3KAQG2tivZ9T1ix0WBZ7+9t7GFm2rJcSrGpbBOMsQM4BP4GolKME5SdkioxcnyxV2eSN+xn8HJI1Q+DEAXoRqF2D+J83J/Gq1x4N8c/A2Q6p4U1bVPHvhCFWe88K6tL9ov4l4G6xnI3HYqriBuGCsAS7gj2mO+t5bNbtJ43tmjEgmVwUKEZ3bumMc59Kh0zWLDWoHn06+t7+FG2GS2lWRQ2AcEqSM4I49xS9pC6jdXewcsrN20RU8G+L9J8feGNP8QaLci80u/iEsMo4OMkFSOzKQVIPIKkHpW7Xh03w18f/AA9+IGv6r8N28PSeHfEEkeoahpfiC4uVZL3LCZ7cRIVQSLsLFtx3DoFUCrFp+0TL4a1C3sPid4Svfh095KI7XUprhL7S5CxwiNdRgLHIdsjbWACqm4sAa0JPaaKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAr5p+Cf7Qnwt8EeGPEOieI/iT4R0DWrXxn4pM+nanrtrb3MW7X79l3xvIGXcrKwyOQwI4Ir6WooA8p/4av+Cfb4veBj9PEVof5SUf8NXfBb/orPgs/TXrU/yevVqKAPKT+1b8Ge3xU8IH6azbn+T0n/DVnwb7fE/wqfpqsJ/k1er0UAeU/wDDVfwe/wCileGj/u6jGf5Gk/4aq+EP/RRfD5+l4p/lXq9FAHwhf/8ABUjwn8MvjVrHgrx2Le+8MmVZdH8ZeGWNzC1s4yq3UAy6uhDKzR53EZEajk/afhDxdpHj7wzpviDQNQi1TRdShW5tLyDO2WNhkMMgEfQgEHgivIrX9jT4dah8YtZ+KPi3Tv8AhOPGWoTq8E2uKstrp0SKEiit7c/Iu1VX52DNuBYFSxFe80AFFFFABXzyuteK/wBpq61e00K/m8H/AAuhlayOt26hr/XGVmSUW7HKxwdR5gDElQOcuqdl+094juPCvwC8a31pH5k7WBtflyCgnZYGcYIIKiQsP92u58H+F7PwX4V0jQLDd9i021jtImYDcyooUM2AAWOMk45JJoAk8K+HbTwh4a0rQrDebLTbWKzhMpBcpGoVSxAAJwBk4GTXlvxwhttS8ZeAdOn2yxy3pWaDPJRpIRyAcgEBgD7HHSvaa8V+J2nwaJ8UvBGrWSm3v9QvRFczB2PmKGhjAwTgfKzA4AznmvleJk5Zc4291yhf0547LrrbR2Vup72R6Y1O+qjO3ryvr0666nrqWNvFYpaJDGlqsYiEIA2BAMBcemOMV5F8FYrbTvH3j6xjMdsEu8Q2owD5ayygFR12qCo44G4eor2qvFPhxaReIvi54y1XUC1xf6ZdmGzbdtWNN0seMDAOFVRz6k8nmpzWKjj8A6aXNzyXZW5HfXyWy6vsPASvhMWpvTlT+fMraeu73XS57XWP4o8M6X4z0C+0TW7GLUdLvozFPbzDKsp7gjkEHBDAgggEEEA1sV4f8dPj9F8KfF2gab/a2l2Nv5Sajqkd9C8kz2rXlvb7YQrqd2yS6l+VXP8Aox+XBNfWHgG/8BdB8QeCNB1fwhrMc9xYaBfvbaJqk0m83ensqyQqzEgl4w/lthFQbFVd20mvUa8RvfiV4yPxiudE0m0bVtLt9ct9NuLMaTOsdvaPYw3El01/nyldWkIELZZgVAALKa9G+GfiC68WfDfwprd95f23U9Ktb2fyhhPMkhR22jJwMscDPSgDqKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKAOf8eeFYvHHgnXvDs0vkR6pYzWZm8sP5ZkRlDhSQCVJDAZHIHIri/2e/G1z4j8DQ6JrMUlj4u8MiPSdasriXzJhLGoCzk9WSZQHVuQcsAzbcn1SvM/iL8F4/FmtReJvDus3HgzxvBGIE1yyiWUTQ7gTFcQMQsycZAbBDKpzhcEA9MrlPGngDSPH1jFb6pHIHhJaGeBtroTjOCQRzgA5B/PFedtH+0XGxVH+GMyA4Ekg1FWYdiQOAT3A4pf+Mj/+qW/+VKufEYeliqbo14qUXunsa0qtShNVKUrSWzR6va6FBZeHYdGjeU20VotmrkjfsCbQScYzgdcde1ZPgf4c6T8P7e5j0vz2kuCpmmuJNzvtztGAABjcegHXnNef/wDGR3p8Lvz1KjP7R3934XfnqNZ/U6DqQquC5oK0X2T3sX9YrKE6ak7S1a727nttcT460vwfomh+Lte8UOtrpeoaatlrFxcXMoRrVRIqxqob5SfPkAEYDM0nGWIrh/8AjI49vhcPx1Ki3+BWt/EHVLHU/i54gt/EsVpKt1beF9KtzBpEEwBBLhiZLkcKR5mAMuCGVyK7DnPSvBNnog0VNW0AyzWWtJBqH2iaWZ5LgG3ijjkYyktuMUUQO7BJGT8xJOnoWh2XhvRdP0nTofs2n6fbx2ttDvZtkUahUXLEk4UAZJJOOSa0aKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigD//Z)

*Figure 5: Metric Diagram*

The system architecture depicted in Fig. 5 illustrates the Node.js server utilized for fetching metrics from a deployed microservice application in a Kubernetes cluster. The Kubernetes cluster is configured with the Istio service mesh. The deployed applications under Istio are subsequently monitored and visualized by the core metric server proposed as centralized monitoring tool. The metrics such as CPU usage, memory usage, and network are obtained by using a PromQL query to identify CPU and memory utilization at the Kubernetes pod, and node levels within the cluster.

Moreover, Kiali enables the visualization of dependency mappings between the microservices, presenting request and response times, thereby facilitating the identification of any breakdowns occurring between microservices. The Node.js server retrieves all relevant endpoints exposed through the Istio service mesh. Upon fetching the required metrics using a scheduled cron job, they are stored in a NoSQL database along with corresponding timestamps. These time series data, along with the timestamps, are then utilized by the optimization server to generate an optimal solution. Subsequently, The Node.js server generates a CSV file from the retrieved NoSQL data, which aids in optimizing the deployment strategy of the Kubernetes cluster. Based on the solution provided by the optimization server, the deployment YAML file is updated and deployed on the cluster

**4.2.3 Optimization Server**

*******Figure 6: Optimization Diagram*

The objective of the optimization server is to predict the future workload on microservice deployment by using the historical data that was continuously fetched through the NodeJS server to create an optimal solution to microservice deployment and create an autoscaling [21] policy through the predicted resource allocation.

The workload prediction is based on the resource utilization matrices that are fetched from the NodeJS server. Gathered resource utilization matrices perform Time series-based prediction on pod utilization metrics such CPU utilization and memory utilization and cluster level metrics, values are taken for a particular period of forecasting. The NodeJS server exposes an API that the optimal server can use to query the data. The data is applied for data-preprocessing steps and train the data for create the prediction module to generate an optimal solution.

Various time series mechanisms are employed for predicting future data. However, in this research, the Bi-LSTM [20] model was selected due to its capability to provide accurate predictions even with a limited amount of data. While ARIMA is a well-known time series method, it was utilized in this study to compare its accuracy with the Bi-LSTM model in predicting CPU and memory usage. The goal was to develop an autoscaling approach through vertical and horizontal auto-scaling methods.

Convolutional 1D (Conv1D) constitutes a foundational element within Convolutional Neural Networks (CNNs), custom-tailored to analyze one-dimensional sequences, such as temporal data. Conv1D operates by sliding a series of adaptable filters across the input sequence, adeptly capturing intricate local patterns and distinctive features embedded within the data. Conversely, the Bidirectional Long Short-Term Memory (Bi-LSTM) stands as a derivative of the Long Short-Term Memory (LSTM) architecture within the realm of recurrent neural networks (RNNs). Bi-LSTM innovatively extends this capability by concurrently processing input sequences in both forward and backward directions, enabling holistic insight into past and future contextual information at every timestep.

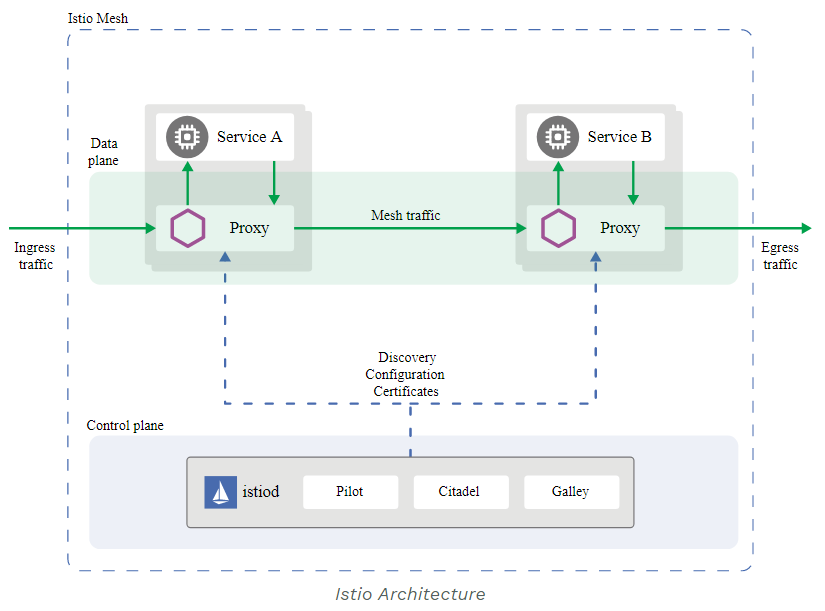
A diagram of a system

Description automatically generated

*Figure 7: Bi-Lstm-CNN Prediction Layers Diagram*

The prediction process relies on a hybrid deep learning architecture that combines Convolutional 1D (Conv1D) layers with Bidirectional Long Short-Term Memory (Bi-LSTM) networks. This combination aims to accurately predict future data patterns. By integrating Conv1D layers with Bi-LSTM networks, our objective was to benefit from the Conv1D layers' ability to identify local patterns while also capturing extensive temporal dependencies through the Bi-LSTM networks. This hybrid approach was carefully selected to enhance the accuracy of forecasting [25] CPU and memory usage.

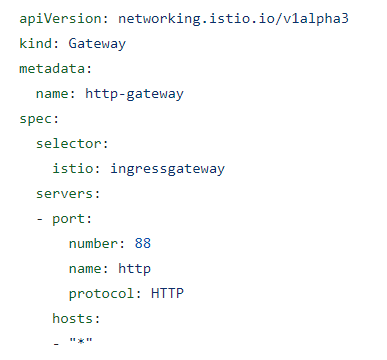
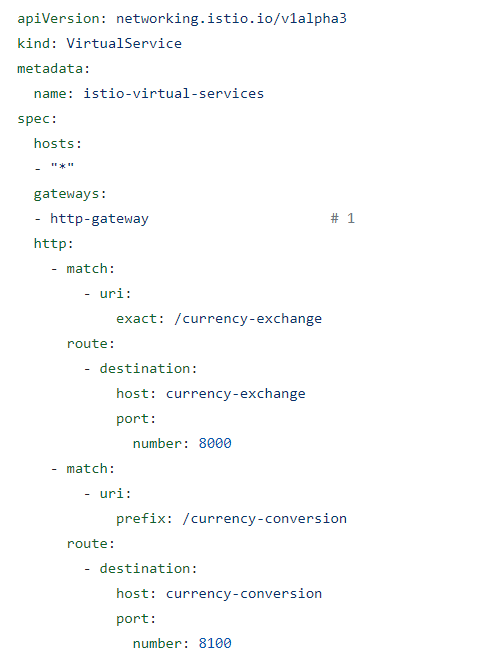
**4.2.4 Istio Service Mesh**

Istio is an open-source service mesh platform designed to simplify the management of microservices in a containerized environment, such as Kubernetes. It provides a set of tools and services for managing traffic, securing communication, and monitoring microservices.

*Figure 8: Istio Architecture Diagram*

A service mesh is a dedicated infrastructure layer for managing communication between microservices. Istio is one such service mesh that intercepts and controls network communication between microservices. It acts as an intermediary between services, handling tasks like load balancing, security, and observability. Istio uses a sidecar proxy, often based on Envoy, alongside each microservice instance. This sidecar proxy is responsible for handling incoming and outgoing network traffic for the microservice. It provides features like load balancing, traffic routing, and encryption without requiring the microservice itself to be aware of these concerns. Istio allows you to define traffic routing rules, such as canary deployments, A/B testing, and blue-green deployments. These rules are configured in a central control plane and are enforced by the sidecar proxies. This enables you to easily manage and control the flow of traffic between microservices. Istio provides robust security features, including mutual TLS (mTLS) authentication between microservices, role-based access control (RBAC), and rate limiting. These features help protect your microservices from unauthorized access and potential threats. Prometheus is an open-source monitoring and alerting toolkit. Istio integrates with Prometheus to collect and store metrics related to the performance and behavior of your microservices. Prometheus scrapes data from the sidecar proxies and other relevant components in the Istio service mesh and stores this data in a time-series database.

In this research first we deployed the microservices to GKE through istio , the Prometheus monitoring tools also deployed under istio. So the metrices of pods that are deployed under istio can bre fetched.



*Figure 9: Gateway and Virtual Service Yaml files*

The following technologies are used to develop the proposed system.

|  |  |
| --- | --- |
| Programming Language | Java  Go  Node js  Python |
| Tools and Technologies | Docker  Kubernetes  Istio  JMeter  TensorFlow  Prometheus  Grafana  Azure  Git  GitHub |

*Table 2: tools and technologies*

## Project Requirements

### Functional Requirements

* + - * Gather time series matrices and logs, structure the data and store it in a persistence database.
      * Expose an API to fetch data that includes all necessary details for creating a machine learning model.
      * Develop a layer to deploy the optimal result to the cluster.
      * A script to start the Gatling and execute the performance tests.
      * The system should be able to display the cluster information through a monitor server.
      * There must be a change in the performance when there was a attack for cluster.

### Non-Functional Requirements

The following are the non-functional requirements that are prioritized for the proposed component as follows.

Availability

Scalability

Performance

Reliability

Maintainability

Usability

### Testing

To make the process and functionalities of the research component follow up in the correct way and to reduce the bugs or errors that can occur during development, create a sample development workflow with configurations and deployment, so if.

there any problems occur, it’s easy to identify and fix, before developing the original workflow.

## Feasibility Study

### Technical Feasibility

The research component involves various tools and technologies that can be used to monitor, scale and make predictions, so technical feasibility was considered at the requirement analysis phase of the research.

### Knowledge in Kubernetes

Microservices applications are deployed in Kubernetes cluster, so some concepts like deployment, services, nodes, pods, etcd, persistence volumes, persistence volume claim and horizontal and vertical auto scaling that covered in this component, with additional advanced concepts .so each member of the group need to understand the basic of Kubernetes to create a deployment. The overall research will mostly use Kubernetes with each member’s components.

### Knowledge in microservices

The overall tools and technologies are mainly come around with a microservice architecture deployment and monitoring. so, a standard microservice application will be deployed to the Kubernetes cluster, so develop a microservice based application.

Knowledge is necessary.

### Knowledge in machine learning

The main goal of this component to predict the resource load and create optimal result that can be assigned to the limit of the vertical auto sculling through gathered time series data.to create a model to optimize the result knowledge of machine.

Learning is important, and in this component deep reinforcement learning algorithm also used to get more optimized solution.

### Knowledge on Monitoring tools

In the component time series matrices are gathered from tools like Prometheus, Grafana using an API call. And, to monitor the performance of the Kubernetes cluster we use Datadog, so the idea to make the relevant API calls and configurations an understanding is important.

### Scheduled feasibility

The proposed research project needs to be completed within a time frame, so the group members following a grant chart to follow up the status of the research, where supervisors also can follow up the status of the completion.

**4.5 Timeline**

Timeline

Description automatically generated

*Figure 10: Gantt Chart*

# 4.6 COMMERCIALIZATION

# The commercialization potential for this research is high as it provides an efficient and automated way to automate and optimize release process. The solution could be marketed to companies and organizations that are using Kubernetes clusters for their applications and services. The automated release process and optimal autoscaling capabilities would allow them to reduce the time and resources required to manage their clusters and improve the performance of their applications.

# Following are the advantage offered by the proposed system:

# 1. The automated release process can help organizations save costs by reducing

# the manual effort required for releases, minimizing the risk of errors, and

# reducing the need for additional resources.

# 2. By automating the release process, organizations can reduce the time required

# for each release, which can lead to faster time-to-market for their products.

# 3. The optimized architectural attributes of the software release process can help

# ensure that the releases are of high quality and meet the desired performance,

# security, extendibility, reliability, and modifiability requirements.

# 4. By adopting an automated and optimized release process, organizations can

# differentiate themselves from their competitors by offering faster and higher.

# quality software releases.

# 5. As more organizations adopt the automated and optimized release process, it

# can become an industry standard, which can further enhance the

# commercialization of this solution.

# Furthermore, the automated and optimized release process can help organizations save cost by reducing manual effort for the releases. The optimal architecture can be used to ensure the performance of the cluster using popular monitoring tools and the multi-cloud support would make it attractive to businesses that operate across multiple cloud providers. Autoscaling could also be extended to other areas, such as network management or resource allocation, which could increase its commercialization potential further.

# RESULTS AND DISCUSION

# 5.1 microservice deployment

# A sample microservice application has been deployed to GKE under the Istio service mesh. The purpose of deploying the microservice is to increase the load and traffic, enabling the monitoring of metrics for the deployed microservice pods through Prometheus. A Kubernetes cluster was created in GKE, and the applications were deployed using the GCloud CLI.

# 

*Figure 11: GKE Cluster*

# 

*Figure 12: GKE Application Deployments*

# 5.2 Prediction model selection

# One of the primary components of the proposed solution involved the development of a prediction model. The aim was to select a model that could provide more accurate predictions, which could be utilized for forecasting future trends. To create the optimal prediction model, various modeling techniques were explored, including time series models such as statistical methods like ARIMA, and deep learning-based recurrent neural network models such as LSTM, Bi-LSTM, and GRU.

# These models were rigorously tested using collected metrics for CPU utilization and memory utilization. The following algorithms were employed in our research environment for comprehensive evaluation:

# LSTM

# ARIMA

# Holt-Winters

# GRU

# Bi-LSTM

# Bi-LSTM-CNN

# To further enhance prediction accuracy while maintaining efficiency, a combination of Convolutional 1D (Conv1D) layers and Bidirectional Long Short-Term Memory (Bi-LSTM) networks was implemented. This combination was specifically chosen due to its ability to achieve a low mean squared error and high accuracy, even when working with limited data.

# In our research environment, Jupyter Notebook was employed as the primary tool for model evaluation. We ensured the installation of relevant dependencies, including TensorFlow, scikit-learn, and Keras. The time series dataset for CPU utilization was collected at 5-minute intervals. Model accuracy and performance were assessed using metrics such as the Mean Absolute Error (MAE), Mean Squared Error (MSE) and Root Mean Squared Error (RMSE). These evaluations allowed us to fine-tune our models and select the most suitable one for our forecasting needs.

# 5.2.1 LSTM Model

# Long Short-Term Memory (LSTM) is a type of recurrent neural network (RNN) architecture that is particularly well-suited for handling sequential data and time series forecasting. LSTMs achieve this by incorporating a memory cell that can store and update information over time, allowing them to remember past observations and make informed predictions about future data points. They can effectively model seasonality, trends, and irregularities in the data, thereby enabling accurate predictions across various time horizons. The following result was obtained for the CPU utilization of the generated dataset.

# 

*Figure 13: LSTM Model Accuracy*

|  |  |  |  |
| --- | --- | --- | --- |
| Model | RMSE | MSE | MAE |
| LSTM | 0.35320 | 0.12475 | 0.18310 |

*Table 3: LSTM Model Accuracy.*

# 5.2.2 ARIMA Model

# The Autoregressive Integrated Moving Average (ARIMA) model is a widely used statistical technique in the field of time series forecasting. The Autoregressive (AR) component considers the relationship between the current observation and its past values, while the Moving Average (MA) component models the influence of past forecast errors on the current observation, helping to capture short-term fluctuations and noise in the data. ARIMA models are particularly useful for time series forecasting because they can handle a wide range of data patterns, including trends, seasonality, and autocorrelation.

# 

*Figure 14: ARIMA Model Accuracy*

|  |  |  |  |
| --- | --- | --- | --- |
| Model | RMSE | MSE | MAE |
| ARIMA | 0.49341 | 0.24345 | 0.46432 |

*Table 4: ARIMA Model Accuracy.*

# 5.2.3 Holt-Winters Model

# The Holt-Winters Model stands out as a widely embraced and potent technique in time series forecasting, particularly adept at handling datasets characterized by trend and seasonality components. The model comprises three essential components: Level (α), which signifies the baseline or average value of the time series data, effectively capturing the overall mean or trend within the dataset; Trend (β), responsible for modeling the direction and rate of change observed over time, thereby capturing any consistent upward or downward movements in the data; and Seasonality (γ), which accounts for recurrent patterns or cycles in the time series, such as daily, weekly, or yearly fluctuations. By estimating and extrapolating these components into the future, the Holt-Winters Model proves invaluable in the realm of time series forecasting.

# 

*Figure 15: HoltWinters Model Accuracy*

|  |  |  |  |
| --- | --- | --- | --- |
| Model | RMSE | MSE | MAE |
| Holt-Winters | 0.34583 | 0.11959 | 0.19297 |

*Table 5: HoltWinters Model Accuracy.*

# 5.2.4 GRU Model

# The Gated Recurrent Unit (GRU) is a type of recurrent neural network (RNN) architecture commonly employed in various machine learning tasks, including time series forecasting. GRUs are composed of specialized gating mechanisms that enable them to selectively update and reset their internal states, enhancing their ability to capture crucial information over extended sequences. They can take a historical sequence of data points as input and learn to discern intricate patterns and dependencies in the data, which can subsequently be utilized for making future predictions.

# 

*Figure 16: GRU Model Accuracy*

|  |  |  |  |
| --- | --- | --- | --- |
| Model | RMSE | MSE | MAE |
| GRU | 0.35113 | 0.12329 | 0.27582 |

*Table 6: GRU Model Accuracy.*

# 5.2.5 Bi-Lstm Model

# The Bidirectional Long Short-Term Memory (Bi-LSTM) model represents a specialized form of recurrent neural network (RNN) architecture that excels in sequence modeling and time series forecasting. In contrast to conventional LSTM models, Bi-LSTM networks handle input sequences in both forward and backward directions. By conducting this bidirectional processing of time series data, the model can glean insights from historical values while also anticipating forthcoming trends. This dual perspective proves invaluable in enhancing the model's capacity to provide precise predictions.

# 

*Figure 17: Bi-Lstm Model Accuracy*

|  |  |  |  |
| --- | --- | --- | --- |
| Model | RMSE | MSE | MAE |
| Bi-Lstm | 0.36486 | 0.1331 | 0.19418 |

*Table 7: Bi-Lstm Model Accuracy.*

# 5.2.6 Bi-Lstm-CNN Model

# The combination of Bidirectional Long Short-Term Memory (Bi-LSTM) and Convolutional Neural Networks (CNN) within a single model represents a potent approach for time series forecasting. Bi-LSTM layers prove highly effective in modeling long-range dependencies inherent in sequential data, as they maintain a memory of both past information and future context. Conversely, CNN layers are well-suited for capturing local patterns or features within time series data. When seamlessly integrated, the Bi-LSTM-CNN model demonstrates its capability to concurrently discern both short-term and long-term patterns within time series data, thus enhancing its forecasting accuracy and robustness.

# 

*Figure 18: Bi-Lstm-CNN Model Accuracy*

|  |  |  |  |
| --- | --- | --- | --- |
| Model | RMSE | MSE | MAE |
| Bi-Lstm-CNN | 0.31872 | 0.10158 | 0.18644 |

*Table 8: Bi-Lstm-CNN Model Accuracy.*

# Performance metrics such as Mean Absolute Error (MAE), Mean Squared Error (MSE), or Root Mean Squared Error (RMSE) are commonly used to assess the accuracy of LSTM models in time series forecasting. Through the evaluation process the bi-lstm model and bi-lstm-cnn model gives a law RMSE , MSE and MAE values. The bi-lstm and the combination model of bi-lstm and cnn was run through several times and tested with several am ount of data.

# 5.3 Data gathering

# Data gathered through the metric server, metric server trigger a cron job every 10min and gathered the metrices of cpu and memory utilization, metrices are gathered through and Prometheus monitoring tool that was deployed to gke(google kubernets engine) through the istio service mesh And the micro services that are monitored through the Prometheus also deployed through the sidecar of the istio service mesh.

# Once the metrices are getheres through the metric server with the following columns , CPU, CPU Usage[%],CPU Usage[KHZ], Memory, Memory usage[Mi],Memory usage[%],Timestamp with more than 2000 data. The data will be stored in the database , and generate and csv file for the prediction server and through and api that was exposed through metric server and request was made for optimization server to make the relevant predictions.

# 5.4 Results and Findings

# According to the test results obtained through various models on the optimization server, the proposed autoscaling policy increases the optimality of the resource cluster. The research demonstrates how the combination of the BiLSTM model and CNN can provide better performance compared to other models for time series forecasting and seasonality analysis. However, it takes more time due to the combined model, and typically, LSTM models take more time with more epochs. The combined model of Bi-LSTM and CNN can provide greater accuracy with both smaller and larger datasets. The loss function of the combined model indicates lower values, which is why it yields more accurate predictions

# 

*Figure 19: Bi-Lstm-CNN Model Loss*

# Conclusion

# The development of the resource prediction system represents a significant step forward in optimizing the software release process and resource management in Kubernetes environments. By leveraging a combination of Convolutional Neural Networks (CNN) and a Bi-LSTM model, this system effectively learns from historical resource usage data per application. The overarching goal of this publication is to introduce a fully automated and highly optimized architecture that not only monitors all relevant metrics through a centralized dashboard but also employs predictive capabilities to allocate resources on-demand for Kubernetes pods.

# The key takeaway from this research is the establishment of a robust prediction mechanism that empowers organizations to scale out their resources effectively. By implementing end-to-end autoscaling policies informed by this predictive model, companies can proactively adapt to fluctuations in demand, ultimately enhancing the efficiency, reliability, and cost-effectiveness of their software deployment processes. This innovative approach offers a promising avenue for achieving better resource allocation and system performance in the ever-evolving landscape of containerized applications.

# Further Improvements

# The integration of reinforcement learning algorithms to enhance resource allocation decisions in Kubernetes environments can be proposed. While existing Convolutional Neural Networks (CNN) and Bi-LSTM models provide valuable insights into historical resource usage, reinforcement learning can introduce dynamic adaptability.

# By incorporating reinforcement learning, the system can continuously learn and optimize resource allocation policies based on real-time feedback and changing workload patterns. This approach would enable the prediction system to not only forecast future resource needs but also actively fine-tune resource allocations in response to evolving application requirements and environmental conditions.

# Furthermore, the system could benefit from a self-learning mechanism that allows it to adapt and refine its predictions over time, reducing the need for manual tuning. This self-improvement aspect of the system would make it more resilient and efficient in managing resources, especially in complex and dynamic Kubernetes environments.

# By combining the power of deep learning with reinforcement learning and self-adaptation, the resource prediction system can achieve even higher levels of automation, efficiency, and cost-effectiveness, making it a valuable asset for organizations seeking to excel in containerized application deployment.

# REFERENCES

# 

# [1] Zhijiao Xiao, Song Hu, "DScaler: A Horizontal Autoscaler of Microservice Based on

# DeepReinforcement Learning " The 23rd Asia-Pacific Network Operations and

# Management Symposium (APNOMS) 2022, IEEE Press, 2020, pp. 460-470.

# [2]  Shubo Zhang , Tianyang Wu, Maolin Pan, Chaomeng Zhang and Yang Yu "A-SARSA: A Predictive Container Auto-Scaling Algorithm Based on Reinforcement Learning " 2020 IEEE International Conference on Web Services (ICWS) , vol. 107, pp. 101-115, 2019.

# [3] Zhang, H.; Jiang, G.; Yoshihira, K.; Chen, H.; Saxena, A. Intelligent Workload Factoring for a Hybrid Cloud Computing Model. In Proceedings of the 2009 Congress on Services—I, Los Angeles, CA, USA, 6–10 July 2009; pp. 701–708. .

# [4] Fang, W.; Lu, Z.; Wu, J.; Cao, Z. RPPS: A Novel Resource Prediction and Provisioning Scheme in Cloud Data Center. In Proceedings of the 2012 IEEE Ninth International Conference on Services Computing, Honolulu, HI, USA, 24–29 June 2012; pp. 609–616.

# [5]  Calheiros, R.N.; Masoumi, E.; Ranjan, R.; Buyya, R. Workload Prediction Using ARIMA Model and Its Impact on Cloud Applications’ QoS. IEEE Trans. Cloud Comput. 2015, 3, 449–458.

# [6] Tang, X.; Liu, Q.; Dong, Y.; Han, J.; Zhang, Z. Fisher: An Efficient Container Load Prediction Model with Deep Neural Network in Clouds. In Proceedings of the 2018 IEEE Intl Conf on Parallel Distributed Processing with Applications, Ubiquitous Computing Communications, Big Data Cloud Computing, Social Computing Networking, Sustainable Computing Communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom), Melbourne, Australia, 11–13 December 2018; pp. 199–206.

# [7]  Yan, M.; Liang, X.; Lu, Z.; Wu, J.; Zhang, W. HANSEL: Adaptive horizontal scaling of microservices using Bi-LSTM. Appl. Soft Comput. 2021, 105, 107216

# [8] Imdoukh, M.; Ahmad, I.; Alfailakawi, M.G. Machine learning-based auto-scaling for containerized applications. Neural Comput. Appl. 2019, 32, 9745–9760.

# [9] Toka, L.; Dobreff, G.; Fodor, B.; Sonkoly, B. Adaptive AI-based auto-scaling for Kubernetes. In Proceedings of the 2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID), Melbourne, Australia, 11–14 May 2020; pp. 599–608.

# [10] Prachitmutita, I.; Aittinonmongkol, W.; Pojjanasuksakul, N.; Supattatham, M.; Padungweang, P. Auto-scaling microservices on IaaS under SLA with cost-effective framework. In Proceedings of the 2018 Tenth International Conference on Advanced Computational Intelligence (ICACI), Xiamen, China, 29–31 March 2018; pp. 583–588.

# [11] H. Arabnejad, C. Pahl, P. Jamshidi, and G. Estrada, “A comparison of reinforcement learning techniques for fuzzy cloud auto-scaling,” in Proc. 17th IEEE/ACM Int. Symp. Clust. Cloud Grid Comput. (CCGRID), 2017, pp. 64–73.

# [12] F. Rossi, M. Nardelli, and V. Cardellini, “Horizontal and vertical scaling of container-based applications using reinforcement learning,” in Proc. 12th IEEE Int. Conf. Cloud Comput. (CLOUD), 2019, pp. 329–338.

[13]Hui Zhang, Princeton, Guofei Jiang; Kenji Yoshihira; Haifeng Chen; Akhilesh Saxena “Intelligent Workload Factoring for a Hybrid Cloud Computing Model”, 2009 Congress on Services – I

[14] Arnaldo Pereira Ferreira, Richard O. Sinnott, “A Performance Evaluation of Containers running on Managed Kubernetes Services” , 2019 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)

[15] Nhat-Minh Dang-Quang and Myungsik Yoo , “Deep Learning-Based

Autoscaling Using Bidirectional Long Short-Term Memory for Kubernetes”, applied science – 11

## 

## 9 APPENDIX

**Metric Server**

**[Appendix 1: Cpu\_usage.model.js]**

import mongoose from "mongoose";

const { Schema } = mongoose;

const CpuUsageSchama = new Schema(

  {

    metrices: {

        type: String,

      },

    timestamp: {

        type: String,

      },

    data: [

        {

          pod: {

            type: String,

          },

          value: {

            type: String,

          },

        },

      ],

  },

  { timestamps: true, versionKey: false },

);

export const CPU = mongoose.model("cpuusage", CpuUsageSchama);

**[Appendix 2: Memory\_usage.model.js]**

import mongoose from "mongoose";

const { Schema } = mongoose;

const MemoryUtilizationSchama = new Schema(

  {

    metrices: {

        type: String,

      },

    timestamp: {

        type: String,

      },

    data: [

        {

          pod: {

            type: String,

          },

          value: {

            type: String,

          },

        },

      ],

  },

  { timestamps: true, versionKey: false },

);

export const MEMORY = mongoose.model("memoryutilization", MemoryUtilizationSchama);

**[Appendix 3: Cpu\_usage.controller.js]**

import express from "express";

import Success from "../utils/success.js";

import axios from "axios";

import fs from "fs";

import path from "path";

import {CPU} from "../models/index.js";

import XLSX from "xlsx";

import ExcelJS from 'exceljs';

const PROMETHEUS\_PORT = "http://localhost:9090/api/v1";

const CPU\_USAGE = `sum (rate (container\_cpu\_usage\_seconds\_total{image!=""}[1m])) by (pod)`;

  export const saveMetricData = async (req, res) => {

    axios

    .get(`${PROMETHEUS\_PORT}/query?query=${CPU\_USAGE}`).then(async (response) => {

        const metricdata = [];

        let temp\_timestamp = 0;

        const data = response.data.data.result;

        await data.forEach(async (timeseries) => {

            let temp\_data = {

              pod: "",

              timestamp: "",

              value: "",

            };

        temp\_data.pod = timeseries.metric.pod;

        temp\_timestamp = timeseries.value[0];

        temp\_data.timestamp = timeseries.value[0];

        temp\_data.value = timeseries.value[1];

        await metricdata.push(temp\_data);

      });

            //Create a Object using Model

            const Cpu\_Usage = new CPU({

                metrices: "cpu\_usage",

                timestamp: temp\_timestamp,

                data: metricdata,

              });

              //Save to Database

              await Cpu\_Usage

                .save()

                .then((cpudata) => {

                    res.json(Success(cpudata, "Successfully cpu usage data saved."));

                })

                .catch((err) => {

                    res.status(err.status).json(err.message);

                });

            })

            .catch((err) => {

                res.status(err.status).json(err.message);

            });

  };

  export const fetchCpuUsage = async (req, res) => {

    CPU.find({})

    .then((cpu) => {

        res.json(Success(cpu, "Successfully fetched all  cpu usage data."));

    })

    .catch((err) => {

        res.status(err.status).json(err.message);

    });

  };

  export const fetchCpuUsageByPod = async (req, res) => {

    let pod\_array = [];

    CPU.find({})

      .then(async (cpu) => {

        await cpu.forEach((cpu\_data) => {

          let pod = {

            timestamp: 0,

            value: 0,

          };

          pod.timestamp = cpu\_data.timestamp;

          let pod\_data = cpu.data.filter(function (pod\_data) {

            return pod\_data.pod == req.params.pod;

          });

          pod\_data.forEach((pod\_value) => {

            pod.value = pod\_value.value;

            pod\_array.push(pod);

          });

        });

        res.json(Success(pod\_array, "Successfully fetched all  cpu usage by pod name."));

      })

      .catch((err) => {

        res.status(err.status).json(err.message);

      });

  };

export const addDataToExcelByPod = async (req, res) => {

    try {

        const dataset = [];

        const file\_name = req.params.pod;

        const filePath = path.join(\_\_dirname, '..', '..', 'optimize\_server', `${file\_name}.xlsx`);

        const workbook = new ExcelJS.Workbook();

        const cpuData = await CPU.find();

        for (const pod\_data of cpuData) {

          const pod = {

            timestamp: 0,

            value: 0,

          };

          pod.timestamp = pod\_data.timestamp;

          const pod\_single = pod\_data.data.filter((data) => {

            const split\_text = data.pod.split("-");

            const PodName = `${split\_text[0]}-${split\_text[1]}`;

            return req.params.pod.includes(PodName);

          });

          const worksheet = workbook.addWorksheet('Sheet 1');

          const headers = Object.keys(pod);

          worksheet.getRow(1).values = headers;

          pod\_single.forEach((data) => {

            pod.value = data.value;

            dataset.push({ ...pod });

            const rowData = Object.values(pod);

            worksheet.addRow(rowData);

          });

        }

        await workbook.xlsx.writeFile(filePath);

        console.log("xlsx Generated");

        res.json(Success(dataset, "Successfully fetched all CPU usage by pod name."));

        await axios

        .get(

          "http://127.0.0.1:8000/prdiction-cpu/prediction\_singlepod?pod\_name=" +

            request.params.podName

        )

        .then(async (res) => {

          let tempArray = [];

          await res.data.res.forEach((singleValue) => {

            tempArray.push(singleValue[0]);

          });

          await response.json(tempArray);

        })

        .catch((error) => {

          response.json(error);

        });

      } catch (err) {

        console.error(err);

        res.status(500).json(Error("Internal server error"));

      }

  };

**[Appendix 4: Memory\_usage.controller.js]**

import express from "express";

import Success from "../utils/success.js";

import axios from "axios";

import fs from "fs";

import path from "path";

import {MEMORY} from "../models/index.js";

import XLSX from "xlsx";

import ExcelJS from 'exceljs';

const PROMETHEUS\_PORT = "http://localhost:9090/api/v1";

const MEM\_USAGE = `sum (rate (container\_memory\_working\_set\_bytes{image!=""}[1m])) by (pod)`;

  export const saveMemoryMetricData = async (req, res) => {

    axios

    .get(`${PROMETHEUS\_PORT}/query?query=${MEM\_USAGE}`).then(async (response) => {

        const metricdata = [];

        let temp\_timestamp = 0;

        const data = response.data.data.result;

        await data.forEach(async (timeseries) => {

            let temp\_data = {

              pod: "",

              timestamp: "",

              value: "",

            };

        temp\_data.pod = timeseries.metric.pod;

        temp\_timestamp = timeseries.value[0];

        temp\_data.timestamp = timeseries.value[0];

        temp\_data.value = timeseries.value[1];

        await metricdata.push(temp\_data);

      });

            //Create a Object using Model

            const Memory\_Usage = new MEMORY({

                metrices: "memory\_utilization",

                timestamp: temp\_timestamp,

                data: metricdata,

              });

              //Save to Database

              await Memory\_Usage

                .save()

                .then((memorydata) => {

                    res.json(Success(memorydata, "Successfully memory usage data saved."));

                })

                .catch((err) => {

                    res.status(err.status).json(err.message);

                });

            })

            .catch((err) => {

                res.status(err.status).json(err.message);

            });

  };

  export const fetchMemoryUsage = async (req, res) => {

    MEMORY.find({})

    .then((memory) => {

        res.json(Success(memory, "Successfully fetched all  memory usage data."));

    })

    .catch((err) => {

        res.status(err.status).json(err.message);

    });

  };

  export const fetchMemoryUsageByPod = async (req, res) => {

    let pod\_array = [];

    MEMORY.find({})

      .then(async (memory) => {

        await cpu.forEach((memory\_data) => {

          let pod = {

            timestamp: 0,

            value: 0,

          };

          pod.timestamp = memory\_data.timestamp;

          let pod\_data = memory.data.filter(function (pod\_data) {

            return pod\_data.pod == req.params.pod;

          });

          pod\_data.forEach((pod\_value) => {

            pod.value = pod\_value.value;

            pod\_array.push(pod);

          });

        });

        res.json(Success(pod\_array, "Successfully fetched all  cpu usage by pod name."));

      })

      .catch((err) => {

        res.status(err.status).json(err.message);

      });

  };

  export const addMemoryDataToExcelByPod = async (req, res) => {

    try {

        const dataset = [];

        const file\_name = req.params.pod;

        const filePath = path.join(\_\_dirname, '..', '..', 'optimize\_server', `${file\_name}.xlsx`);

        const workbook = new ExcelJS.Workbook();

        const memoryData = await MEMORY.find();

        for (const pod\_data of memoryData) {

          const pod = {

            timestamp: 0,

            value: 0,

          };

          pod.timestamp = pod\_data.timestamp;

          const pod\_single = pod\_data.data.filter((data) => {

            const split\_text = data.pod.split("-");

            const PodName = `${split\_text[0]}-${split\_text[1]}`;

            return req.params.pod.includes(PodName);

          });

          const worksheet = workbook.addWorksheet('Sheet 1');

          const headers = Object.keys(pod);

          worksheet.getRow(1).values = headers;

          pod\_single.forEach((data) => {

            pod.value = data.value;

            dataset.push({ ...pod });

            const rowData = Object.values(pod);

            worksheet.addRow(rowData);

          });

        }

        await workbook.xlsx.writeFile(filePath);

        console.log("xlsx Generated");

        res.json(Success(dataset, "Successfully fetched all Memory usage by pod name."));

        await workbook.xlsx.writeFile(filePath);

        console.log("xlsx Generated");

        res.json(Success(dataset, "Successfully fetched all CPU usage by pod name."));

        await axios

        .get(

          "http://127.0.0.1:8000/prdiction-memory/prediction\_singlepod?pod\_name=" +

            request.params.podName

        )

        .then(async (res) => {

          let tempArray = [];

          await res.data.res.forEach((singleValue) => {

            tempArray.push(singleValue[0]);

          });

          await response.json(tempArray);

        })

        .catch((error) => {

          response.json(error);

        });

      } catch (err) {

        console.error(err);

        res.status(500).json(Error("Internal server error"));

      }

  };

**[Appendix 5: Cpu\_usage.route.js]**

import express from "express";

import {

  saveMetricData,

  fetchCpuUsage,

  fetchCpuUsageByPod,

  addDataToExcelByPod,

} from "../controllers/index.js";

const cpuRouter = express.Router();

cpuRouter.get("/", fetchCpuUsage);

cpuRouter.get("/saveCpuMetrices", saveMetricData);

cpuRouter.get("fetchByPod/:pod", fetchCpuUsageByPod);

cpuRouter.get("export/exportCpuDataToExcel/:pod", addDataToExcelByPod);

export default cpuRouter;

**[Appendix 6: Memory\_usage.route.js]**

import express from "express";

import {

 saveMemoryMetricData,

 fetchMemoryUsage,

 fetchMemoryUsageByPod,

 addMemoryDataToExcelByPod,

} from "../controllers/index.js";

const memoryRouter = express.Router();

memoryRouter.get("/", fetchMemoryUsage);

memoryRouter.get("/saveMemoryMetrices", saveMemoryMetricData);

memoryRouter.get("fetchMemoryByPod/:pod", fetchMemoryUsageByPod);

memoryRouter.get("export/exportMemoryDataToExcel/:pod", addMemoryDataToExcelByPod);

export default memoryRouter;

**[Appendix 7: index.route.js]**

import express from "express";

import memoryRouter from "./memory\_usage.route.js";

import cpuRouter from "./cpu\_usage.route.js";

const apiRouter = express.Router();

apiRouter.use("/cpu", cpuRouter);

apiRouter.use("/memory", memoryRouter);

export default apiRouter;

**[Appendix 8: app.js]**

import "dotenv/config";

import express from "express";

import cors from "cors";

import { connect } from "./utils/dbConnect.js";

import apiRouter from "./routes/index.js";

const app = express();

app.use(cors());

app.use(express.json());

// mongodb connection

connect();

app.set("views", "views");

app.set("view engine", "pug");

app.use(express.static("public"));

app.use("/api", apiRouter);

app.get("/", (req, res) => {

  res.render("index");

});

export default app;

**[Appendix 9: server.js]**

import app from "./app.js";

const PORT = process.env.PORT || 5000;

app.listen(PORT, () => {

  // eslint-disable-next-line no-console

  console.log(`Server started on port ${PORT}`);

});

**[Appendix 10: Package.json]**

{

  "name": "plantme-backend",

  "version": "1.0.0",

  "description": "This project is related to the platme application backend.",

  "main": "app.js",

  "type": "module",

  "scripts": {

    "start": "node server.js",

    "dev": "nodemon server.js",

    "test": "jest --setupFiles dotenv/config",

    "lint": "eslint .",

    "lint:fix": "eslint . --fix",

    "format": "prettier --write .",

    "format:check": "prettier --check .",

    "prepare": "husky install",

    "build": "webpack --config webpack.dev.js"

  },

  "jest": {

    "transform": {

      "^.+\\.[t|j]sx?$": "babel-jest"

    }

  },

  "license": "GPL-3.0",

  "dependencies": {

    "all": "^0.0.0",

    "axios": "^1.5.0",

    "bcrypt": "^5.0.1",

    "celebrate": "^15.0.1",

    "cors": "^2.8.5",

    "dotenv": "^16.0.0",

    "exceljs": "^4.3.0",

    "express": "^4.17.3",

    "jsonwebtoken": "^8.5.1",

    "mongoose": "^6.2.9",

    "pug": "^3.0.2",

    "xlsx": "^0.18.5"

  },

  "devDependencies": {

    "@babel/core": "^7.17.9",

    "babel-jest": "^27.5.1",

    "eslint": "^8.12.0",

    "eslint-config-airbnb-base": "^15.0.0",

    "eslint-config-prettier": "^8.5.0",

    "eslint-plugin-import": "^2.25.4",

    "eslint-plugin-prettier": "^4.0.0",

    "husky": "^7.0.0",

    "jest": "^27.5.1",

    "nodemon": "^2.0.15",

    "prettier": "^2.6.2",

    "supertest": "^6.2.2"

  }

}

**Optimization Server**

**[Appendix 11: Server.js]**

from fastapi import FastAPI  
from routes import CPU\_Usage\_Route,MEMORY\_Usage\_Route  
app = FastAPI()  
  
app.include\_router(CPU\_Usage\_Route)  
app.include\_router(MEMORY\_Usage\_Route)  
  
@app.get('/')  
def index():  
 return 'Prediction Server Backend api Up and running'

**[Appendix 12: cpu\_usage\_prediction.py]**

from utils.data\_preprocessing import change\_timestamp\_to\_dateTime, drop\_column, missing\_values, split\_test\_train, \  
 scale\_minmax\_scaler, import\_dataframe\_from\_csv, create\_timeSeries\_generater  
from prediction\_models import BiLstm\_CNN\_model  
  
  
async def make\_predictions\_cpu(pod\_name: str):  
 df = await data\_load\_cpu\_by\_podname(pod\_name)  
 df = await preprocess\_cpu\_utilization\_dataset(df)  
 dataframe = df[["value"]]  
 print(dataframe.head())  
 train, test = split\_test\_train(dataframe, 0.8)  
 scaler\_train, scaler\_test, scaler = scale\_minmax\_scaler(train, test)  
 generator = create\_timeSeries\_generater(scaler\_train)  
 model\_bilstm\_cnn = BiLstm\_CNN\_model(64, generator, 100, 32)  
 model\_bilstm\_cnn = model\_bilstm\_cnn.build\_model()  
 history\_BiLSTM\_CNN, model\_bilstm\_cnn = await model\_bilstm\_cnn.fit\_model(model\_bilstm\_cnn)  
  
 ## Los Plot ####  
  
 prediction\_bilstm\_cnn = model\_bilstm\_cnn.predict(model\_bilstm\_cnn, test, scaler\_train, 12, 1, scaler)  
  
 res = prediction\_bilstm\_cnn.tolist()  
  
 return {"res": res}  
  
 df = import\_dataframe\_from\_csv('cpu\_usage.csv')  
 df = change\_timestamp\_to\_dateTime\_and\_sort(df, 'dateTime', 'timestamp')  
 return df  
  
  
async def data\_load\_cpu\_by\_podname(pod: str):  
 df = import\_dataframe\_from\_csv("CPU/ " + pod + ".csv")  
 df = change\_timestamp\_to\_dateTime(df, 'dateTime', 'timestamp')  
 return df  
  
  
async def preprocess\_cpu\_utilization\_dataset(dataframe):  
 df = missing\_values(dataframe)  
 return df

**[Appendix 13: memory\_usage\_prediction.py]**

from utils.data\_preprocessing import change\_timestamp\_to\_dateTime, drop\_column, missing\_values, split\_test\_train, \  
 scale\_minmax\_scaler, import\_dataframe\_from\_csv, create\_timeSeries\_generater  
from prediction\_models import BiLstm\_CNN\_model  
  
  
async def make\_predictions\_cpu(pod\_name: str):  
 df = await data\_load\_cpu\_by\_podname(pod\_name)  
 df = await preprocess\_cpu\_utilization\_dataset(df)  
 dataframe = df[["value"]]  
 print(dataframe.head())  
 train, test = split\_test\_train(dataframe, 0.8)  
 scaler\_train, scaler\_test, scaler = scale\_minmax\_scaler(train, test)  
 generator = create\_timeSeries\_generater(scaler\_train)  
 model\_bilstm\_cnn = BiLstm\_CNN\_model(64, generator, 100, 32)  
 model\_bilstm\_cnn = model\_bilstm\_cnn.build\_model()  
 history\_BiLSTM\_CNN, model\_bilstm\_cnn = await model\_bilstm\_cnn.fit\_model(model\_bilstm\_cnn)  
  
 ## Los Plot ####  
  
 prediction\_bilstm\_cnn = model\_bilstm\_cnn.predict(model\_bilstm\_cnn, test, scaler\_train, 12, 1, scaler)  
  
 res = prediction\_bilstm\_cnn.tolist()  
  
 return {"res": res}  
  
 df = import\_dataframe\_from\_csv('cpu\_usage.csv')  
 df = change\_timestamp\_to\_dateTime\_and\_sort(df, 'dateTime', 'timestamp')  
 return df  
  
  
async def data\_load\_cpu\_by\_podname(pod: str):  
 df = import\_dataframe\_from\_csv("CPU/ " + pod + ".csv")  
 df = change\_timestamp\_to\_dateTime(df, 'dateTime', 'timestamp')  
 return df  
  
  
async def preprocess\_cpu\_utilization\_dataset(dataframe):  
 df = missing\_values(dataframe)  
 return df

**[Appendix 14: Bi\_Lstm\_Model.py]**

import tensorflow as tf  
from tensorflow import keras  
from keras.models import Sequential  
from keras.layers import Dense, LSTM,Bidirectional  
import numpy as np  
  
  
class Bi\_LSTM\_model():  
 def \_\_init\_\_(self, units, generator, epohs, batch\_size):  
 self.units = units  
 self.generator = generator  
 self.epohs = epohs  
 self.batch\_size = batch\_size  
  
 def build\_model(self):  
 model = Sequential()  
 # Input layer  
 model.add(Bidirectional(  
 LSTM(units=self.units, return\_sequences=True),  
 input\_shape=(12, 1)))  
 # Hidden layer  
 model.add(Bidirectional(LSTM(units=self.units)))  
 model.add(Dense(1))  
 # Compile model  
 model.compile(optimizer='adam', loss='mse')  
 return model  
  
 async def fit\_model(self, model):  
 early\_stop = keras.callbacks.EarlyStopping(monitor='val\_loss',  
 patience=50)  
 history = model.fit(self.generator, epochs=self.epohs,  
 batch\_size=self.batch\_size,  
 callbacks=[early\_stop])  
 return history, model  
  
 def predict(self, model, test, scaler\_train,n\_input,n\_features,scaler):  
 test\_prediction = []  
 first\_eval\_batch = scaler\_train[-12:]  
 current\_batch = first\_eval\_batch.reshape((1, n\_input, n\_features))  
  
 for i in range(len(test)):  
 current\_pred = model.predict(current\_batch)[0]  
 test\_prediction.append(current\_pred)  
 current\_batch = np.append(current\_batch[:, 1:, :], [[current\_pred]], axis=1)  
  
 true\_prediction = scaler.inverse\_transform(test\_prediction)  
 return true\_prediction

**[Appendix 15: Bi\_Lstm\_Cnn\_Model.py]**

import tensorflow as tf  
from tensorflow import keras  
from keras.models import Sequential  
from keras.layers import Dropout  
from keras.layers import Flatten  
from keras.layers import Input, Dense, LSTM, Conv1D, Dropout, Bidirectional, Multiply  
from keras.models import Model  
from keras.optimizers import Adam  
from keras.models import \*  
import numpy as np  
  
  
class Bi\_LSTM\_CNN\_model():  
 def \_\_init\_\_(self, units, generator, epohs, batch\_size):  
 self.units = units  
 self.generator = generator  
 self.epohs = epohs  
 self.batch\_size = batch\_size  
  
 def build\_model(self):  
 inputs = Input(shape=(12, 1))  
 x = Conv1D(filters=128, kernel\_size=1, activation='relu')(inputs)  
 x = Dropout(0.02)(x)  
  
 x = Conv1D(filters=64, kernel\_size=3, activation='relu')(x)  
 x = Dropout(0.02)(x)  
  
 x = Conv1D(filters=32, kernel\_size=3, activation='relu')(x)  
 x = Dropout(0.02)(x)  
 lstm\_out = Bidirectional(LSTM(16, return\_sequences=False, activation='relu'))(x)  
 lstm\_out = Dense(16)(lstm\_out)  
 lstm\_out = Dense(8)(lstm\_out)  
 lstm\_out = Dense(4)(lstm\_out)  
 attention\_mul = Flatten()(lstm\_out)  
 output = Dense(2)(attention\_mul)  
 output = Dense(1)(output)  
 model = Model(inputs=[inputs], outputs=output)  
 model.compile(loss='mean\_squared\_error', optimizer=Adam(learning\_rate=0.001))  
 return model  
  
 async def fit\_model(self, model):  
 early\_stop = keras.callbacks.EarlyStopping(monitor='val\_loss',  
 patience=50)  
 history = model.fit(self.generator, epochs=self.epohs,  
 batch\_size=self.batch\_size,  
 callbacks=[early\_stop])  
 return history, model  
  
 def predict(self, model, test, scaler\_train,n\_input,n\_features,scaler):  
 test\_prediction = []  
 first\_eval\_batch = scaler\_train[-12:]  
 current\_batch = first\_eval\_batch.reshape((1, n\_input, n\_features))  
  
 for i in range(len(test)):  
 current\_pred = model.predict(current\_batch)[0]  
 test\_prediction.append(current\_pred)  
 current\_batch = np.append(current\_batch[:, 1:, :], [[current\_pred]], axis=1)  
  
 true\_prediction = scaler.inverse\_transform(test\_prediction)  
 return true\_prediction

**[Appendix 16: Lstm\_Model.py]**

import tensorflow as tf  
from tensorflow import keras  
from keras.models import Sequential  
from keras.layers import Dense, LSTM,Bidirectional  
import numpy as np  
  
  
class LSTM\_model():  
 def \_\_init\_\_(self, units, generator, epohs, batch\_size):  
 self.units = units  
 self.generator = generator  
 self.epohs = epohs  
 self.batch\_size = batch\_size  
  
 def build\_model(self):  
 model = Sequential()  
 # Input layer  
 model.add(  
 LSTM(units=self.units, return\_sequences=False, activation='relu', input\_shape=(12,1)))  
 # Hidden layer  
 model.add(LSTM(units=self.units))  
 model.add(Dense(1))  
 # Compile model  
 model.compile(optimizer='adam', loss='mse')  
 return model  
  
 async def fit\_model(self, model):  
 early\_stop = keras.callbacks.EarlyStopping(monitor='val\_loss',  
 patience=50)  
 history = model.fit(self.generator, epochs=self.epohs,  
 batch\_size=self.batch\_size,  
 callbacks=[early\_stop])  
 return history, model  
  
 def predict(self, model, test, scaler\_train,n\_input,n\_features,scaler):  
 test\_prediction = []  
 first\_eval\_batch = scaler\_train[-12:]  
 current\_batch = first\_eval\_batch.reshape((1, n\_input, n\_features))  
  
 for i in range(len(test)):  
 current\_pred = model.predict(current\_batch)[0]  
 test\_prediction.append(current\_pred)  
 current\_batch = np.append(current\_batch[:, 1:, :], [[current\_pred]], axis=1)  
  
 true\_prediction = scaler.inverse\_transform(test\_prediction)  
 return true\_prediction

**[Appendix 17: Cpu\_usage\_route.py]**

from fastapi import APIRouter  
from controller import make\_predictions\_cpu  
router = APIRouter(  
 prefix="/prdiction-cpu",  
 tags=['CPU\_prediction']  
)  
  
  
@router.get('/prediction\_singlepod')  
async def make\_predictions\_cpu(pod\_name:str):  
 result = await make\_predictions\_cpu(pod\_name)  
 return result

**[Appendix 18: memory\_usage\_route.py]**

from fastapi import APIRouter  
from controller import make\_pod\_predictions\_memory  
router = APIRouter(  
 prefix="/prdiction-memory",  
 tags=['MEMORY\_prediction']  
)  
  
  
@router.get('/prediction\_singlepod')  
async def make\_pod\_predictions\_memory(pod\_name:str):  
 result = await make\_pod\_predictions\_memory(pod\_name)  
 return result

**[Appendix 19: data\_preprocessing.py]**

from sklearn.preprocessing import MinMaxScaler, StandardScaler  
import numpy as np  
from keras.preprocessing.sequence import TimeseriesGenerator  
import pandas as pd  
from datetime import datetime  
import os  
  
  
def change\_timestamp\_to\_dateTime(dataframe, dateTime\_col\_name, timestamp\_col\_name):  
 dataframe[dateTime\_col\_name] = [datetime.fromtimestamp(x) for x in dataframe[timestamp\_col\_name]]  
 dataframe = dataframe.sort\_values('dateTime')  
 dataframe = dataframe.set\_index('dateTime')  
 return dataframe  
  
  
def drop\_column(dataframe, column\_name):  
 return dataframe.drop(column\_name, axis=1)  
  
  
def missing\_values(dataframe):  
 for col in dataframe.columns:  
 if col == 'timestamp' or col == 'dateTime':  
 continue  
 print(dataframe[col].isna().sum())  
 print('')  
  
 if dataframe[col].isna().sum() > 0:  
 # Locate the missing value  
 df\_missing\_date = dataframe.loc[dataframe[col].isna() == True]  
 # Replcase missing value with interpolation  
 dataframe[col].interpolate(inplace=True)  
 dataframe.fillna(method='pad')  
  
 return dataframe  
  
  
def split\_test\_train(dataframe, ratio):  
 train\_size = int(len(dataframe) \* ratio)  
 train = dataframe.iloc[:train\_size]  
 test = dataframe.iloc[train\_size:]  
 return train, test  
  
  
def scale\_minmax\_scaler(train, test):  
 scaler = MinMaxScaler()  
 scaler\_train = scaler.fit(train)  
 scaler\_train = scaler.transform(train)  
 scaler\_test = scaler.transform(test)  
 return scaler\_train, scaler\_test, scaler  
  
  
def create\_timeSeries\_generater(scaler\_train):  
 n\_input = 12  
 n\_features = 1  
 generator = TimeseriesGenerator(scaler\_train, scaler\_train, length=n\_input, batch\_size=1)  
 return generator  
  
  
def import\_dataframe\_from\_csv(csv\_name):  
 directory = os.getcwd() + '/app/util/datasets/' + csv\_name  
 df = pd.read\_csv(directory)  
 return df