DUMPSBOSS.

AWS Certified Machine Learning Engineer - Associate

Amazon AWS MLA-C01

Version Demo

Total Demo Questions: 15

Total Premium Questions: 207

Buy Premium PDF

https://dumpsboss.co

support@dumpsboss.co

support@dumpsboss.co
dumpsboss.co

QUESTION NO: 1

A company uses a batching solution to process daily analytics. The company wants to provide near real-time updates, use open-source technology, and avoid managing or scaling infrastructure.

Which solution will meet these requirements?

A. Create Amazon Managed Streaming for Apache Kafka (Amazon MSK) Serverless clusters.

B. Create Amazon MSK Provisioned clusters.

C. Create Amazon Kinesis Data Streams with Application Auto Scaling.

D. Create self-hosted Apache Flink applications on Amazon EC2.

ANSWER: A

Explanation:

If you want near real-time updates, the usual move is to switch from batch processing to a streaming setup. The question also calls out open-source tech and “don’t make me manage servers or scaling,” which is basically what Amazon MSK Serverless is built for.

MSK Serverless gives you Apache Kafka (open source) without the headache of provisioning brokers, sizing them, or scaling the cluster as traffic changes. You still get Kafka-compatible tooling and semantics, but AWS handles the operational parts for you.

MSK Provisioned (option B) is still managed, but you’re on the hook for choosing broker sizes, capacity planning, and scaling—so it doesn’t match the “avoid managing or scaling infrastructure” requirement. Kinesis Data Streams (option C) is great for streaming, but it’s AWS-native rather than Apache Kafka/open-source compatible in the same way. And running Flink on EC2 (option D) is the opposite of serverless—you’d manage instances, scaling, patching, and availability.

References: https://docs.aws.amazon.com/msk/latest/developerguide/msk-serverless.html and https://aws.amazon.com/msk/features/msk-serverless/

QUESTION NO: 2

An ML engineer is designing an AI-powered traffic management system. The system must use near real-time inference to predict congestion and prevent collisions.

The system must also use batch processing to perform historical analysis of predictions over several hours to improve the model. The inference endpoints must scale automatically to meet demand.

Which combination of solutions will meet these requirements? (Select TWO.)

A. Use Amazon SageMaker real-time inference endpoints with automatic scaling based on ConcurrentInvocationsPerInstance.

B. Use AWS Lambda with reserved concurrency and SnapStart to connect to SageMaker endpoints.

C. Use an Amazon SageMaker Processing job for batch historical analysis. Schedule the job with Amazon EventBridge.

D. Use Amazon EC2 Auto Scaling to host containers for batch analysis.

E. Use AWS Lambda for historical analysis.

ANSWER: A C

Explanation:

For the near real-time part, Amazon SageMaker real-time inference endpoints are the right fit because they're built for low-latency predictions and can automatically scale out when traffic spikes. You can tie autoscaling to metrics like invocations per instance, so the endpoint adds or removes capacity without you babysitting it. That directly matches the "near real-time" and "scale automatically" requirements.

For the multi-hour historical analysis, a SageMaker Processing job is a much better match than Lambda. Processing jobs are meant for batch workloads, can run for a long time, and are commonly used to analyze data, compute metrics, and prep datasets for model improvement. Scheduling them with Amazon EventBridge is a clean way to kick them off on a timer (or based on events) without running servers yourself.

Lambda isn't suitable for "several hours" of processing because of its runtime limits, and EC2 Auto Scaling for batch analysis would work but adds extra infrastructure work compared to a managed Processing job.

Refs: https://docs.aws.amazon.com/sagemaker/latest/dg/realtime-endpoints.html and
https://docs.aws.amazon.com/sagemaker/latest/dg/processing-job.html and
https://docs.aws.amazon.com/eventbridge/latest/userguide/eb-run-lambda-schedule.html

QUESTION NO: 3

An ML engineer is training an XGBoost regression model in Amazon SageMaker AI. The ML engineer conducts several rounds of hyperparameter tuning with random grid search. After these rounds of tuning, the error rate on the test hold-out dataset is much larger than the error rate on the training dataset.

The ML engineer needs to make changes before running the hyperparameter grid search again. Which changes will improve the model's performance? (Select TWO.)

A. Increase the model complexity by increasing the number of features in the dataset.

B. Decrease the model complexity by reducing the number of features in the dataset.

C. Decrease the model complexity by reducing the number of samples in the dataset.

D. Increase the value of the L2 regularization parameter.

E. Decrease the value of the L2 regularization parameter.

ANSWER: B D

Explanation:

This gap between training error (low) and test error (high) is the classic sign of overfitting. In plain terms, the model is "memorizing" patterns that only exist in the training data, so it looks great during training but falls apart when it sees new data.

One straightforward fix is to simplify what the model can learn. Reducing the number of features can remove noisy or weak signals that encourage the model to fit quirks in the training set. Fewer, better features usually means the model generalizes more cleanly to the hold-out test set.

Another strong lever in XGBoost is regularization. Increasing the L2 regularization term (lambda) adds a penalty for overly large weights, which discourages the model from becoming too “twitchy” and complex. That typically improves generalization and narrows the train-test gap. You can see these regularization parameters in the XGBoost docs: https://xgboost.readthedocs.io/en/stable/parameter.html and AWS’s guidance on XGBoost in SageMaker: https://docs.aws.amazon.com/sagemaker/latest/dg/xgboost.html

The other options go the wrong way: adding features or lowering regularization tends to make overfitting worse, and reducing the number of samples usually makes generalization harder, not easier.

QUESTION NO: 4

A company is exploring generative AI and wants to add a new product feature. An ML engineer is making API calls from existing Amazon EC2 instances to Amazon Bedrock.

The EC2 instances are in a private subnet and must remain private during the implementation. The EC2 instances have a security group that allows access to all IP addresses in the private subnet.

What should the ML engineer do to establish a connection between the EC2 instances and Amazon Bedrock?

A. Modify the security group to allow inbound and outbound traffic to and from Amazon Bedrock.

B. Use AWS PrivateLink to access Amazon Bedrock through an interface VPC endpoint.

C. Configure Amazon Bedrock to use the private subnet where the EC2 instances are deployed.

D. Use AWS Direct Connect to link the VPC to Amazon Bedrock.

ANSWER: B

Explanation:

Because the EC2 instances sit in a private subnet, you don’t want them reaching Bedrock over the public internet (even if you locked down security groups). The clean way to keep traffic private is to create an **interface VPC endpoint** using **AWS PrivateLink**, so calls to Amazon Bedrock stay on the AWS network end-to-end.

Just tweaking the security group (Option A) doesn’t magically create a private path to Bedrock—you’d still need a route out (like a NAT gateway) and you’d still be using public endpoints. Option C isn’t possible because Bedrock is a managed service and you can’t “place” it inside your subnet. Option D (Direct Connect) is mainly for connecting on-premises networks to AWS, not for private access from an EC2 private subnet to an AWS managed service.

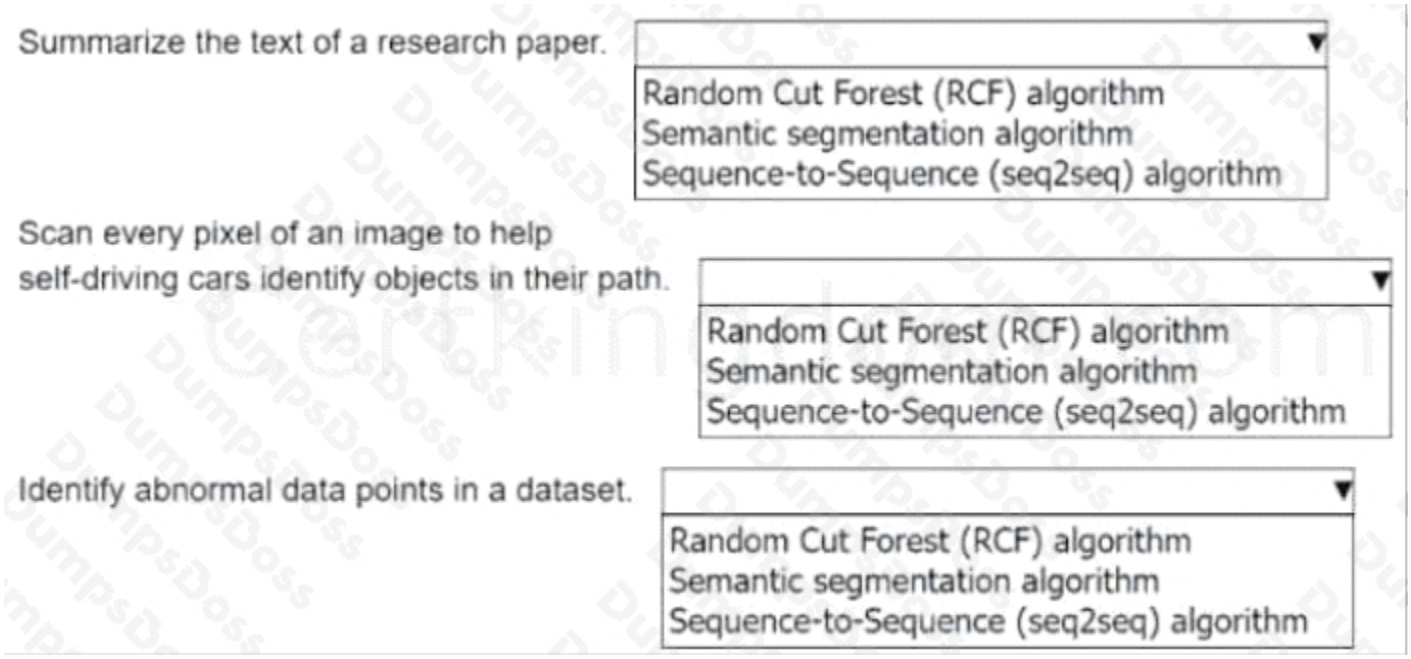
So the practical, AWS-recommended move is PrivateLink with an interface endpoint for Bedrock. References: https://docs.aws.amazon.com/bedrock/latest/userguide/vpc-interface-endpoints.html and https://docs.aws.amazon.com/vpc/latest/privatelink/what-is-privatelink.html

QUESTION NO: 5 - (HOTSPOT)

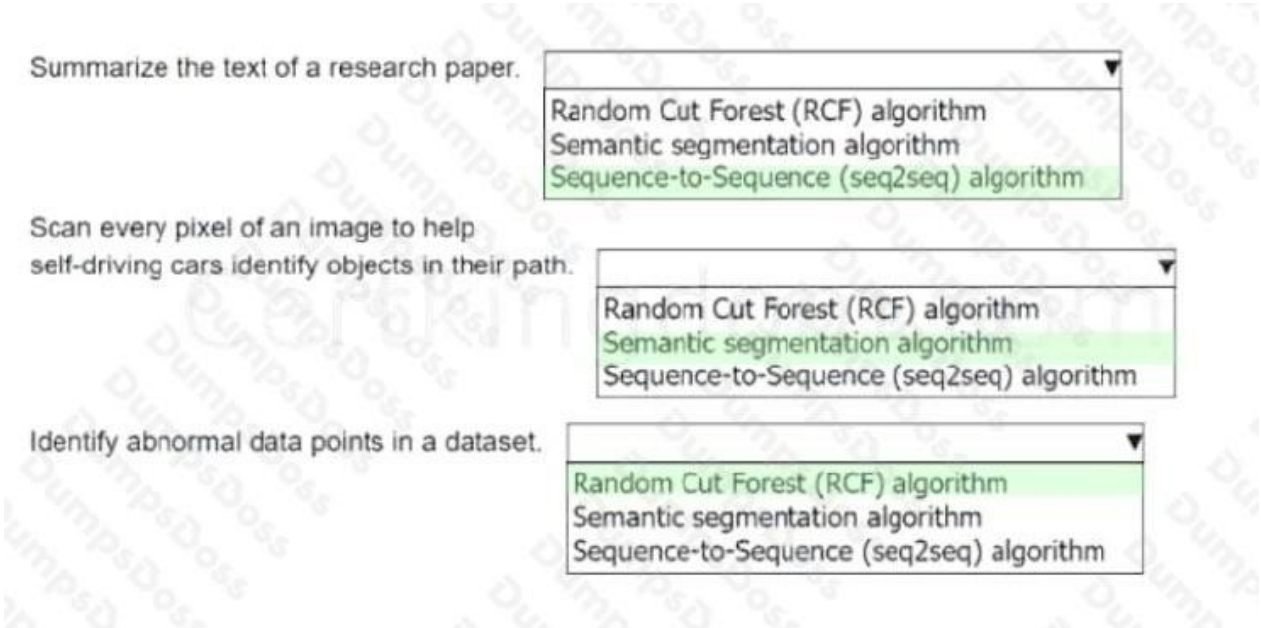
An ML engineer must choose the appropriate Amazon SageMaker algorithm to solve specific AI problems.

Select the correct SageMaker built-in algorithm from the following list for each use case. Each algorithm should be selected one time.

Random Cut Forest (RCF) algorithm Semantic segmentation algorithm Sequence-to-Sequence (seq2seq) algorithm



ANSWER:



Explanation:

The correct way to think about this question is to match each use case to what the SageMaker built-in algorithm is designed to do. For **text summarization** ("Summarize the text of a research paper"), you want a model that takes an input sequence of tokens (the paper text) and produces an output sequence (the summary). That's exactly the classic encoder-decoder

pattern, which is why the **Sequence-to-Sequence (seq2seq)** built-in algorithm is the right fit. SageMaker's seq2seq is commonly used for tasks like translation and summarization where the output is another piece of text rather than a class label. See: SageMaker seq2seq documentation.

For the self-driving car scenario ("Scan every pixel of an image to help self-driving cars identify objects in their path"), the key phrase is **every pixel**. That implies you're not just classifying the whole image, and you're not just drawing bounding boxes; you're assigning a label to each pixel (road, car, pedestrian, sidewalk, etc.). That is the definition of **semantic segmentation**, so the **Semantic segmentation** built-in algorithm is the correct selection. Reference: SageMaker Semantic Segmentation documentation.

Finally, "Identify abnormal data points in a dataset" is an **anomaly detection** problem, typically unsupervised (you often don't have labels for anomalies). SageMaker's **Random Cut Forest (RCF)** is purpose-built for detecting outliers in numeric/tabular data by isolating points that are easier to separate from the rest of the distribution. That makes RCF the best match for abnormal data point detection. Reference: SageMaker Random Cut Forest documentation.

QUESTION NO: 6

An ML engineer is training an ML model to identify medical patients for disease screening. The tabular dataset for training contains 50,000 patient records: 1,000 with the disease and 49,000 without the disease.

The ML engineer splits the dataset into a training dataset, a validation dataset, and a test dataset. What should the ML engineer do to transform the data and make the data suitable for training?

A. Apply principal component analysis (PCA) to oversample the minority class in the training dataset.

C. Randomly oversample the majority class in the validation dataset.

D. Apply k-means clustering to undersample the minority class in the test dataset.

ANSWER: B

Explanation:

This dataset is heavily imbalanced (about 2% positive cases). If you train as-is, the model can get "good" accuracy by mostly predicting "no disease," but it won't learn the patterns that actually matter for catching sick patients.

The usual fix is to rebalance only the *training* set, and keep validation/test sets in the original real-world distribution so your evaluation stays honest. SMOTE is a common approach for tabular data: it creates new minority-class examples by interpolating between existing positive cases, which helps the model see more disease-like patterns without just duplicating rows.

PCA doesn't oversample anything—it's for dimensionality reduction. Oversampling the majority class makes the imbalance worse, and changing the test set (or undersampling the minority there) breaks your ability to measure real performance.

References: https://imbalanced-learn.org/stable/references/generated/imblearn.over_sampling.SMOTE.html and https://docs.aws.amazon.com/sagemaker/latest/dg/autopilot-data-exploration.html

QUESTION NO: 7

An ML engineer is setting up an Amazon SageMaker AI pipeline for an ML model. The pipeline must automatically initiate a re-training job if any data drift is detected.

How should the ML engineer set up the pipeline to meet this requirement?

A. Use an AWS Glue crawler and an AWS Glue extract, transform, and load (ETL) job to detect data drift. Use AWS Glue triggers to automate the retraining job.

B. Use Amazon Managed Service for Apache Flink to detect data drift. Use an AWS Lambda function to automate the re-training job.

C. Use SageMaker Model Monitor to detect data drift. Use an AWS Lambda function to automate the re-training job.

D. Use Amazon Quick Suite (previously known as Amazon QuickSight) anomaly detection to detect data drift. Use an AWS Step Functions workflow to automate the re-training job.

ANSWER: C

Explanation:

The clean AWS-native way to do drift-based retraining is to use SageMaker Model Monitor. Model Monitor can continuously compare live inference data to a baseline (usually captured from your training data) and flag things like feature distribution drift, missing values, or schema changes. That's exactly what "data drift detected" means in production ML.

Once Model Monitor detects drift, it publishes results and metrics that you can watch in Amazon CloudWatch. From there, you can trigger an AWS Lambda function (for example via a CloudWatch alarm or EventBridge rule) to kick off retraining—either by starting a new SageMaker training job directly or by starting a SageMaker Pipelines execution that runs your full retrain-and-deploy workflow.

The other options don't really fit. AWS Glue is great for ETL and cataloging, but it doesn't do ML drift detection. Managed Flink is stream processing; you'd have to build your own drift logic. QuickSight anomaly detection is aimed at BI dashboards, not monitoring ML input features.

References: https://docs.aws.amazon.com/sagemaker/latest/dg/model-monitor.html and https://docs.aws.amazon.com/sagemaker/latest/dg/pipelines.html

QUESTION NO: 8

A company's ML engineer has deployed an ML model for sentiment analysis to an Amazon SageMaker endpoint. The ML engineer needs to explain to company stakeholders how the model makes predictions.

Which solution will provide an explanation for the model's predictions?

A. Use SageMaker Model Monitor on the deployed model.

B. Use SageMaker Clarify on the deployed model.

C. Show the distribution of inferences from A/Đ' testing in Amazon CloudWatch.

D. Add a shadow endpoint. Analyze prediction differences on samples.

ANSWER: B

Explanation:

To explain *why* the sentiment model predicted "positive" or "negative," you need an explainability tool that can break down how each input feature influenced the output. Amazon SageMaker Clarify is built exactly for that: it can generate feature

attribution (for example, showing which words or features pushed the prediction toward a certain sentiment) and produce reports you can share with non-technical stakeholders.

The other choices are useful, but they don't really answer the "how did the model decide?" question. SageMaker Model Monitor focuses on drift and data quality over time, not explanations. A/B testing metrics in CloudWatch and shadow endpoints help compare model versions safely, but they still don't tell you which inputs drove a single prediction.

Reference: https://docs.aws.amazon.com/sagemaker/latest/dg/clarify.html

QUESTION NO: 9

A company launches a feature that predicts home prices. An ML engineer trained a regression model using the SageMaker AI XGBoost algorithm. The model performs well on training data but underperforms on real-world validation data.

Which solution will improve the validation score with the LEAST implementation effort?

A. Create a larger training dataset with more real-world data and retrain.

B. Increase the num_round hyperparameter.

C. Change the eval_metric from RMSE to Error.

D. Increase the lambda hyperparameter.

ANSWER: D

Explanation:

This is a classic overfitting situation: the model looks great on training data but doesn't generalize to validation (real-world) data. With XGBoost, one of the quickest, lowest-effort fixes is to add more regularization so the model doesn't "memorize" the training set.

In SageMaker's XGBoost, `lambda` is the L2 regularization term. Increasing it usually makes the model a bit more conservative (smaller weights, simpler fits), which often improves validation performance without changing your data pipeline or model architecture. It's basically a one-line hyperparameter tweak and rerun.

The other options are either likely to make overfitting worse (`num_round` often increases model complexity), don't actually change learning behavior (switching `eval_metric` only changes how you report performance), or require much more work (collecting/curating more real-world data).

References: https://docs.aws.amazon.com/sagemaker/latest/dg/xgboost.html and https://xgboost.readthedocs.io/en/stable/parameter.html

QUESTION NO: 10 - (HOTSPOT)

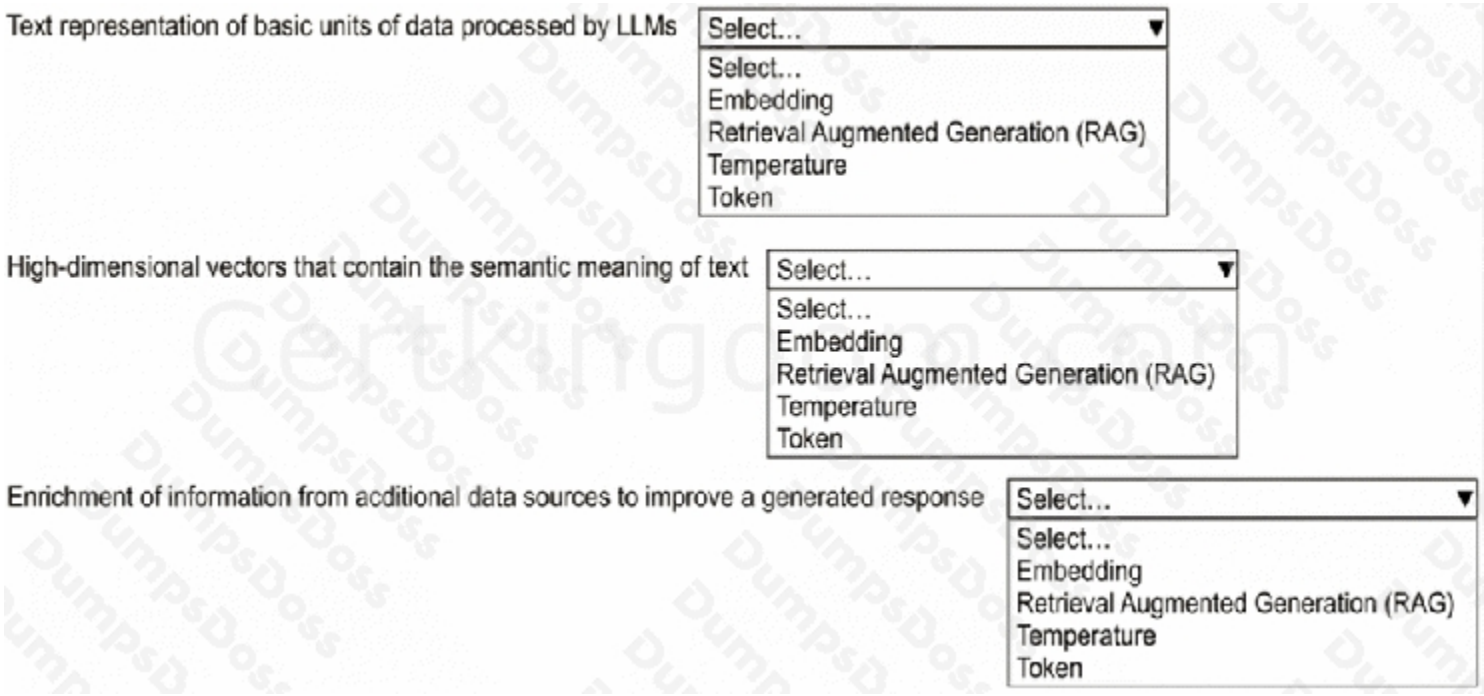
An ML engineer is building a generative AI application on Amazon Bedrock by using large language models (LLMs).

Select the correct generative AI term from the following list for each description. Each term should be selected one time or not at all. (Select three.)

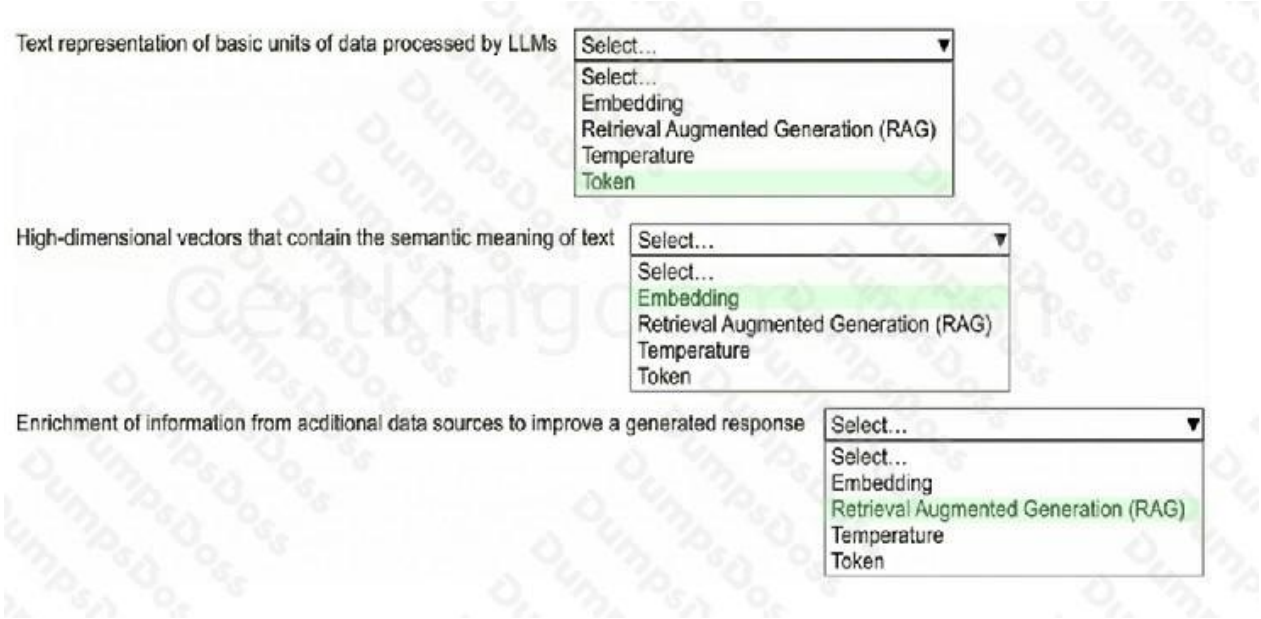
Embedding

Retrieval Augmented Generation (RAG) Temperature

Token



ANSWER:



Explanation:

The three dropdown selections shown in the answered image line up with the standard generative AI definitions you'd use when building on Amazon Bedrock.

1) "Text representation of basic units of data processed by LLMs" → Token
A token is the basic unit an LLM reads and generates (often a word, part of a word, or punctuation). This is why model limits

and pricing are commonly expressed in “input tokens” and “output tokens.” If you pick anything other than *Token* here, you’d be mixing up the model’s fundamental text unit with higher-level concepts like embeddings or retrieval.

2) “High-dimensional vectors that contain the semantic meaning of text” → Embedding
An embedding is a numeric vector representation of text (or other data) that captures semantic similarity. Embeddings are what you store in a vector database or vector index so you can do similarity search (for example, finding the most relevant passages to a user question). This matches the description precisely: “high-dimensional vectors” that represent meaning.

3) “Enrichment of information from additional data sources to improve a generated response” → Retrieval Augmented Generation (RAG)
RAG is the pattern where you retrieve relevant context from an external source (documents, knowledge base, vector store) and then provide that context to the LLM so the response is grounded and more accurate. That’s exactly what “enrichment from additional data sources” is describing.

Notice that **Temperature** is not used, which is correct here. Temperature controls randomness/creativity in generation, not tokenization, vector meaning, or retrieval augmentation.

References: Amazon Bedrock User Guide, Amazon Bedrock Knowledge Bases (RAG), AWS guidance on RAG concepts.

QUESTION NO: 11

An ML engineer needs to use data with Amazon SageMaker Canvas to train an ML model. The data is

stored in Amazon S3 and is complex in structure. The ML engineer must use a file format that minimizes processing time for the data.

Which file format will meet these requirements?

A. CSV files compressed with Snappy

B. JSON objects in JSONL format

C. JSON files compressed with gzip

D. Apache Parquet files

ANSWER: D

Explanation:

For complex data sitting in S3, Apache Parquet is usually the fastest option because it’s a columnar format. That means tools can read only the columns they actually need instead of scanning the whole file every time. In practice, this cuts down disk I/O and speeds up loading and preprocessing, which is exactly what you want if you’re trying to minimize processing time in SageMaker Canvas.

CSV and JSON (even when compressed) are still row-based text formats. They tend to be slower to parse, heavier to scan, and generally less efficient for analytics-style reads—especially as datasets get wider (more columns) and more nested/complex. Compression helps storage and transfer, but it doesn’t fix the fundamental “read everything and parse text” overhead.

Parquet is also a common, well-supported format across AWS analytics and ML services, and it works cleanly with S3-based workflows. References: https://parquet.apache.org/docs/overview/ and https://docs.aws.amazon.com/sagemaker/latest/dg/canvas-data-import.html

QUESTION NO: 12

A company is developing an application that reads animal descriptions from user prompts and generates images based on the information in the prompts. The application reads a message from an Amazon Simple Queue Service (Amazon SQS) queue. Then the application uses Amazon Titan Image Generator on Amazon Bedrock to generate an image based on the information in the message.

Finally, the application removes the message from SQS queue.

Which IAM permissions should the company assign to the application's IAM role? (Select TWO.)

A. Allow the bedrock:InvokeModel action for the Amazon Titan Image Generator resource.

B. Allow the bedrock:Get* action for the Amazon Titan Image Generator resource.

C. Allow the sqs:ReceiveMessage action and the sqs:DeleteMessage action for the SQS queue resource.

D. Allow the sqs:GetQueueAttributes action and the sqs:DeleteMessage action for the SQS queue resource.

E. Allow the sagemaker:PutRecord* action for the Amazon Titan Image Generator resource.

ANSWER: A C

Explanation:

The app has two jobs: pull work from SQS and call Bedrock to generate the image. To actually run inference with Titan Image Generator in Bedrock, the role needs `bedrock:InvokeModel`. Read-only permissions like `bedrock:Get*` won't generate anything—they're for viewing metadata, not invoking the model.

On the SQS side, the workflow is "receive a message, process it, then delete it." That means the role must have `sqs:ReceiveMessage` to read messages and `sqs:DeleteMessage` to remove them after success. `sqs:GetQueueAttributes` can be useful for visibility, but it doesn't let the app consume messages by itself.

So the right pair is: permission to invoke the Bedrock model, and permission to receive and delete SQS messages.
References: https://docs.aws.amazon.com/bedrock/latest/userguide/security_iam_id-based-policy-examples.html and https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/sqs-api-permissions-reference.html

QUESTION NO: 13

An ML engineer is training a simple neural network model. The models performance improves initially and then degrades after a certain number of epochs.

Which solutions will mitigate this problem? (Select TWO.)

A. Enable early stopping on the model.

B. Increase dropout in the layers.

C. Increase the number of layers.

D. Increase the number of neurons.

E. Investigate and reduce the sources of model bias.

ANSWER: A B

Explanation:

This pattern—getting better for a while and then getting worse after more epochs—is a classic sign of overfitting. The model starts off learning general patterns, but after enough training it begins to "memorize" the training data, and validation/test performance drops.

Early stopping helps by watching a validation metric and stopping training once that metric stops improving. It's basically a guardrail that prevents the model from training past the point where it generalizes well. This is a common, practical regularization approach in deep learning workflows.

Increasing dropout also fights overfitting. Dropout randomly turns off a fraction of neurons during training, which forces the network to spread learning across many paths instead of relying on a few "lucky" neurons. That usually improves generalization and makes performance more stable across epochs.

On the other hand, adding more layers or more neurons increases model capacity, which typically makes overfitting easier, not harder. And "model bias" here isn't the issue—this is about training dynamics over epochs, not systematic underfitting or fairness concerns.

References: https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/EarlyStopping and https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dropout

QUESTION NO: 14

A company wants to improve the sustainability of its ML operations.

Which actions will reduce the energy usage and computational resources that are associated with the company's training jobs? (Choose two.)

A. Use Amazon SageMaker Debugger to stop training jobs when non-converging conditions are detected.

B. Use Amazon SageMaker Ground Truth for data labeling.

C. Deploy models by using AWS Lambda functions.

D. Use AWS Trainium instances for training.

E. Use PyTorch or TensorFlow with the distributed training option.

ANSWER: A D

Explanation:

Option A helps because SageMaker Debugger can spot training runs that are going nowhere (for example, loss isn't improving, gradients blow up, or the job is basically stuck). Catching that early and stopping the run avoids wasting GPU/CPU hours, which directly cuts power use and compute spend. It's one of the simplest "stop burning cycles" wins.

Option D is also a solid sustainability move. AWS Trainium instances (Trn1/Trn2) are built specifically for training deep learning models efficiently, so you typically get more training work done per watt compared with many general-purpose or older GPU setups. If you can use supported frameworks, switching instance families can reduce the energy and resources needed for the same training outcome.

Ground Truth (B) is about labeling, not training efficiency. Lambda (C) is for inference/automation, not training jobs. Distributed training (E) can speed things up, but it can also increase total resources depending on scaling and communication overhead—so it's not a guaranteed energy reducer.

References: https://docs.aws.amazon.com/sagemaker/latest/dg/train-debugger.html and https://aws.amazon.com/machine-learning/trainium/

QUESTION NO: 15

An ML engineer has developed a binary classification model outside of Amazon SageMaker. The ML engineer needs to make the model accessible to a SageMaker Canvas user for additional tuning.

The model artifacts are stored in an Amazon S3 bucket. The ML engineer and the Canvas user are part of the same SageMaker domain.

Which combination of requirements must be met so that the ML engineer can share the model with the Canvas user? (Choose two.)

A. The ML engineer and the Canvas user must be in separate SageMaker domains.

B. The Canvas user must have permissions to access the S3 bucket where the model artifacts are stored.

C. The model must be registered in the SageMaker Model Registry.

D. The ML engineer must host the model on AWS Marketplace.

E. The ML engineer must deploy the model to a SageMaker endpoint.

ANSWER: B C

Explanation:

First, the Canvas user needs to be able to actually read the model files. Since the artifacts live in Amazon S3, that means the user (or the execution role Canvas uses on their behalf) must have permission to access that bucket and the specific object path. Without S3 access, Canvas can't import anything, even if the model is "shared" in other ways.

Second, registering the model in the SageMaker Model Registry is the clean way to make a model discoverable and manageable inside the SageMaker ecosystem. Once it's in the registry, it can be shared and governed more easily within the same SageMaker domain, which is exactly what you want when handing it off to a Canvas user for further work.

You don't need a separate domain, Marketplace hosting, or an endpoint just to share and tune the model—those are for different use cases like distribution or real-time inference.

References: https://docs.aws.amazon.com/sagemaker/latest/dg/model-registry.html and https://docs.aws.amazon.com/AmazonS3/latest/userguide/s3-access-control.html