DUMPSARENA

NVIDIA AI Infrastructure and Operations

NVIDIA NCA-AIIO

Version Demo

Total Demo Questions: 10

Total Premium Questions: 197

Buy Premium PDF

https://dumpsarena.co

sales@dumpsarena.co

sales@dumpsarena.co
dumpsarena.co

QUESTION NO: 1

You are working with a large healthcare dataset containing millions of patient records. Your goal is to identify patterns and extract actionable insights that could improve patient outcomes. The dataset is highly dimensional, with numerous variables, and requires significant processing power to analyze effectively. Which two techniques are most suitable for extracting meaningful insights from this large, complex dataset? (Select two)

A. SMOTE (Synthetic Minority Over-sampling Technique)

B. Data Augmentation

C. Batch Normalization

D. K-means Clustering

E. Dimensionality Reduction (e.g., PCA)

ANSWER: D E

Explanation:

For a large, highly dimensional healthcare dataset, the most suitable techniques for extracting meaningful insights are (1) clustering to discover natural groupings and (2) dimensionality reduction to simplify the feature space while preserving structure. **K-means clustering** is a common unsupervised learning method that groups similar patient records (e.g., cohorts with similar lab profiles, comorbidities, or utilization patterns). These clusters can reveal actionable subpopulations for care pathways or risk stratification. **Dimensionality reduction** (such as PCA) is well-suited to high-dimensional data because it compresses many correlated variables into fewer components, improving downstream visualization, noise reduction, and the performance/interpretability of subsequent models. Both are also directly supported and GPU-accelerated in NVIDIA RAPIDS cuML, aligning with the “significant processing power” requirement.

SMOTE is primarily for addressing class imbalance in supervised learning, not for general pattern discovery. **Data augmentation** is typically used to expand training data (often in vision/audio) rather than extract insights from tabular patient records. **Batch normalization** is a neural-network training stabilization technique, not an exploratory analytics method.

References: RAPIDS cuML Documentation, scikit-learn Clustering Overview

QUESTION NO: 2

Your organization runs multiple AI workloads on a shared NVIDIA GPU cluster. Some workloads are more critical than others. Recently, you've noticed that less critical workloads are consuming more GPU resources, affecting the performance of critical workloads. What is the best approach to ensure that critical workloads have priority access to GPU resources?

A. Implement GPU Quotas with Kubernetes Resource Management

B. Use CPU-based Inference for Less Critical Workloads

C. Upgrade the GPUs in the Cluster to More Powerful Models

D. Implement Model Optimization Techniques

ANSWER: A

Explanation:

The best approach is to enforce scheduling and resource governance at the cluster orchestrator level—i.e., use Kubernetes resource management to ensure critical workloads get scheduled first and are protected from lower-priority contention. In practice, this means requesting GPUs via the NVIDIA device plugin (e.g., `nvidia.com/gpu`) and combining that with Kubernetes controls such as namespaces/ResourceQuotas (to prevent teams or low-importance namespaces from consuming all GPUs) and PriorityClass (so higher-priority pods preempt or are scheduled ahead of lower-priority pods when the cluster is constrained). This is the most direct way to guarantee "priority access" in a shared GPU environment and aligns with how NVIDIA's GPU Operator integrates GPU enablement into Kubernetes while relying on Kubernetes' native scheduling and quota mechanisms for policy.

Option B (CPU inference) may reduce GPU pressure but doesn't guarantee priority; it's an application-level workaround with performance tradeoffs. Option C (upgrading GPUs) increases capacity but still allows noisy-neighbor behavior without policy controls. Option D (model optimization) can improve efficiency but likewise doesn't enforce fairness or priority. Governance (quotas + priority) is the correct control plane solution.

References: Kubernetes Resource Quotas, Pod Priority and Preemption

QUESTION NO: 3

After deploying an AI model on an NVIDIA T4 GPU in a production environment, you notice that the inference latency is inconsistent, varying significantly during different times of the day. Which of the following actions would most likely resolve the issue?

A. Increase the number of inference threads.

B. Upgrade the GPU driver.

C. Deploy the model on a CPU instead of a GPU.

D. Implement GPU isolation for the inference process.

ANSWER: D

Explanation:

The most likely cause of "time-of-day" latency variability in production is resource contention: other jobs/users/containers may be sharing the same GPU (or the host), leading to fluctuating GPU scheduling, memory bandwidth pressure, and PCIe/CPU interference. The most direct remediation is to isolate the inference workload so it has predictable access to GPU resources (e.g., dedicate the GPU to inference, enforce container/device isolation, and avoid co-locating competing GPU workloads). This aligns with NVIDIA best practices for production inference where consistent QoS is required.

Option A (increasing inference threads) can actually worsen jitter by increasing contention for CPU, GPU queues, and memory, especially if the system is already oversubscribed. Option B (upgrading the GPU driver) may fix bugs or improve compatibility, but it's not the most likely fix for diurnal latency swings caused by shared utilization patterns. Option C (moving to CPU) typically increases latency and does not inherently solve variability; it just shifts the bottleneck elsewhere.

Note: the question's original explanation mentions MIG, but NVIDIA T4 (Turing) does not support MIG; MIG is supported on NVIDIA Ampere/Hopper data center GPUs. Isolation is still the correct concept, just not via MIG on T4.

References: NVIDIA MIG User Guide (supported GPU architectures), NVIDIA Triton Inference Server User Guide (performance/QoS considerations)

QUESTION NO: 4

You are responsible for managing an AI infrastructure that includes multiple GPU clusters for deep learning workloads. One of your tasks is to efficiently allocate resources and manage workloads across these clusters using an orchestration platform. Which of the following approaches would best optimize the utilization of GPU resources while ensuring high availability of the AI workloads?

A. Use a round-robin scheduling algorithm across all GPU clusters

B. Assign workloads to clusters based on a predefined static schedule

C. Implement a load-balancing algorithm that dynamically assigns workloads based on real-time GPU availability

D. Use a first-come, first-served (FCFS) scheduling policy across all clusters

ANSWER: C

Explanation:

The best approach is to use dynamic, metrics-aware scheduling/load balancing that places (and, where supported, re-places) workloads based on real-time GPU availability and health. In practice, Kubernetes-based orchestration with NVIDIA GPU Operator/DCGM exposes GPU telemetry (utilization, memory, errors) so the scheduler and higher-level controllers can make placement decisions that avoid hot spots, improve bin-packing, and react to failures. This aligns with high availability goals because workloads can be steered away from unhealthy/overloaded nodes or clusters and rescheduled elsewhere, rather than being stuck to a fixed plan.

Round-robin (A) and FCFS (D) are blind to actual GPU capacity, fragmentation, and heterogeneous GPU types; they often lead to poor utilization and can overload a cluster while others sit idle. A static predefined schedule (B) similarly fails to adapt to changing demand, node drain/maintenance, or GPU failures, which is common in production AI environments. Dynamic load balancing (C) is the only option that explicitly incorporates real-time resource state, which is the key requirement for optimizing utilization while maintaining availability.

References: NVIDIA GPU Operator Documentation, NVIDIA DCGM Documentation

QUESTION NO: 5

An AI operations team is tasked with monitoring a large-scale AI infrastructure where multiple GPUs are utilized in parallel. To ensure optimal performance and early detection of issues, which two criteria are essential for monitoring the GPUs? (Select two)

A. GPU utilization percentage

B. Number of active CPU threads

C. Average CPU temperature

D. Memory bandwidth usage on GPUs

E. GPU fan noise levels

ANSWER: A D

Explanation:

For day-to-day GPU operations monitoring, you want metrics that directly reflect how busy the GPUs are and whether they're becoming a bottleneck for your workloads. **GPU utilization percentage** is essential because it shows how much time the GPU is actively executing work; sustained low utilization can indicate input pipeline/CPU bottlenecks, while sustained high utilization can indicate saturation and potential queuing. **Memory bandwidth usage on GPUs** (or related memory-throughput indicators) is also important because many AI workloads are memory-bound; high bandwidth pressure can explain poor scaling, performance regressions, or contention when multiple jobs share a GPU.

In contrast, **number of active CPU threads** and **average CPU temperature** are host-side metrics. They can be useful for overall node health, but they are not essential criteria for monitoring GPU performance specifically. **GPU fan noise levels** is not a standard operational metric and is not a reliable indicator of GPU health or performance; instead, you'd monitor GPU temperature, power, clocks, ECC errors, and throttling reasons.

NVIDIA's tooling (e.g., `nvidia-smi` / NVML) exposes utilization and memory-related metrics commonly used in monitoring stacks. See NVIDIA System Management Interface (nvidia-smi) and NVIDIA NVML API Documentation.

QUESTION NO: 6

You are responsible for managing an AI infrastructure that runs a critical deep learning application. The application experiences intermittent performance drops, especially when processing large datasets. Upon investigation, you find that some of the GPUs are not being fully utilized while others are overloaded, causing the overall system to underperform. What would be the most effective solution to address the uneven GPU utilization and optimize the performance of the deep learning application?

A. Reduce the size of the datasets being processed.

B. Increase the clock speed of the GPUs.

C. Add more GPUs to the system.

D. Implement dynamic load balancing for the GPUs.

ANSWER: D

Explanation:

The symptom described—some GPUs overloaded while others are underutilized—points to poor work distribution rather than a raw capacity or frequency problem. The most effective fix is to introduce dynamic load balancing so work is scheduled based on real-time GPU availability/queue depth. In practice, this can be done at the application layer (e.g., batching and distributing work across devices), at the orchestration layer (e.g., Kubernetes scheduling with appropriate GPU resource requests/limits and scaling), or via inference-serving frameworks that support dynamic batching and multiple model instances to spread load. This directly addresses the root cause: uneven utilization leading to stalls and throughput drops.

Reducing dataset size (A) is a workaround that can harm training/inference quality and does not correct imbalance. Increasing GPU clock speed (B) may marginally help the already-overloaded GPUs but won't improve the idle ones and can increase power/thermal throttling risk. Adding more GPUs (C) increases capacity but does not inherently fix skewed scheduling; you can still end up with hotspots and idle devices if the workload distribution remains uneven.

References: NVIDIA Triton Inference Server – Model configuration (instances/batching), NVIDIA GPU Operator documentation.

QUESTION NO: 7

A large enterprise is deploying a high-performance AI infrastructure to accelerate its machine

learning workflows. They are using multiple NVIDIA GPUs in a distributed environment. To optimize the workload distribution and maximize GPU utilization, which of the following tools or frameworks should be integrated into their system? (Select two)

A. NVIDIA CUDA

B. NVIDIA NGC (NVIDIA GPU Cloud)

C. TensorFlow Serving

D. NVIDIA NCCL (NVIDIA Collective Communications Library)

E. Keras

ANSWER: A D

Explanation:

For distributed training across multiple NVIDIA GPUs (often across multiple nodes), you typically need (1) a GPU programming/runtime layer and (2) an optimized multi-GPU communication layer. NVIDIA CUDA is the foundational platform that enables GPU-accelerated compute and provides the core runtime and libraries that most ML frameworks rely on to execute kernels efficiently on NVIDIA GPUs. While CUDA alone doesn't "schedule" work across nodes, it is still a required building block for maximizing GPU utilization because the training stack ultimately runs on CUDA-enabled kernels.

NVIDIA NCCL is the key distributed component here: it provides highly optimized collective communication primitives (e.g., all-reduce, all-gather, broadcast) used by distributed deep learning to synchronize gradients and parameters efficiently across GPUs and nodes. NCCL is widely used under the hood by frameworks like PyTorch DDP and Horovod to scale training while keeping GPUs busy.

NVIDIA NGC is valuable for obtaining optimized containers and software, but it's not itself a workload distribution/communication framework. TensorFlow Serving targets inference deployment rather than distributed training utilization. Keras is a high-level modeling API and depends on backends (e.g., TensorFlow) for distribution and GPU efficiency rather than providing it directly.

References: NVIDIA CUDA Documentation, NVIDIA NCCL User Guide

QUESTION NO: 8

A data center is running a cluster of NVIDIA GPUs to support various AI workloads. The operations team needs to monitor GPU performance to ensure workloads are running efficiently and to prevent potential hardware failures. Which two key measures should they focus on to monitor the GPUs effectively? (Select two)

A. Disk I/O rates

B. CPU clock speed

C. GPU temperature and power consumption

D. GPU memory utilization

E. Network bandwidth usage

ANSWER: C D

Explanation:

For effective GPU operations monitoring, the team should prioritize metrics that directly indicate (1) hardware health/risk and (2) whether workloads are using the GPU efficiently. **GPU temperature and power consumption** are core health signals: sustained high temperatures can trigger thermal throttling (reduced performance) and increase failure risk, while abnormal or consistently high power draw can indicate misconfiguration, cooling issues, or workload behavior that may stress the board. These are also common alerting targets in data centers because they correlate strongly with stability and longevity.

GPU memory utilization is a key efficiency metric for AI workloads. Many training/inference jobs are memory-bound; high memory usage can lead to OOM errors or reduced throughput due to fragmentation/pressure, while unexpectedly low usage can indicate underutilization or scheduling/placement issues. Both temperature/power and memory utilization are first-class metrics exposed by NVIDIA management tooling such as `nvidia-smi` (NVML-backed), making them practical and standard for continuous monitoring.

The other options are not “key GPU measures” in this context: disk I/O and network bandwidth are system-level bottlenecks (important, but not GPU health/perf measures), and CPU clock speed is unrelated to GPU condition. References: NVIDIA System Management Interface (nvidia-smi), NVIDIA NVML API Documentation.

QUESTION NO: 9

You are part of a team analyzing the results of a machine learning experiment that involved training models with different hyperparameter settings across various datasets. The goal is to identify trends in how hyperparameters and dataset characteristics influence model performance, particularly accuracy and overfitting. Which analysis method would best help in identifying the relationships between hyperparameters, dataset characteristics, and model performance?

A. Conduct a correlation matrix analysis between hyperparameters, dataset characteristics, and performance metrics.

B. Apply PCA (Principal Component Analysis) to reduce the dimensionality of hyperparameter settings.

C. Create a bar chart comparing accuracy for different hyperparameter settings.

D. Use a pie chart to show the distribution of accuracy scores across datasets.

ANSWER: A

Explanation:

A correlation matrix analysis (Option A) is the best fit when the goal is to identify relationships between multiple variables—hyperparameters (e.g., learning rate, batch size), dataset characteristics (e.g., dataset size, class imbalance), and outcomes (accuracy, overfitting indicators like train–val gap). A correlation matrix quantifies pairwise associations across all variables at once, making it easy to spot trends such as “higher learning rate correlates with worse generalization” or “larger datasets correlate with higher accuracy.” In GPU-accelerated analytics workflows, RAPIDS cuDF supports computing correlations efficiently on large experiment tables, which aligns well with NVIDIA AI infrastructure best practices for scalable experiment analysis.

PCA (Option B) is mainly for dimensionality reduction and visualization; it can help summarize variance but does not directly report interpretable relationships between specific hyperparameters and specific metrics. A bar chart (Option C) can compare accuracy across a small set of settings, but it’s limited to a narrow slice of the problem and doesn’t capture multivariate relationships. A pie chart (Option D) is generally inappropriate for analyzing trends or relationships in continuous performance metrics.

References: RAPIDS cuDF DataFrame.corr(), scikit-learn PCA documentation.

QUESTION NO: 10

Which two software components are directly involved in the life cycle of AI development and deployment, particularly in model training and model serving? (Select two)

A. Prometheus

B. MLflow

C. Airflow

D. Apache Spark

E. Kubeflow

ANSWER: B E

Explanation:

The best two choices are **MLflow** and **Kubeflow** because both are designed to manage key parts of the ML lifecycle and are commonly used in GPU-accelerated AI platforms for *training* and *serving/deployment* workflows. MLflow focuses on experiment tracking, packaging models, and deploying/serving models (via MLflow Models/Serving integrations), making it directly relevant to the train-to-serve lifecycle. Kubeflow is a Kubernetes-native ML toolkit that orchestrates end-to-end pipelines, including distributed training jobs (e.g., TFJob/PyTorchJob via Kubeflow Training Operator) and model serving (commonly integrated with KServe and/or NVIDIA Triton Inference Server in Kubernetes environments).

Prometheus is important operationally, but it's primarily for metrics monitoring/alerting rather than training or serving itself. **Airflow** is a general workflow orchestrator (often for ETL/data pipelines) and can be used around ML, but it's not a core training/serving component. **Apache Spark** is mainly for large-scale data processing/feature engineering; it's not typically a model serving component and isn't focused on managing the ML lifecycle end-to-end.

References: MLflow Documentation, Kubeflow Documentation