NO.1 You work for a biotech startup that is experimenting with deep learning ML models based on properties of biological organisms. Your team frequently works on early-stage experiments with new architectures of ML models, and writes custom TensorFlow ops in C++. You train your models on large datasets and large batch sizes. Your typical batch size has 1024 examples, and each example is about 1 MB in size. The average size of a network with all weights and embeddings is 20 GB. What hardware should you choose for your models?

A. A cluster with 2 n1-highcpu-64 machines, each with 8 NVIDIA Tesla V100 GPUs (128 GB GPU memory in total), and a n1-highcpu-64 machine with 64 vCPUs and 58 GB RAM

B. A cluster with 2 a2-megagpu-16g machines, each with 16 NVIDIA Tesla A100 GPUs (640 GB GPU memory in total), 96 vCPUs, and 1.4 TB RAM

C. A cluster with an n1-highcpu-64 machine with a v2-8 TPU and 64 GB RAM

D. A cluster with 4 n1-highcpu-96 machines, each with 96 vCPUs and 86 GB RAM

Answer: B

Explanation:

The best hardware to choose for your models is a cluster with 2 a2-megagpu-16g machines, each with 16 NVIDIA Tesla A100 GPUs (640 GB GPU memory in total), 96 vCPUs, and 1.4 TB RAM. This hardware configuration can provide you with enough compute power, memory, and bandwidth to handle your large and complex deep learning models, as well as your custom TensorFlow ops in C++. The NVIDIA Tesla A100 GPUs are the latest and most advanced GPUs from NVIDIA, which offer high performance, scalability, and efficiency for various ML workloads. They also support multi-instance GPU (MIG) technology, which allows you to partition each GPU into up to seven smaller instances, each with its own memory, cache, and compute cores. This can enable you to run multiple experiments in parallel, or to optimize the resource utilization and cost efficiency of your models. The a2-megagpu-16g machines are part of the Google Cloud Accelerator-Optimized VM (A2) family, which are designed to provide the best performance and flexibility for GPU-intensive applications. They also offer high-speed NVLink interconnects between the GPUs, which can improve the data transfer and communication between the GPUs. Moreover, the a2-megagpu-16g machines have 96 vCPUs and 1.4 TB RAM, which can support the CPU and memory requirements of your models, as well as the data preprocessing and postprocessing tasks.

The other options are not optimal for the following reasons:

* A. A cluster with 2 n1-highcpu-64 machines, each with 8 NVIDIA Tesla V100 GPUs (128 GB GPU memory in total), and a n1-highcpu-64 machine with 64 vCPUs and 58 GB RAM is not a good option, as it has less GPU memory, compute power, and bandwidth than the a2-megagpu-16g machines. The NVIDIA Tesla V100 GPUs are the previous generation of GPUs from NVIDIA, which have lower performance, scalability, and efficiency than the NVIDIA Tesla A100 GPUs. They also do not support the MIG technology, which can limit the flexibility and optimization of your models. Moreover, the n1-highcpu-64 machines are part of the Google Cloud N1 VM family, which are general-purpose VMs that do not offer the best performance and features for GPU-intensive applications. They also have lower vCPUs and RAM than the a2-megagpu-16g machines, which can affect the CPU and memory requirements of your models, as well as the data preprocessing and postprocessing tasks.
* C. A cluster with an n1-highcpu-64 machine with a v2-8 TPU and 64 GB RAM is not a good option, as it has less GPU memory, compute power, and bandwidth than the a2-megagpu-16g machines. The v2-8
* TPU is a cloud tensor processing unit (TPU) device, which is a custom ASIC chip designed by Google to accelerate ML workloads. However, the v2-8 TPU is the second generation of TPUs, which have

lower performance, scalability, and efficiency than the latest v3-8 TPUs. They also have less memory and bandwidth than the NVIDIA Tesla A100 GPUs, which can limit the size and complexity of your models, as well as the data transfer and communication between the devices. Moreover, the n1-highcpu-64 machine has lower vCPUs and RAM than the a2-megagpu-16g machines, which can affect the CPU and memory requirements of your models, as well as the data preprocessing and postprocessing tasks.
* D. A cluster with 4 n1-highcpu-96 machines, each with 96 vCPUs and 86 GB RAM is not a good option, as it does not have any GPUs, which are essential for accelerating deep learning models. The n1-highcpu-96 machines are part of the Google Cloud N1 VM family, which are general-purpose VMs that do not offer the best performance and features for GPU-intensive applications. They also have lower RAM than the a2-megagpu-16g machines, which can affect the memory requirements of your models, as well as the data preprocessing and postprocessing tasks.
References:
* Professional ML Engineer Exam Guide
* Preparing for Google Cloud Certification: Machine Learning Engineer Professional Certificate
* Google Cloud launches machine learning engineer certification
* NVIDIA Tesla A100 GPU
* Google Cloud Accelerator-Optimized VM (A2) family
* Google Cloud N1 VM family
* Cloud TPU

NO.2 You are an ML engineer at a global car manufacturer. You need to build an ML model to predict car sales in different cities around the world. Which features or feature crosses should you use to train city-specific relationships between car type and number of sales?

A. Three individual features binned latitude, binned longitude, and one-hot encoded car type

B. One feature obtained as an element-wise product between latitude, longitude, and car type

C. One feature obtained as an element-wise product between binned latitude, binned longitude, and one-hot encoded car type

D. Two feature crosses as a element-wise product the first between binned latitude and one-hot encoded car type, and the second between binned longitude and one-hot encoded car type

Answer: C

Explanation:
A feature cross is a synthetic feature that is obtained by combining two or more existing features, usually by taking their product or concatenation. A feature cross can help to capture the nonlinear and interaction effects between the original features, and improve the predictive performance of the model. A feature cross can be applied to different types of features, such as numeric, categorical, or geospatial features1.
For the use case of building an ML model to predict car sales in different cities around the world, the best option is to use one feature obtained as an element-wise product between binned latitude, binned longitude, and one-hot encoded car type. This option involves creating a feature cross that combines three individual features: binned latitude, binned longitude, and one-hot encoded car type. Binning is a technique that transforms a continuous numeric feature into a discrete categorical feature by dividing its range into equal intervals, or bins. One-hot encoding is a technique that transforms a categorical feature into a binary vector, where each element corresponds to a possible category, and has a value of 1 if the feature belongs to that category, and 0 otherwise. By applying binning and one-hot encoding to the latitude, longitude, and car type features, the feature cross can

capture the city-specific relationships between car type and number of sales, as each combination of bins and car types can represent a different city and its preference for a certain car type.
For example, the feature cross can learn that a city with a latitude bin of [40, 50], a longitude bin of [-80, -70], and a car type of SUV has a higher number of sales than a city with a latitude bin of [-10, 0], a longitude bin of
[10, 20], and a car type of sedan. Therefore, using one feature obtained as an element-wise product between binned latitude, binned longitude, and one-hot encoded car type is the best option for this use case.
References:
* Feature Crosses | Machine Learning Crash Course

NO.3 You work at a gaming startup that has several terabytes of structured data in Cloud Storage. This data includes gameplay time data user metadata and game metadata. You want to build a model that recommends new games to users that requires the least amount of coding. What should you do ?

A. Load the data in BigQuery Use BigQuery ML to tram an Autoencoder model.

B. Load the data in BigQuery Use BigQuery ML to train a matrix factorization model.

C. Read data to a Vertex AI Workbench notebook Use TensorFlow to train a two-tower model.

D. Read data to a Vertex AI Workbench notebook Use TensorFlow to train a matrix factorization model.

Answer: B
Explanation:
BigQuery is a serverless data warehouse that allows you to perform SQL queries on large-scale data. BigQuery ML is a feature of BigQuery that enables you to create and execute machine learning models using standard SQL queries. You can use BigQuery ML to train a matrix factorization model, which is a common technique for recommender systems. Matrix factorization models learn the latent factors that represent the preferences of users and the characteristics of items, and use them to predict the ratings or interactions between users and items. You can use the CREATE MODEL statement to create a matrix factorization model in BigQuery ML, and specify the matrix_factorization option as the model type. You can also use the ML.RECOMMEND function to generate recommendations for new games based on the trained model.
This solution requires the least amount of coding, as you only need to write SQL queries to train and use the model. References: The answer can be verified from official Google Cloud documentation and resources related to BigQuery and BigQuery ML.
* BigQuery ML | Google Cloud
* Using matrix factorization | BigQuery ML
* ML.RECOMMEND function | BigQuery ML

NO.4 You need to design a customized deep neural network in Keras that will predict customer purchases based on their purchase history. You want to explore model performance using multiple model architectures, store training data, and be able to compare the evaluation metrics in the same dashboard. What should you do?

A. Create multiple models using AutoML Tables

B. Automate multiple training runs using Cloud Composer

C. Run multiple training jobs on AI Platform with similar job names

D. Create an experiment in Kubeflow Pipelines to organize multiple runs

Answer: D

Explanation:

Kubeflow Pipelines is a service that allows you to create and run machine learning workflows on Google Cloud using various features, model architectures, and hyperparameters. You can use Kubeflow Pipelines to scale up your workflows, leverage distributed training, and access specialized hardware such as GPUs and TPUs1. An experiment in Kubeflow Pipelines is a workspace where you can try different configurations of your pipelines and organize your runs into logical groups. You can use experiments to compare the performance of different models and track the evaluation metrics in the same dashboard2.

For the use case of designing a customized deep neural network in Keras that will predict customer purchases based on their purchase history, the best option is to create an experiment in Kubeflow Pipelines to organize multiple runs. This option allows you to explore model performance using multiple model architectures, store training data, and compare the evaluation metrics in the same dashboard. You can use Keras to build and train your deep neural network models, and then package them as pipeline components that can be reused and combined with other components. You can also use Kubeflow Pipelines SDK to define and submit your pipelines programmatically, and use Kubeflow Pipelines UI to monitor and manage your experiments.

Therefore, creating an experiment in Kubeflow Pipelines to organize multiple runs is the best option for this use case.

References:

* Kubeflow Pipelines documentation
* Experiment | Kubeflow

NO.5 Your team has a model deployed to a Vertex AI endpoint You have created a Vertex AI pipeline that automates the model training process and is triggered by a Cloud Function. You need to prioritize keeping the model up-to-date, but also minimize retraining costs. How should you configure retraining'?

A. Configure Pub/Sub to call the Cloud Function when a sufficient amount of new data becomes available.

B. Configure a Cloud Scheduler job that calls the Cloud Function at a predetermined frequency that fits your team's budget.

C. Enable model monitoring on the Vertex AI endpoint Configure Pub/Sub to call the Cloud Function when anomalies are detected.

D. Enable model monitoring on the Vertex AI endpoint Configure Pub/Sub to call the Cloud Function when feature drift is detected.

Answer: D

Explanation:

According to the official exam guide1, one of the skills assessed in the exam is to "configure and optimize model monitoring jobs". Vertex AI Model Monitoring documentation states that "model monitoring helps you detect when your model's performance degrades over time due to changes in the data that your model receives or returns" and that "you can configure model monitoring to send notifications to Pub/Sub when it detects anomalies or drift in your model's predictions"2. Therefore, enabling model monitoring on the Vertex AI endpoint and configuring Pub/Sub to call the Cloud Function when feature drift is detected would help you keep the model up-to-date and minimize

retraining costs. The other options are not relevant or optimal for this scenario. References:
* Professional ML Engineer Exam Guide
* Vertex AI Model Monitoring
* Google Professional Machine Learning Certification Exam 2023
* Latest Google Professional Machine Learning Engineer Actual Free Exam Questions

NO.6 You need to analyze user activity data from your company's mobile applications. Your team will use BigQuery for data analysis, transformation, and experimentation with ML algorithms. You need to ensure real-time ingestion of the user activity data into BigQuery. What should you do?

A. Configure Pub/Sub to stream the data into BigQuery.

B. Run an Apache Spark streaming job on Dataproc to ingest the data into BigQuery.

C. Run a Dataflow streaming job to ingest the data into BigQuery.

D. Configure Pub/Sub and a Dataflow streaming job to ingest the data into BigQuery,

Answer: C

Explanation:
The best option to ensure real-time ingestion of the user activity data into BigQuery is to run a Dataflow streaming job to ingest the data into BigQuery. Dataflow is a fully managed service that can handle both batch and stream processing of data, and can integrate seamlessly with BigQuery and other Google Cloud services.
Dataflow can also use Apache Beam as the programming model, which provides a unified and portable API for developing data pipelines. By using Dataflow, you can avoid the complexity and overhead of managing your own infrastructure, and focus on the logic and transformation of your data. Dataflow can also handle various types of data, such as structured, unstructured, or binary data, and can apply windowing, aggregation, and other operations on the data streams.
The other options are not optimal for the following reasons:
* A. Configuring Pub/Sub to stream the data into BigQuery is not a good option, as Pub/Sub is a messaging service that can publish and subscribe to data streams, but cannot perform any transformation or processing on the data. Pub/Sub can be used as a source or a sink for Dataflow, but not as a standalone solution for ingesting data into BigQuery.
* B. Running an Apache Spark streaming job on Dataproc to ingest the data into BigQuery is not a good option, as it requires setting up and managing your own cluster of virtual machines, which can increase the cost and complexity of your solution. Moreover, Apache Spark is not natively integrated with BigQuery, and requires using connectors or intermediate storage to write data to BigQuery, which can introduce latency and inefficiency.
* D. Configuring Pub/Sub and a Dataflow streaming job to ingest the data into BigQuery is not a bad option, but it is not necessary, as Dataflow can directly read data from the mobile applications without using Pub/Sub as an intermediary. Using Pub/Sub can add an extra layer of abstraction and reliability, but it can also increase the cost and complexity of your solution, and introduce some delay in the data ingestion.
References:
* Professional ML Engineer Exam Guide
* Preparing for Google Cloud Certification: Machine Learning Engineer Professional Certificate
* Google Cloud launches machine learning engineer certification
* Dataflow documentation
* BigQuery documentation

NO.7 Your data science team has requested a system that supports scheduled model retraining, Docker containers, and a service that supports autoscaling and monitoring for online prediction requests. Which platform components should you choose for this system?

A. Vertex AI Pipelines and App Engine

B. Vertex AI Pipelines and AI Platform Prediction

C. Cloud Composer, BigQuery ML , and AI Platform Prediction

D. Cloud Composer, AI Platform Training with custom containers, and App Engine

Answer: B

Explanation:

Vertex AI Pipelines and AI Platform Prediction are the platform components that best suit the requirements of the data science team. Vertex AI Pipelines is a service that allows you to orchestrate and automate your machine learning workflows using pipelines. Pipelines are portable and scalable ML workflows that are based on containers. You can use Vertex AI Pipelines to schedule model retraining, use custom containers, and integrate with other Google Cloud services. AI Platform Prediction is a service that allows you to host your trained models and serve online predictions. You can use AI Platform Prediction to deploy models trained on Vertex AI or elsewhere, and benefit from features such as autoscaling, monitoring, logging, and explainability. References:

* Vertex AI Pipelines
* AI Platform Prediction

NO.8 You are developing an ML model intended to classify whether X-Ray images indicate bone fracture risk. You have trained on Api Resnet architecture on Vertex AI using a TPU as an accelerator, however you are unsatisfied with the trainning time and use memory usage. You want to quickly iterate your training code but make minimal changes to the code. You also want to minimize impact on the models accuracy. What should you do?

A. Configure your model to use bfloat16 instead float32

B. Reduce the global batch size from 1024 to 256

C. Reduce the number of layers in the model architecture

D. Reduce the dimensions of the images used un the model

Answer: A

Explanation:

Using bfloat16 instead of float32 can reduce the memory usage and training time of the model, while having minimal impact on the accuracy. Bfloat16 is a 16-bit floating-point format that preserves the range of 32-bit floating-point numbers, but reduces the precision from 24 bits to 8 bits. This means that bfloat16 can store the same magnitude of numbers as float32, but with less detail. Bfloat16 is supported by TPUs and some GPUs, and can be used as a drop-in replacement for float32 in most cases. Bfloat16 can also improve the numerical stability of the model, as it reduces the risk of overflow and underflow errors.

Reducing the global batch size, the number of layers, or the dimensions of the images can also reduce the memory usage and training time of the model, but they can also affect the model's accuracy and performance.

Reducing the global batch size can make the model less stable and converge slower, as it reduces the amount of information available for each gradient update. Reducing the number of layers can make the model less expressive and powerful, as it reduces the depth and complexity of the network. Reducing the dimensions of the images can make the model less accurate and robust, as it reduces

the resolution and quality of the input data. References:
* Bfloat16: The secret to high performance on Cloud TPUs
* Bfloat16 floating-point format
* How does Batch Size impact your model learning

NO.9 You trained a text classification model. You have the following SignatureDefs:

```
signature_def['serving_default']:
  The given SavedModel SignatureDef contains the following input(s):
    inputs['text'] tensor_info:
        dtype: DT_STRING
        shape: (-1, 2)
        name: serving_default_text:0
  The given SavedModel SignatureDef contains the following output(s):
    outputs['Softmax'] tensor_info:
        dtype: DT_FLOAT
        shape: (-1, 2)
        name: StatefulPartitionedCall:0
  Method name is: tensorflow/serving/predict
```

You started a TensorFlow-serving component server and tried to send an HTTP request to get a prediction using:

```
headers = {"content-type": "application/json"}
json_response = requests.post('http://localhost:8501/v1/models/text_model:predict', data=data,
headers=headers)
```

What is the correct way to write the predict request?

A. data = json.dumps({"signature_name": "serving_default'\ "instances": [fab', 'be1, 'cd']]})

B. data = json dumps({"signature_name": "serving_default"! "instances": [['a', 'b', "c", 'd', 'e', 'f']]})

C. data = json.dumps({"signature_name": "serving_default, "instances": [['a', 'b\ 'c'1, [d\ 'e\ T]]})

D. data = json dumps({"signature_name": f,serving_default", "instances": [['a', 'b'], [c\ 'd'], ['e\ T]]})

Answer: D

Explanation:

A predict request is a way to send data to a trained model and get predictions in return. A predict request can be written in different formats, such as JSON, protobuf, or gRPC, depending on the service and the platform that are used to host and serve the model. A predict request usually contains the following information:
* The signature name: This is the name of the signature that defines the inputs and outputs of the model. A signature is a way to specify the expected format, type, and shape of the data that the model can accept and produce. A signature can be specified when exporting or saving the model, or it can be automatically inferred by the service or the platform. A model can have multiple signatures, but only one can be used for each predict request.
* The instances: This is the data that is sent to the model for prediction. The instances can be a single instance or a batch of instances, depending on the size and shape of the data. The instances should match the input specification of the signature, such as the number, name, and type of the input tensors.

For the use case of training a text classification model, the correct way to write the predict request is D. data = json.dumps({"signature_name": "serving_default", "instances": [['a', 'b'], ['c', 'd'], ['e', 'f']]})
This option involves writing the predict request in JSON format, which is a common and convenient format for sending and receiving data over the web. JSON stands for JavaScript Object Notation, and

it is a way to represent data as a collection of name-value pairs or an ordered list of values. JSON can be easily converted to and from Python objects using the json module.
This option also involves using the signature name "serving_default", which is the default signature name that is assigned to the model when it is saved or exported without specifying a custom signature name. The serving_default signature defines the input and output tensors of the model based on the SignatureDef that is shown in the image. According to the SignatureDef, the model expects an input tensor called "text" that has a shape of (-1, 2) and a type of DT_STRING, and produces an output tensor called "softmax" that has a shape of (-1, 2) and a type of DT_FLOAT. The -1 in the shape indicates that the dimension can vary depending on the number of instances, and the 2 indicates that the dimension is fixed at 2. The DT_STRING and DT_FLOAT indicate that the data type is string and float, respectively.
This option also involves sending a batch of three instances to the model for prediction. Each instance is a list of two strings, such as ['a', 'b'], ['c', 'd'], or ['e', 'f']. These instances match the input specification of the signature, as they have a shape of (3, 2) and a type of string. The model will process these instances and produce a batch of three predictions, each with a softmax output that has a shape of (1, 2) and a type of float.
The softmax output is a probability distribution over the two possible classes that the model can predict, such as positive or negative sentiment.
Therefore, writing the predict request as data = json.dumps({"signature_name": "serving_default", "instances": [['a', 'b'], ['c', 'd'], ['e', 'f']]}) is the correct and valid way to send data to the text classification model and get predictions in return.
References:
* [json - JSON encoder and decoder]

NO.10 You work for a company that manages a ticketing platform for a large chain of cinemas. Customers use a mobile app to search for movies they're interested in and purchase tickets in the app. Ticket purchase requests are sent to Pub/Sub and are processed with a Dataflow streaming pipeline configured to conduct the following steps:
1. Check for availability of the movie tickets at the selected cinema.
2. Assign the ticket price and accept payment.
3. Reserve the tickets at the selected cinema.
4. Send successful purchases to your database.
Each step in this process has low latency requirements (less than 50 milliseconds). You have developed a logistic regression model with BigQuery ML that predicts whether offering a promo code for free popcorn increases the chance of a ticket purchase, and this prediction should be added to the ticket purchase process.
You want to identify the simplest way to deploy this model to production while adding minimal latency. What should you do?

A. Run batch inference with BigQuery ML every five minutes on each new set of tickets issued.

B. Export your model in TensorFlow format, and add a tfx_bsl.public.beam.RunInference step to the Dataflow pipeline.

C. Export your model in TensorFlow format, deploy it on Vertex AI, and query the prediction endpoint from your streaming pipeline.

D. Convert your model with TensorFlow Lite (TFLite), and add it to the mobile app so that the promo code and the incoming request arrive together in Pub/Sub.

Answer: B

Explanation:
The simplest way to deploy a logistic regression model with BigQuery ML to production while adding minimal latency is to export the model in TensorFlow format, and add a tfx_bsl.public.beam.RunInference step to the Dataflow pipeline. This option has the following advantages:
* It allows the model prediction to be performed in real time, as part of the Dataflow streaming pipeline that processes the ticket purchase requests. This ensures that the promo code offer is based on the most recent data and customer behavior, and that the offer is delivered to the customer without delay.
* It leverages the compatibility and performance of TensorFlow and Dataflow, which are both part of the Google Cloud ecosystem. TensorFlow is a popular and powerful framework for building and deploying machine learning models, and Dataflow is a fully managed service that runs Apache Beam pipelines for data processing and transformation. By using the tfx_bsl.public.beam.RunInference step, you can easily integrate your TensorFlow model with your Dataflow pipeline, and take advantage of the parallelism and scalability of Dataflow.
* It simplifies the model deployment and management, as the model is packaged with the Dataflow pipeline and does not require a separate service or endpoint. The model can be updated by redeploying the Dataflow pipeline with a new model version.
The other options are less optimal for the following reasons:
* Option A: Running batch inference with BigQuery ML every five minutes on each new set of tickets issued introduces additional latency and complexity. This option requires running a separate BigQuery job every five minutes, which can incur network overhead and latency. Moreover, this option requires storing and retrieving the intermediate results of the batch inference, which can consume storage space and increase the data transfer time.
* Option C: Exporting the model in TensorFlow format, deploying it on Vertex AI, and querying the prediction endpoint from the streaming pipeline introduces additional latency and cost. This option requires creating and managing a Vertex AI endpoint, which is a managed service that provides various tools and features for machine learning, such as training, tuning, serving, and monitoring. However, querying the Vertex AI endpoint from the streaming pipeline requires making an HTTP request, which can incur network overhead and latency. Moreover, this option requires paying for the Vertex AI endpoint usage, which can increase the cost of the model deployment.
* Option D: Converting the model with TensorFlow Lite (TFLite), and adding it to the mobile app so that the promo code and the incoming request arrive together in Pub/Sub introduces additional challenges and risks. This option requires converting the model to a TFLite format, which is a lightweight and optimized format for running TensorFlow models on mobile and embedded devices. However, converting the model to TFLite may not preserve the accuracy or functionality of the original model, as
* some operations or features may not be supported by TFLite. Moreover, this option requires updating the mobile app with the TFLite model, which can be tedious and time-consuming, and may depend on the user's willingness to update the app. Additionally, this option may expose the model to potential security or privacy issues, as the model is running on the user's device and may be accessed or modified by malicious actors.
References:
* [Exporting models for prediction | BigQuery ML]
* [tfx_bsl.public.beam.run_inference | TensorFlow Extended]
* [Vertex AI documentation]

* [TensorFlow Lite documentation]

NO.11 You work for an online publisher that delivers news articles to over 50 million readers. You have built an AI model that recommends content for the company's weekly newsletter. A recommendation is considered successful if the article is opened within two days of the newsletter's published date and the user remains on the page for at least one minute.
All the information needed to compute the success metric is available in BigQuery and is updated hourly. The model is trained on eight weeks of data, on average its performance degrades below the acceptable baseline after five weeks, and training time is 12 hours. You want to ensure that the model's performance is above the acceptable baseline while minimizing cost. How should you monitor the model to determine when retraining is necessary?

A. Use Vertex AI Model Monitoring to detect skew of the input features with a sample rate of 100% and a monitoring frequency of two days.

B. Schedule a cron job in Cloud Tasks to retrain the model every week before the newsletter is created.

C. Schedule a weekly query in BigQuery to compute the success metric.

D. Schedule a daily Dataflow job in Cloud Composer to compute the success metric.

Answer: C

Explanation:
The best option for monitoring the model to determine when retraining is necessary is to schedule a weekly query in BigQuery to compute the success metric. This option has the following advantages:
* It allows the model performance to be evaluated regularly, based on the actual outcome of the recommendations. By computing the success metric, which is the percentage of articles that are opened within two days and read for at least one minute, you can measure how well the model is achieving its objective and compare it with the acceptable baseline.
* It leverages the scalability and efficiency of BigQuery, which is a serverless, fully managed, and highly scalable data warehouse that can run complex queries over petabytes of data in seconds. By using BigQuery, you can access and analyze all the information needed to compute the success metric, such as the newsletter publication date, the article opening date, and the user reading time, without worrying about the infrastructure or the cost.
* It simplifies the model monitoring and retraining workflow, as the weekly query can be scheduled and executed automatically using BigQuery's built-in scheduling feature. You can also set up alerts or
* notifications to inform you when the success metric falls below the acceptable baseline, and trigger the model retraining process accordingly.
The other options are less optimal for the following reasons:
* Option A: Using Vertex AI Model Monitoring to detect skew of the input features with a sample rate of
100% and a monitoring frequency of two days introduces additional complexity and overhead. This option requires setting up and managing a Vertex AI Model Monitoring service, which is a managed service that provides various tools and features for machine learning, such as training, tuning, serving, and monitoring. However, using Vertex AI Model Monitoring to detect skew of the input features may not reflect the actual performance of the model, as skew is the discrepancy between the distributions of the features in the training dataset and the serving data, which may not affect the outcome of the recommendations. Moreover, using a sample rate of 100% and a monitoring frequency of two days may incur unnecessary cost and latency, as it requires analyzing all the input features every two days, which may not be needed for the model monitoring.

* Option B: Scheduling a cron job in Cloud Tasks to retrain the model every week before the newsletter is created introduces additional cost and risk. This option requires creating and running a cron job in Cloud Tasks, which is a fully managed service that allows you to schedule and execute tasks that are invoked by HTTP requests. However, using Cloud Tasks to retrain the model every week may not be optimal, as it may retrain the model more often than necessary, wasting compute resources and cost. Moreover, using Cloud Tasks to retrain the model before the newsletter is created may introduce risk, as it may deploy a new model version that has not been tested or validated, potentially affecting the quality of the recommendations.
* Option D: Scheduling a daily Dataflow job in Cloud Composer to compute the success metric introduces additional complexity and cost. This option requires creating and running a Dataflow job in Cloud Composer, which is a fully managed service that runs Apache Airflow pipelines for workflow orchestration. Dataflow is a fully managed service that runs Apache Beam pipelines for data processing and transformation. However, using Dataflow and Cloud Composer to compute the success metric may not be necessary, as it may add more steps and overhead to the model monitoring process. Moreover, using Dataflow and Cloud Composer to compute the success metric daily may not be optimal, as it may compute the success metric more often than needed, consuming more compute resources and cost.
References:
* [BigQuery documentation]
* [Vertex AI Model Monitoring documentation]
* [Cloud Tasks documentation]
* [Cloud Composer documentation]
* [Dataflow documentation]

NO.12 You work for a magazine distributor and need to build a model that predicts which customers will renew their subscriptions for the upcoming year. Using your company's historical data as your training set, you created a TensorFlow model and deployed it to AI Platform. You need to determine which customer attribute has the most predictive power for each prediction served by the model. What should you do?

A. Use AI Platform notebooks to perform a Lasso regression analysis on your model, which will eliminate features that do not provide a strong signal.

B. Stream prediction results to BigQuery. Use BigQuery's CORR(X1, X2) function to calculate the Pearson correlation coefficient between each feature and the target variable.

C. Use the AI Explanations feature on AI Platform. Submit each prediction request with the 'explain' keyword to retrieve feature attributions using the sampled Shapley method.

D. Use the What-If tool in Google Cloud to determine how your model will perform when individual features are excluded. Rank the feature importance in order of those that caused the most significant performance drop when removed from the model.

Answer: C

Explanation:
* Option A is incorrect because using AI Platform notebooks to perform a Lasso regression analysis on your model, which will eliminate features that do not provide a strong signal, is not a suitable way to determine which customer attribute has the most predictive power for each prediction served by the model. Lasso regression is a method of feature selection that applies a penalty to the coefficients of the linear model, and shrinks them to zero for irrelevant features1. However, this method assumes

that the model is linear and additive, which may not be the case for a TensorFlow model. Moreover, this method does not provide feature attributions for each prediction, but rather for the entire dataset.
* Option B is incorrect because streaming prediction results to BigQuery, and using BigQuery's CORR(X1, X2) function to calculate the Pearson correlation coefficient between each feature and the target variable, is not a valid way to determine which customer attribute has the most predictive power for each prediction served by the model. The Pearson correlation coefficient is a measure of the linear relationship between two variables, ranging from -1 to 12. However, this method does not account for the interactions between features or the non-linearity of the model. Moreover, this method does not provide feature attributions for each prediction, but rather for the entire dataset.
* Option C is correct because using the AI Explanations feature on AI Platform, and submitting each prediction request with the 'explain' keyword to retrieve feature attributions using the sampled Shapley method, is the best way to determine which customer attribute has the most predictive power for each prediction served by the model. AI Explanations is a service that allows you to get feature attributions for your deployed models on AI Platform3. Feature attributions are values that indicate how much each feature contributed to the prediction for a given instance4. The sampled Shapley method is a technique that uses the Shapley value, a game-theoretic concept, to measure the contribution of each feature to the prediction5. By using AI Explanations, you can get feature attributions for each prediction request, and identify the most important features for each customer.
* Option D is incorrect because using the What-If tool in Google Cloud to determine how your model will perform when individual features are excluded, and ranking the feature importance in order of those that caused the most significant performance drop when removed from the model, is not a practical way to determine which customer attribute has the most predictive power for each prediction served by the model. The What-If tool is a tool that allows you to visualize and analyze your ML models and datasets.
* However, this method requires manually editing or removing features for each instance, and observing the change in the prediction. This method is not scalable or efficient, and may not capture the interactions between features or the non-linearity of the model.
References:
* Lasso regression
* Pearson correlation coefficient
* AI Explanations overview
* Feature attributions
* Sampled Shapley method
* [What-If tool overview]

NO.13 You built a deep learning-based image classification model by using on-premises data. You want to use Vertex AI to deploy the model to production Due to security concerns you cannot move your data to the cloud. You are aware that the input data distribution might change over time You need to detect model performance changes in production. What should you do?

A. Use Vertex Explainable AI for model explainability Configure feature-based explanations.

B. Use Vertex Explainable AI for model explainability Configure example-based explanations.

C. Create a Vertex AI Model Monitoring job. Enable training-serving skew detection for your model.

D. Create a Vertex AI Model Monitoring job. Enable feature attribution skew and dnft detection for your model.

Answer: C

Explanation:
Vertex AI Model Monitoring is a service that allows you to monitor the performance and quality of your ML models in production. You can use Vertex AI Model Monitoring to detect changes in the input data distribution, the prediction output distribution, or the model accuracy over time. Training-serving skew detection is a feature of Vertex AI Model Monitoring that compares the statistics of the data used for training the model and the data used for serving the model. If there is a significant difference between the two data distributions, it indicates that the model might be outdated or inaccurate. By enabling training-serving skew detection for your model, you can detect model performance changes in production and trigger retraining or redeployment of your model as needed. This way, you can ensure that your model is always up-to-date and accurate, without moving your data to the cloud. References:
* Vertex AI Model Monitoring documentation
* Training-serving skew detection documentation
* Preparing for Google Cloud Certification: Machine Learning Engineer Professional Certificate

NO.14 You are developing a recommendation engine for an online clothing store. The historical customer transaction data is stored in BigQuery and Cloud Storage. You need to perform exploratory data analysis (EDA), preprocessing and model training. You plan to rerun these EDA, preprocessing, and training steps as you experiment with different types of algorithms. You want to minimize the cost and development effort of running these steps as you experiment. How should you configure the environment?
A. Create a Vertex AI Workbench user-managed notebook using the default VM instance, and use th
e
%%bigquery magic commands in Jupyter to query the tables.
B. Create a Vertex AI Workbench managed notebook to browse and query the tables directly from the JupyterLab interface.
C. Create a Vertex AI Workbench user-managed notebook on a Dataproc Hub. and use the %%bigquery magic commands in Jupyter to query the tables.
D. Create a Vertex AI Workbench managed notebook on a Dataproc cluster, and use the spark-bigquery-connector to access the tables.
Answer: A
Explanation:
* Cost-effectiveness: User-managed notebooks in Vertex AI Workbench allow you to leverage pre-configured virtual machines with reasonable resource allocation, keeping costs lower compared to options involving managed notebooks or Dataproc clusters.
* Development flexibility: User-managed notebooks offer full control over the environment, allowing you to install additional libraries or dependencies needed for your specific EDA, preprocessing, and model training tasks. This flexibility is crucial while experimenting with different algorithms.
* BigQuery integration: The %%bigquery magic commands provide seamless integration with BigQuery within the Jupyter Notebook environment. This enables efficient querying and exploration of customer transaction data stored in BigQuery directly from the notebook, streamlining the workflow.
Other options and why they are not the best fit:
* B. Managed notebook: While managed notebooks offer an easier setup, they might have limited customization options, potentially hindering your ability to install specific libraries or tools.

* C. Dataproc Hub: Dataproc Hub focuses on running large-scale distributed workloads, and it might be overkill for your scenario involving exploratory analysis and experimentation with different algorithms.
Additionally, it could incur higher costs compared to a user-managed notebook.
* D. Dataproc cluster with spark-bigquery-connector: Similar to option C, using a Dataproc cluster with the spark-bigquery-connector would be more complex and potentially more expensive than using
%%bigquery magic commands within a user-managed notebook for accessing BigQuery data.
References:
* https://cloud.google.com/vertex-ai/docs/workbench/instances/bigquery
* https://cloud.google.com/vertex-ai-notebooks

NO.15 You are using Keras and TensorFlow to develop a fraud detection model Records of customer transactions are stored in a large table in BigQuery. You need to preprocess these records in a cost-effective and efficient way before you use them to train the model. The trained model will be used to perform batch inference in BigQuery. How should you implement the preprocessing workflow?

A. Implement a preprocessing pipeline by using Apache Spark, and run the pipeline on Dataproc Save the preprocessed data as CSV files in a Cloud Storage bucket.

B. Load the data into a pandas DataFrame Implement the preprocessing steps using panda's transformations. and train the model directly on the DataFrame.

C. Perform preprocessing in BigQuery by using SQL Use the BigQueryClient in TensorFlow to read the data directly from BigQuery.

D. Implement a preprocessing pipeline by using Apache Beam, and run the pipeline on Dataflow Save the preprocessed data as CSV files in a Cloud Storage bucket.

Answer: C
Explanation:
* Option A is not the best answer because it requires using Apache Spark and Dataproc, which may incur additional cost and complexity for running and managing the cluster. It also requires saving the preprocessed data as CSV files in a Cloud Storage bucket, which may increase the storage cost and the data transfer latency.
* Option B is not the best answer because it requires loading the data into a pandas DataFrame, which may not be scalable or efficient for large datasets. It also requires training the model directly on the DataFrame, which may not leverage the distributed computing capabilities of BigQuery.
* Option C is the best answer because it allows performing preprocessing in BigQuery by using SQL, which is a cost-effective and efficient way to manipulate large datasets. It also allows using the BigQueryClient in TensorFlow to read the data directly from BigQuery, which is a convenient and fast way to access the data for training the model1.
* Option D is not the best answer because it requires using Apache Beam and Dataflow, which may incur additional cost and complexity for running and managing the pipeline. It also requires saving the preprocessed data as CSV files in a Cloud Storage bucket, which may increase the storage cost and the data transfer latency.
References:
* 1: Read data from BigQuery | TensorFlow I/O

NO.16 You have a functioning end-to-end ML pipeline that involves tuning the hyperparameters of your ML model using AI Platform, and then using the best-tuned parameters for training.

Hypertuning is taking longer than expected and is delaying the downstream processes. You want to speed up the tuning job without significantly compromising its effectiveness. Which actions should you take?
Choose 2 answers

A. Decrease the number of parallel trials

B. Decrease the range of floating-point values

C. Set the early stopping parameter to TRUE

D. Change the search algorithm from Bayesian search to random search.

E. Decrease the maximum number of trials during subsequent training phases.

Answer: C E

Explanation:
Hyperparameter tuning is the process of finding the optimal values for the parameters of a machine learning model that affect its performance. AI Platform provides a service for hyperparameter tuning that can run multiple trials in parallel and use different search algorithms to find the best combination of hyperparameters.
However, hyperparameter tuning can be time-consuming and costly, especially if the search space is large and the model training is complex. Therefore, it is important to optimize the tuning job to reduce the time and resources required.
One way to speed up the tuning job is to set the early stopping parameter to TRUE. This means that the tuning service will automatically stop trials that are unlikely to perform well based on the intermediate results. This can save time and resources by avoiding unnecessary computations for trials that are not promising. The early stopping parameter can be set in the trainingInput.hyperparameters field of the training job request1 Another way to speed up the tuning job is to decrease the maximum number of trials during subsequent training phases. This means that the tuning service will use fewer trials to refine the search space after the initial phase. This can reduce the time required for the tuning job to converge to the optimal solution. The maximum number of trials can be set in the trainingInput.hyperparameters.maxTrials field of the training job request1 The other options are not effective ways to speed up the tuning job. Decreasing the number of parallel trials will reduce the concurrency of the tuning job and increase the overall time required. Decreasing the range of floating-point values will reduce the diversity of the search space and may miss some optimal solutions. Changing the search algorithm from Bayesian search to random search will reduce the efficiency of the tuning job and may require more trials to find the best solution1
References: 1: Hyperparameter tuning overview

NO.17 Your data science team needs to rapidly experiment with various features, model architectures, and hyperparameters. They need to track the accuracy metrics for various experiments and use an API to query the metrics over time. What should they use to track and report their experiments while minimizing manual effort?

A. Use Kubeflow Pipelines to execute the experiments Export the metrics file, and query the results using the Kubeflow Pipelines API.

B. Use AI Platform Training to execute the experiments Write the accuracy metrics to BigQuery, and query the results using the BigQueryAPI.

C. Use AI Platform Training to execute the experiments Write the accuracy metrics to Cloud Monitoring, and query the results using the Monitoring API.

D. Use AI Platform Notebooks to execute the experiments. Collect the results in a shared Google

Sheets file, and query the results using the Google Sheets API

Answer: C

Explanation:

AI Platform Training is a service that allows you to run your machine learning experiments on Google Cloud using various features, model architectures, and hyperparameters. You can use AI Platform Training to scale up your experiments, leverage distributed training, and access specialized hardware such as GPUs and TPUs1.

Cloud Monitoring is a service that collects and analyzes metrics, logs, and traces from Google Cloud, AWS, and other sources. You can use Cloud Monitoring to create dashboards, alerts, and reports based on your data2. The Monitoring API is an interface that allows you to programmatically access and manipulate your monitoring data3.

By using AI Platform Training and Cloud Monitoring, you can track and report your experiments while minimizing manual effort. You can write the accuracy metrics from your experiments to Cloud Monitoring using the AI Platform Training Python package4. You can then query the results using the Monitoring API and compare the performance of different experiments. You can also visualize the metrics in the Cloud Console or create custom dashboards and alerts5. Therefore, using AI Platform Training and Cloud Monitoring is the best option for this use case.

References:

* AI Platform Training documentation
* Cloud Monitoring documentation
* Monitoring API overview
* Using Cloud Monitoring with AI Platform Training
* Viewing evaluation metrics

NO.18 You work on a data science team at a bank and are creating an ML model to predict loan default risk. You have collected and cleaned hundreds of millions of records worth of training data in a BigQuery table, and you now want to develop and compare multiple models on this data using TensorFlow and Vertex AI. You want to minimize any bottlenecks during the data ingestion state while considering scalability. What should you do?

A. Use the BigQuery client library to load data into a dataframe, and use tf.data.Dataset.from_tensor_slices() to read it.

B. Export data to CSV files in Cloud Storage, and use tf.data.TextLineDataset() to read them.

C. Convert the data into TFRecords, and use tf.data.TFRecordDataset() to read them.

D. Use TensorFlow I/O's BigQuery Reader to directly read the data.

Answer: D

Explanation:

The best option for developing and comparing multiple models on a large-scale BigQuery table using TensorFlow and Vertex AI is to use TensorFlow I/O's BigQuery Reader to directly read the data. This option has the following advantages:

* It minimizes any bottlenecks during the data ingestion stage, as the BigQuery Reader can stream data from BigQuery to TensorFlow in parallel and in batches, without loading the entire table into memory or disk. The BigQuery Reader can also perform data transformations and filtering using SQL queries, reducing the need for additional preprocessing steps in TensorFlow.
* It leverages the scalability and performance of BigQuery, as the BigQuery Reader can handle hundreds of millions of records worth of training data efficiently and reliably. BigQuery is a serverless,

fully managed, and highly scalable data warehouse that can run complex queries over petabytes of data in seconds.
* It simplifies the integration with Vertex AI, as the BigQuery Reader can be used with both custom and pre-built TensorFlow models on Vertex AI. Vertex AI is a unified platform for machine learning that provides various tools and features for data ingestion, data labeling, data preprocessing, model training, model tuning, model deployment, model monitoring, and model explainability.
The other options are less optimal for the following reasons:
* Option A: Using the BigQuery client library to load data into a dataframe, and using tf.data.Dataset.from_tensor_slices() to read it, introduces memory and performance issues. This option requires loading the entire BigQuery table into a Pandas dataframe, which can consume a lot of memory and cause out-of-memory errors. Moreover, using tf.data.Dataset.from_tensor_slices() to read the dataframe can be slow and inefficient, as it creates one slice per row of the dataframe, resulting in a large number of small tensors.
* Option B: Exporting data to CSV files in Cloud Storage, and using tf.data.TextLineDataset() to read them, introduces additional steps and complexity. This option requires exporting the BigQuery table to one or more CSV files in Cloud Storage, which can take a long time and consume a lot of storage space.
Moreover, using tf.data.TextLineDataset() to read the CSV files can be slow and error-prone, as it requires parsing and decoding each line of text, handling missing values and invalid data, and applying data transformations and validations.
* Option C: Converting the data into TFRecords, and using tf.data.TFRecordDataset() to read them, introduces additional steps and complexity. This option requires converting the BigQuery table into one or more TFRecord files, which are binary files that store serialized TensorFlow examples. This can take a long time and consume a lot of storage space. Moreover, using tf.data.TFRecordDataset() to read the TFRecord files requires defining and parsing the schema of the TensorFlow examples, which can be tedious and error-prone.
References:
* [TensorFlow I/O documentation]
* [BigQuery documentation]
* [Vertex AI documentation]

NO.19 You work on an operations team at an international company that manages a large fleet of on-premises servers located in few data centers around the world. Your team collects monitoring data from the servers, including CPU/memory consumption. When an incident occurs on a server, your team is responsible for fixing it.
Incident data has not been properly labeled yet. Your management team wants you to build a predictive maintenance solution that uses monitoring data from the VMs to detect potential failures and then alerts the service desk team. What should you do first?

A. Train a time-series model to predict the machines' performance values. Configure an alert if a machine's actual performance values significantly differ from the predicted performance values.

B. Implement a simple heuristic (e.g., based on z-score) to label the machines' historical performance data.
Train a model to predict anomalies based on this labeled dataset.

C. Develop a simple heuristic (e.g., based on z-score) to label the machines' historical performance data.
Test this heuristic in a production environment.

D. Hire a team of qualified analysts to review and label the machines' historical performance data. Train a model based on this manually labeled dataset.

Answer: B

Explanation:

* Option A is incorrect because training a time-series model to predict the machines' performance values, and configuring an alert if a machine's actual performance values significantly differ from the predicted performance values, is not the best way to build a predictive maintenance solution that uses monitoring data from the VMs to detect potential failures and then alerts the service desk team. This option assumes that the performance values follow a predictable pattern, which may not be the case for complex systems. Moreover, this option does not use any historical incident data, which may contain useful information for identifying failures. Furthermore, this option does not involve any model evaluation or validation, which are essential steps for ensuring the quality and reliability of the model.
* Option B is correct because implementing a simple heuristic (e.g., based on z-score) to label the machines' historical performance data, and training a model to predict anomalies based on this labeled dataset, is a reasonable way to build a predictive maintenance solution that uses monitoring data from the VMs to detect potential failures and then alerts the service desk team. This option uses a simple and fast method to label the historical performance data, which is necessary for supervised learning. A z-score is a measure of how many standard deviations a value is away from the mean of a distribution1.

By using a z-score, we can label the performance values that are unusually high or low as anomalies, which may indicate failures. Then, we can train a model to learn the patterns of normal and anomalous performance values, and use it to predict anomalies on new data. We can also evaluate and validate the model using metrics such as precision, recall, or F1-score, and compare it with other models or methods.

* Option C is incorrect because developing a simple heuristic (e.g., based on z-score) to label the machines' historical performance data, and testing this heuristic in a production environment, is not a safe way to build a predictive maintenance solution that uses monitoring data from the VMs to detect potential failures and then alerts the service desk team. This option does not involve any model training or evaluation, which are essential steps for ensuring the quality and reliability of the solution. Moreover, this option does not test the heuristic on a separate dataset, such as a validation or test set, before deploying it to production, which may lead to errors or failures in the production environment.
* Option D is incorrect because hiring a team of qualified analysts to review and label the machines' historical performance data, and training a model based on this manually labeled dataset, is not a feasible way to build a predictive maintenance solution that uses monitoring data from the VMs to detect potential failures and then alerts the service desk team. This option may produce high-quality labels, but it is also costly, time-consuming, and prone to human errors or biases. Moreover, this option may not scale well with large or complex datasets, which may require more analysts or more time to label.

References:

* Z-score
* [Predictive maintenance]
* [Anomaly detection]
* [Time-series analysis]
* [Model evaluation]

NO.20 You work at an organization that maintains a cloud-based communication platform that integrates conventional chat, voice, and video conferencing into one platform. The audio recordings are stored in Cloud Storage. All recordings have an 8 kHz sample rate and are more than one minute long. You need to implement a new feature in the platform that will automatically transcribe voice call recordings into a text for future applications, such as call summarization and sentiment analysis. How should you implement the voice call transcription feature following Google-recommended best practices?

A. Use the original audio sampling rate, and transcribe the audio by using the Speech-to-Text API with synchronous recognition.

B. Use the original audio sampling rate, and transcribe the audio by using the Speech-to-Text API with asynchronous recognition.

C. Upsample the audio recordings to 16 kHz. and transcribe the audio by using the Speech-to-Text API with synchronous recognition.

D. Upsample the audio recordings to 16 kHz. and transcribe the audio by using the Speech-to-Text API with asynchronous recognition.

Answer: D

NO.21 You have a large corpus of written support cases that can be classified into 3 separate categories: Technical Support, Billing Support, or Other Issues. You need to quickly build, test, and deploy a service that will automatically classify future written requests into one of the categories. How should you configure the pipeline?

A. Use the Cloud Natural Language API to obtain metadata to classify the incoming cases.

B. Use AutoML Natural Language to build and test a classifier. Deploy the model as a REST API.

C. Use BigQuery ML to build and test a logistic regression model to classify incoming requests. Use BigQuery ML to perform inference.

D. Create a TensorFlow model using Google's BERT pre-trained model. Build and test a classifier, and deploy the model using Vertex AI.

Answer: B

Explanation:

AutoML Natural Language is a service that allows you to quickly build, test and deploy natural language processing (NLP) models without needing to have expertise in NLP or machine learning. You can use it to train a classifier on your corpus of written support cases, and then use the AutoML API to perform classification on new requests. Once the model is trained, it can be deployed as a REST API. This allows the classifier to be integrated into your pipeline and be easily consumed by other systems.

NO.22 You work with a team of researchers to develop state-of-the-art algorithms for financial analysis. Your team develops and debugs complex models in TensorFlow. You want to maintain the ease of debugging while also reducing the model training time. How should you set up your training environment?

A. Configure a v3-8 TPU VM SSH into the VM to tram and debug the model.

B. Configure a v3-8 TPU node Use Cloud Shell to SSH into the Host VM to train and debug the model.

C. Configure a M-standard-4 VM with 4 NVIDIA P100 GPUs SSH into the VM and use Parameter

Server Strategy to train the model.

D. Configure a M-standard-4 VM with 4 NVIDIA P100 GPUs SSH into the VM and use MultiWorkerMirroredStrategy to train the model.

Answer: A

Explanation:

A TPU VM is a virtual machine that has direct access to a Cloud TPU device. TPU VMs provide a simpler and more flexible way to use Cloud TPUs, as they eliminate the need for a separate host VM and network setup. TPU VMs also support interactive debugging tools such as TensorFlow Debugger (tfdbg) and Python Debugger (pdb), which can help researchers develop and troubleshoot complex models. A v3-8 TPU VM has

8 TPU cores, which can provide high performance and scalability for training large models. SSHing into the TPU VM allows the user to run and debug the TensorFlow code directly on the TPU device, without any network overhead or data transfer issues. References:

* 1: TPU VMs Overview
* 2: TPU VMs Quickstart
* 3: Debugging TensorFlow Models on Cloud TPUs

NO.23 You recently used BigQuery ML to train an AutoML regression model. You shared results with your team and received positive feedback. You need to deploy your model for online prediction as quickly as possible. What should you do?

A. Retrain the model by using BigQuery ML. and specify Vertex AI as the model registry Deploy the model from Vertex AI Model Registry to a Vertex AI endpoint.

B. Retrain the model by using Vertex AI Deploy the model from Vertex AI Model Registry to a Vertex AI endpoint.

C. Alter the model by using BigQuery ML and specify Vertex AI as the model registry Deploy the model from Vertex AI Model Registry to a Vertex AI endpoint.

D. Export the model from BigQuery ML to Cloud Storage Import the model into Vertex AI Model Registry Deploy the model to a Vertex AI endpoint.

Answer: A

Explanation:

BigQuery ML is a service that allows you to create and train ML models using SQL queries. You can use BigQuery ML to train an AutoML regression model, which is a type of model that automatically selects the best features and architecture for your data. You can also specify Vertex AI as the model registry, which is a service that allows you to store and manage your ML models. By using Vertex AI as the model registry, you can easily deploy your model to a Vertex AI endpoint, which is a service that allows you to serve your ML models online and scale them automatically. By using BigQuery ML, Vertex AI model registry, and Vertex AI endpoint, you can deploy your model for online prediction as quickly as possible, without having to export, import, or retrain your model. References:

* BigQuery ML documentation
* Vertex AI documentation
* Preparing for Google Cloud Certification: Machine Learning Engineer Professional Certificate

NO.24 You are developing an ML model to predict house prices. While preparing the data, you discover that an important predictor variable, distance from the closest school, is often missing and does not have high variance. Every instance (row) in your data is important. How should you handle

the missing data?

A. Delete the rows that have missing values.

B. Apply feature crossing with another column that does not have missing values.

C. Predict the missing values using linear regression.

D. Replace the missing values with zeros.

Answer: C

Explanation:

The best option for handling missing data in this case is to predict the missing values using linear regression.

Linear regression is a supervised learning technique that can be used to estimate the relationship between a continuous target variable and one or more predictor variables. In this case, the target variable is the distance from the closest school, and the predictor variables are the other features in the dataset, such as house size, location, number of rooms, etc. By fitting a linear regression model on the data that has no missing values, we can then use the model to predict the missing values for the distance from the closest school feature. This way, we can preserve all the instances in the dataset and avoid introducing bias or reducing variance. The other options are not suitable for handling missing data in this case, because:

* Deleting the rows that have missing values would reduce the size of the dataset and potentially lose important information. Since every instance is important, we want to keep as much data as possible.
* Applying feature crossing with another column that does not have missing values would create a new feature that combines the values of two existing features. This might increase the complexity of the model and introduce noise or multicollinearity. It would not solve the problem of missing values, as the new feature would still have missing values whenever the distance from the closest school feature is missing.
* Replacing the missing values with zeros would distort the distribution of the feature and introduce bias.

It would also imply that the houses with missing values are located at the same distance from the closest school, which is unlikely to be true. A zero value might also be outside the range of the feature, as the distance from the closest school is unlikely to be exactly zero for any house.

References:

* Linear Regression
* Imputation of missing values
* Google Cloud launches machine learning engineer certification
* Google Professional Machine Learning Engineer Certification
* Professional ML Engineer Exam Guide
* Preparing for Google Cloud Certification: Machine Learning Engineer Professional Certificate

NO.25 You are deploying a new version of a model to a production Vertex AI endpoint that is serving traffic You plan to direct all user traffic to the new model You need to deploy the model with minimal disruption to your application What should you do?

A. 1 Create a new endpoint.
2 Create a new model Set it as the default version Upload the model to Vertex AI Model Registry.
3. Deploy the new model to the new endpoint.
4 Update Cloud DNS to point to the new endpoint

B. 1. Create a new endpoint.

2. Create a new model Set the parentModel parameter to the model ID of the currently deployed model and set it as the default version Upload the model to Vertex AI Model Registry
3. Deploy the new model to the new endpoint and set the new model to 100% of the traffic
C. 1 Create a new model Set the parentModel parameter to the model ID of the currently deployed model Upload the model to Vertex AI Model Registry.
2 Deploy the new model to the existing endpoint and set the new model to 100% of the traffic.
D. 1, Create a new model Set it as the default version Upload the model to Vertex AI Model Registry
2 Deploy the new model to the existing endpoint
Answer: C
Explanation:
The best option for deploying a new version of a model to a production Vertex AI endpoint that is serving traffic, directing all user traffic to the new model, and deploying the model with minimal disruption to your application, is to create a new model, set the parentModel parameter to the model ID of the currently deployed model, upload the model to Vertex AI Model Registry, deploy the new model to the existing endpoint, and set the new model to 100% of the traffic. This option allows you to leverage the power and simplicity of Vertex AI to update your model version and serve online predictions with low latency. Vertex AI is a unified platform for building and deploying machine learning solutions on Google Cloud. Vertex AI can deploy a trained model to an online prediction endpoint, which can provide low-latency predictions for individual instances. A model is a resource that represents a machine learning model that you can use for prediction. A model can have one or more versions, which are different implementations of the same model. A model version can have different parameters, code, or data than another version of the same model. A model version can help you experiment and iterate on your model, and improve the model performance and accuracy. A parentModel parameter is a parameter that specifies the model ID of the model that the new model version is based on. A parentModel parameter can help you inherit the settings and metadata of the existing model, and avoid duplicating the model configuration. Vertex AI Model Registry is a service that can store and manage your machine learning models on Google Cloud. Vertex AI Model Registry can help you upload and organize your models, and track the model versions and metadata. An endpoint is a resource that provides the service endpoint (URL) you use to request the prediction. An endpoint can have one or more deployed models, which are instances of model versions that are associated with physical resources. A deployed model can help you serve online predictions with low latency, and scale up or down based on the traffic. By creating a new model, setting the parentModel parameter to the model ID of the currently deployed model, uploading the model to Vertex AI Model Registry, deploying the new model to the existing endpoint, and setting the new model to 100% of the traffic, you can deploy a new version of a model to a production Vertex AI endpoint that is serving traffic, direct all user traffic to the new model, and deploy the model with minimal disruption to your application1.
The other options are not as good as option C, for the following reasons:
* Option A: Creating a new endpoint, creating a new model, setting it as the default version, uploading the model to Vertex AI Model Registry, deploying the new model to the new endpoint, and updating Cloud DNS to point to the new endpoint would require more skills and steps than creating a new model, setting the parentModel parameter to the model ID of the currently deployed model, uploading the model to Vertex AI Model Registry, deploying the new model to the existing endpoint, and setting the new model to 100% of the traffic. Cloud DNS is a service that can provide reliable and scalable Domain Name System (DNS) services on Google Cloud. Cloud DNS can help you manage your DNS records, and resolve domain names to IP addresses. By updating Cloud DNS to point to the new

endpoint, you can redirect the user traffic to the new endpoint, and avoid breaking the existing application. However, creating a new endpoint, creating a new model, setting it as the default version, uploading the model to Vertex AI Model Registry, deploying the new model to the new endpoint, and updating Cloud DNS to point to the new endpoint would require more skills and steps than creating a new model, setting the parentModel parameter to the model ID of the currently deployed model, uploading the model to Vertex AI Model Registry, deploying the new model to the existing endpoint, and setting the new model to
100% of the traffic. You would need to write code, create and configure the new endpoint, create and configure the new model, upload the model to Vertex AI Model Registry, deploy the model to the new endpoint, and update Cloud DNS to point to the new endpoint. Moreover, this option would create a new endpoint, which can increase the maintenance and management costs2.
* Option B: Creating a new endpoint, creating a new model, setting the parentModel parameter to the model ID of the currently deployed model and setting it as the default version, uploading the model to Vertex AI Model Registry, and deploying the new model to the new endpoint and setting the new model
* to 100% of the traffic would require more skills and steps than creating a new model, setting the parentModel parameter to the model ID of the currently deployed model, uploading the model to Vertex AI Model Registry, deploying the new model to the existing endpoint, and setting the new model to
100% of the traffic. A parentModel parameter is a parameter that specifies the model ID of the model that the new model version is based on. A parentModel parameter can help you inherit the settings and metadata of the existing model, and avoid duplicating the model configuration. A default version is a model version that is used for prediction when no other version is specified. A default version can help you simplify the prediction request, and avoid specifying the model version every time. By setting the parentModel parameter to the model ID of the currently deployed model and setting it as the default version, you can create a new model that is based on the existing model, and use it for prediction without specifying the model version. However, creating a new endpoint, creating a new model, setting the parentModel parameter to the model ID of the currently deployed model and setting it as the default version, uploading the model to Vertex AI Model Registry, and deploying the new model to the new endpoint and setting the new model to 100% of the traffic would require more skills and steps than creating a new model, setting the parentModel parameter to the model ID of the currently deployed model, uploading the model to Vertex AI Model Registry, deploying the new model to the existing endpoint, and setting the new model to 100% of the traffic. You would need to write code, create and configure the new endpoint, create and configure the new model, upload the model to Vertex AI Model Registry, and deploy the model to the new endpoint. Moreover, this option would create a new endpoint, which can increase the maintenance and management costs2.
* Option D: Creating a new model, setting it as the default version, uploading the model to Vertex AI Model Registry, and deploying the new model to the existing endpoint would not allow you to inherit the settings and metadata of the existing model, and could cause errors or poor performance. A default version is a model version that is used for prediction when no other version is specified. A default version can help you simplify the prediction request, and avoid specifying the model version every time.
By setting the new model as the default version, you can use the new model for prediction without specifying the model version. However, creating a new model, setting it as the default version, uploading the model to Vertex AI Model Registry, and deploying the new model to the existing

endpoint would not allow you to inherit the settings and metadata of the existing model, and could cause errors or poor performance. You would need to write code, create and configure the new model, upload the model to Vertex AI Model Registry, and deploy the model to the existing endpoint. Moreover, this option would not set the parentModel parameter to the model ID of the currently deployed model, which could prevent you from inheriting the settings and metadata of the existing model, and cause inconsistencies or conflicts between the model versions2.
References:
* Preparing for Google Cloud Certification: Machine Learning Engineer, Course 3: Production ML Systems, Week 2: Serving ML Predictions
* Google Cloud Professional Machine Learning Engineer Exam Guide, Section 3: Scaling ML models in production, 3.1 Deploying ML models to production
* Official Google Cloud Certified Professional Machine Learning Engineer Study Guide, Chapter 6: Production ML Systems, Section 6.2: Serving ML Predictions
* Vertex AI
* Cloud DNS

NO.26 You have deployed a scikit-learn model to a Vertex AI endpoint using a custom model server. You enabled auto scaling; however, the deployed model fails to scale beyond one replica, which led to dropped requests.
You notice that CPU utilization remains low even during periods of high load. What should you do?
A. Attach a GPU to the prediction nodes.
B. Increase the number of workers in your model server.
C. Schedule scaling of the nodes to match expected demand.
D. Increase the minReplicaCount in your DeployedModel configuration.
Answer: B
Explanation:
Auto scaling is a feature that allows you to automatically adjust the number of prediction nodes based on the traffic and load of your deployed model1. However, auto scaling depends on the CPU utilization of your prediction nodes, which is the percentage of CPU resources used by your model server1. If your CPU utilization is low, even during periods of high load, it means that your model server is not fully utilizing the available CPU resources, and thus auto scaling will not trigger more replicas2.
One possible reason for low CPU utilization is that your model server is using a single worker process to handle prediction requests3. A worker process is a subprocess that runs your model code and handles prediction requests3. If you have only one worker process, it can only handle one request at a time, which can lead to dropped requests when the traffic is high3. To increase the CPU utilization and the throughput of your model server, you can increase the number of worker processes, which will allow your model server to handle multiple requests in parallel3.
To increase the number of workers in your model server, you need to modify your custom model server code and use the --workers flag to specify the number of worker processes you want to use3. For example, if you are using a Gunicorn server, you can use the following command to start your model server with four worker processes:
gunicorn --bind :$PORT --workers 4 --threads 1 --timeout 60 main:app
By increasing the number of workers in your model server, you can increase the CPU utilization of your prediction nodes, and thus enable auto scaling to scale beyond one replica.
The other options are not suitable for your scenario, because they either do not address the root

cause of low CPU utilization, such as attaching a GPU or scheduling scaling, or they do not enable auto scaling, such as increasing the minReplicaCount, which is a fixed number of nodes that will always run regardless of the traffic1.
References:
* Scaling prediction nodes | Vertex AI | Google Cloud
* Troubleshooting | Vertex AI | Google Cloud
* Using a custom prediction routine with online prediction | Vertex AI | Google Cloud

NO.27 You need to build classification workflows over several structured datasets currently stored in BigQuery.
Because you will be performing the classification several times, you want to complete the following steps without writing code: exploratory data analysis, feature selection, model building, training, and hyperparameter tuning and serving. What should you do?
A. Configure AutoML Tables to perform the classification task
B. Run a BigQuery ML task to perform logistic regression for the classification
C. Use AI Platform Notebooks to run the classification model with pandas library
D. Use AI Platform to run the classification model job configured for hyperparameter tuning
Answer: A
Explanation:
AutoML Tables is a service that allows you to automatically build and deploy state-of-the-art machine learning models on structured data without writing code. You can use AutoML Tables to perform the following steps for the classification task:
* Exploratory data analysis: AutoML Tables provides a graphical user interface (GUI) and a command-line interface (CLI) to explore your data, visualize statistics, and identify potential issues.
* Feature selection: AutoML Tables automatically selects the most relevant features for your model based on the data schema and the target column. You can also manually exclude or include features, or create new features from existing ones using feature engineering.
* Model building: AutoML Tables automatically builds and evaluates multiple machine learning models using different algorithms and architectures. You can also specify the optimization objective, the budget, and the evaluation metric for your model.
* Training and hyperparameter tuning: AutoML Tables automatically trains and tunes your model using the best practices and techniques from Google's research and engineering teams. You can monitor the training progress and the performance of your model on the GUI or the CLI.
* Serving: AutoML Tables automatically deploys your model to a fully managed, scalable, and secure environment. You can use the GUI or the CLI to request predictions from your model, either online
* (synchronously) or offline (asynchronously).
References:
* [AutoML Tables documentation]
* [AutoML Tables overview]
* [AutoML Tables how-to guides]