DUMPSBOSS.

NVIDIA Generative AI LLMs

NVIDIA NCA-GENL

Version Demo

Total Demo Questions: 10

Total Premium Questions: 95

Buy Premium PDF

https://dumpsboss.co

support@dumpsboss.co

support@dumpsboss.co
dumpsboss.co

QUESTION NO: 1

In neural networks, the vanishing gradient problem refers to what problem or issue?

A. The problem of overfitting in neural networks, where the model performs well on the training data but poorly on new, unseen data.

B. The issue of gradients becoming too large during backpropagation, leading to unstable training.

C. The problem of underfitting in neural networks, where the model fails to capture the underlying patterns in the data.

D. The issue of gradients becoming too small during backpropagation, resulting in slow convergence or stagnation of the training process.

ANSWER: D

Explanation:

The vanishing gradient problem is the situation where gradients shrink toward zero as they are backpropagated through many layers (or many time steps in RNNs). When gradients become extremely small, earlier layers receive little to no learning signal, so their weights update very slowly (or effectively stop updating). This commonly leads to slow convergence, stalled training, and difficulty learning long-range dependencies—especially with saturating activations like sigmoid/tanh and in very deep networks without architectural mitigations. Option D matches this definition precisely.

Option B describes the opposite failure mode: exploding gradients, where gradients grow too large and cause unstable updates (often addressed with gradient clipping). Options A and C are generalization/capacity issues (overfitting and underfitting) and are not specifically about gradient magnitudes during backpropagation. In practice, vanishing gradients are mitigated by choices like ReLU-family activations, careful initialization, normalization layers, and skip/residual connections (e.g., ResNets/Transformers), which help preserve gradient flow.

References: Deep Learning Book (Goodfellow et al.) – RNNs and vanishing/exploding gradients, PyTorch nn documentation (activation functions and common building blocks).

QUESTION NO: 2

When implementing data parallel training, which of the following considerations needs to be taken into account?

A. The model weights are synced across all processes/devices only at the end of every epoch.

B. A master-worker method for syncing the weights across different processes is desirable due to its scalability.

C. A ring all-reduce is an efficient algorithm for syncing the weights across different processes/devices.

D. The model weights are kept independent for as long as possible increasing the model exploration.

ANSWER: C

Explanation:

In data parallel training, each GPU/process holds a full replica of the model and computes gradients on a different shard of the mini-batch. A key implementation consideration is how to efficiently aggregate those gradients (or equivalently synchronize parameters) across all replicas every iteration so that all model copies stay consistent. Collective communication primitives such as all-reduce are the standard approach; in particular, ring all-reduce is widely used because it is bandwidth-efficient and scales well by avoiding a single “parameter server” bottleneck. Frameworks like PyTorch DDP and NVIDIA NCCL commonly rely on all-reduce collectives for gradient synchronization.

Option A is incorrect because synchronizing only at epoch boundaries would let replicas drift for many steps, effectively becoming different models and harming convergence (unless doing special algorithms like local SGD, which is not the default “data parallel” assumption). Option B is incorrect because master-worker/parameter-server designs can become communication bottlenecks and are generally less scalable than decentralized collectives for synchronous data parallelism. Option D is incorrect because keeping weights independent contradicts standard synchronous data parallel training, where replicas must be kept in lockstep via gradient/weight synchronization.

References: NVIDIA NCCL User Guide – Collective operations (AllReduce), PyTorch DistributedDataParallel documentation.

QUESTION NO: 3

What is the purpose of the NVIDIA NGC catalog?

A. To provide a platform for testing and debugging software applications.

B. To provide a platform for developers to collaborate and share software development projects.

C. To provide a marketplace for buying and selling software development tools and resources.

D. To provide a curated collection of GPU-optimized AI and data science software.

ANSWER: D

Explanation:

The NVIDIA NGC (NVIDIA GPU Cloud) catalog exists to give developers and enterprises a curated hub of NVIDIA-optimized software assets—most notably GPU-accelerated containers, pretrained models, Helm charts, and SDK resources—so they can build, train, and deploy AI/data science workloads faster and more reliably on NVIDIA hardware. In practice, NGC reduces setup time and integration risk by providing artifacts that are already validated and tuned for NVIDIA GPUs and the broader NVIDIA AI software stack (e.g., CUDA, TensorRT, NeMo, etc.). This aligns directly with how NGC is used in generative AI/LLM workflows: teams can pull ready-to-run containers and models rather than assembling everything from scratch.

Option D is correct because it captures NGC’s core purpose: a curated collection of GPU-optimized AI and data science software. Option A is incorrect because NGC is not primarily a testing/debugging platform (those functions are handled by developer tools and profilers, not a catalog). Option B is incorrect because NGC is not a general collaboration/code-hosting service like GitHub/GitLab. Option C is incorrect because NGC is not positioned as a marketplace for buying/selling tools; it’s a distribution catalog for NVIDIA and partner assets.

References: NVIDIA NGC Catalog User Guide, NVIDIA GPU Cloud (NGC) overview.

QUESTION NO: 4

Which of the following contributes to the ability of RAPIDS to accelerate data processing? (Pick the 2 correct responses)

A. Ensuring that CPUs are running at full clock speed.

B. Subsampling datasets to provide rapid but approximate answers.

C. Using the GPU for parallel processing of data.

D. Enabling data processing to scale to multiple GPUs.

E. Providing more memory for data analysis.

ANSWER: C D

Explanation:

RAPIDS accelerates data processing primarily by moving common dataframe and ML operations onto NVIDIA GPUs and by scaling those operations across multiple GPUs when needed. Using the GPU enables massive parallelism (thousands of cores) and high memory bandwidth, which can dramatically speed up operations like joins, groupbys, filtering, and many ML primitives compared to CPU-only execution—this is exactly what option C describes. RAPIDS also supports multi-GPU and distributed execution (e.g., via Dask with cuDF/cuML), allowing larger-than-single-GPU workloads to be partitioned and processed concurrently across GPUs, which is the core idea behind option D.

Option A is not a RAPIDS feature; RAPIDS doesn't "ensure CPUs run at full clock speed" and CPU frequency isn't the main mechanism for RAPIDS acceleration. Option B is incorrect because RAPIDS is generally designed to compute the same results as equivalent CPU libraries (pandas/NumPy/Scikit-learn style workflows), not to rely on subsampling for approximate answers as a primary acceleration strategy. Option E is misleading: RAPIDS doesn't inherently "provide more memory"; available memory depends on the hardware (GPU VRAM/system RAM), though RAPIDS can manage GPU memory efficiently.

References: RAPIDS Overview, RAPIDS Documentation

QUESTION NO: 5

Which of the following claims is correct about quantization in the context of Deep Learning? (Pick the

2 correct responses)

A. Quantization might help in saving power and reducing heat production.

B. It consists of removing a quantity of weights whose values are zero.

C. It leads to a substantial loss of model accuracy.

D. Helps reduce memory requirements and achieve better cache utilization.

E. It only involves reducing the number of bits of the parameters.

ANSWER: A D

Explanation:

Quantization is a model optimization technique that represents weights and/or activations with lower-precision numeric formats (commonly FP16, INT8, or even INT4) instead of FP32. This typically reduces compute cost and memory bandwidth, which can translate into lower power draw and less heat—especially on edge devices or when using INT8/FP16 tensor cores—so option A is correct. Quantization also reduces the model's memory footprint (e.g., INT8 weights are 4× smaller than FP32), improving cache residency and reducing memory traffic, which often improves inference throughput and latency; that makes option D correct.

Option B describes pruning (removing weights, often zeros), not quantization. Option C is too strong: while quantization can reduce accuracy, modern approaches (post-training quantization with calibration, or quantization-aware training) are designed to keep accuracy loss small and workload-dependent, not “substantial” by default. Option E is incorrect because quantization is not “only” reducing bits; practical quantization includes scale/zero-point (or other) mapping, calibration, and sometimes per-tensor/per-channel schemes.

References: NVIDIA TensorRT Quantization, TensorRT Developer Guide.

QUESTION NO: 6

How does A/B testing contribute to the optimization of deep learning models' performance and effectiveness in real-world applications? (Pick the 2 correct responses)

A. A/B testing helps validate the impact of changes or updates to deep learning models by statistically analyzing the outcomes of different versions to make informed decisions for model optimization.

B. A/B testing allows for the comparison of different model configurations or hyperparameters to identify the most effective setup for improved performance.

C. A/B testing in deep learning models is primarily used for selecting the best training dataset without requiring a model architecture or parameters.

D. A/B testing guarantees immediate performance improvements in deep learning models without the need for further analysis or experimentation.

E. A/B testing is irrelevant in deep learning as it only applies to traditional statistical analysis and not complex neural network models.

ANSWER: A B

Explanation:

A/B testing is a practical way to optimize deployed deep learning systems because it evaluates changes under real production conditions with controlled exposure. The key value is that it provides statistically grounded evidence about whether a new model (or a change in the serving stack) improves the metrics you actually care about in the real world—accuracy/quality, latency, conversion, user satisfaction, error rates, etc. That’s why option A is correct: you can roll out a candidate model to a subset of traffic and compare outcomes against a baseline, enabling data-driven decisions rather than relying on offline benchmarks alone.

Option B is also correct: A/B testing can compare different model variants (which may reflect different architectures, training runs, quantization levels, decoding strategies, or other configuration choices) to determine which performs best on online KPIs. This is commonly paired with canary releases and model versioning in inference platforms.

Option C is incorrect because dataset selection is typically handled via offline experimentation and evaluation pipelines; A/B testing is about comparing system variants in production. Option D is wrong because A/B testing never “guarantees” improvement—results require statistical analysis and may show no gain or regressions. Option E is incorrect because A/B testing is widely used for ML/LLM deployments, including complex neural models.

References: NVIDIA Triton Inference Server Documentation, NVIDIA Developer Blog: Deploying models with Triton

QUESTION NO: 7

In the context of machine learning model deployment, how can Docker be utilized to enhance the process?

A. To automatically generate features for machine learning models.

B. To provide a consistent environment for model training and inference.

C. To reduce the computational resources needed for training models.

D. To directly increase the accuracy of machine learning models.

ANSWER: B

Explanation:

Docker enhances ML model deployment by packaging the application (your inference code), model artifacts, and all runtime dependencies (CUDA/cuDNN versions, Python libraries, system packages, configuration files) into a portable container image. This makes the environment consistent across development, testing, and production, which is critical for reproducibility and for avoiding “it works on my machine” issues. In NVIDIA workflows, this is especially common with Triton Inference Server and NGC: NVIDIA publishes GPU-optimized container images, and teams deploy those images unchanged across different hosts as long as the NVIDIA Container Toolkit and compatible drivers are present. That consistency simplifies CI/CD, scaling, rollbacks, and multi-node deployments because the container becomes the unit of deployment.

Option A is incorrect because Docker does not perform feature engineering; it only provides an execution environment. Option C is incorrect because containerization doesn’t inherently reduce compute needs; it may even add slight overhead, though typically minimal. Option D is incorrect because Docker does not directly improve model accuracy; accuracy is driven by data, architecture, and training, not packaging.

References: NVIDIA Triton Inference Server Documentation, NVIDIA NGC Overview

QUESTION NO: 8

What are the main advantages of instructed large language models over traditional, small language models (< 300M parameters)? (Pick the 2 correct responses)

A. Trained without the need for labeled data.

B. Smaller latency, higher throughput.

C. It is easier to explain the predictions.

D. Cheaper computational costs during inference.

E. Single generic model can do more than one task.

ANSWER: D E

Explanation:

Instructed large language models (LLMs) are typically pretrained on massive unlabeled corpora and then instruction-tuned so they can follow natural-language prompts and generalize across many tasks. The biggest practical advantage over small (<300M) task-specific models is breadth: one instruction-tuned LLM can perform summarization, extraction, classification, translation, and dialogue without training a separate model per task, so option E is correct. This “one model, many tasks” behavior is a core motivation for instruction tuning and is widely documented in modern LLM training workflows.

Option D can also be correct in the sense of overall system cost: while per-token inference on a large model is usually more expensive than on a small model, an instructed LLM can reduce total inference/maintenance cost when it replaces multiple

specialized models, avoids repeated retraining, and simplifies deployment (fewer pipelines, fewer model versions). So “cheaper computational costs during inference” can be true at the application level (total cost of ownership), even if not always true per request.

Option A is not a distinguishing advantage: both small and large models can be trained without labeled data during pretraining, and instruction tuning often uses labeled/curated prompt-response data. Option B is generally false because larger models tend to have higher latency/lower throughput. Option C is false because larger neural models are usually harder to interpret.

References: NVIDIA NeMo Framework User Guide, Wei et al., “Finetuned Language Models Are Zero-Shot Learners (FLAN)”.

QUESTION NO: 9

Which of the following optimizations are provided by TensorRT? (Choose two.)

A. Data augmentation

B. Variable learning rate

C. Multi-Stream Execution

D. Layer Fusion

E. Residual connections

ANSWER: C D

Explanation:

TensorRT is an inference optimizer and runtime that improves deployed model performance by transforming and scheduling the network for efficient execution on NVIDIA GPUs. Two classic TensorRT optimizations are **layer fusion** and **multi-stream execution**. Layer fusion combines compatible sequences of operations (for example, convolution + bias + activation) into fewer GPU kernels, reducing kernel-launch overhead and memory traffic, which typically improves latency and throughput. Multi-stream execution enables concurrent execution of independent inference work using CUDA streams, helping increase GPU utilization and overall throughput when serving multiple requests or overlapping compute and data movement.

The other options are not TensorRT “optimizations” in the inference-engine sense. **Data augmentation** and **variable learning rate** are training-time techniques (augmentation is preprocessing; learning-rate schedules affect optimizer updates) and are outside TensorRT’s scope. **Residual connections** are an architectural pattern in model design; TensorRT can execute networks that contain residuals, but it doesn’t “add” residual connections as an optimization.

References: NVIDIA TensorRT Documentation (Architecture Overview), TensorRT: Working with CUDA (streams).

QUESTION NO: 10

What is the main consequence of the scaling law in deep learning for real-world applications?

A. With more data, it is possible to exceed the irreducible error region.

B. The best performing model can be established even in the small data region.

C. Small and medium error regions can approach the results of the big data region.

D. In the power-law region, with more data it is possible to achieve better results.

ANSWER: D

Explanation:

Scaling laws for deep learning show that, over a broad “power-law” regime, model loss (or error) decreases in a predictable way as you scale up key ingredients such as training compute, dataset size, and model parameters. The practical consequence for real-world applications is that you can often get reliably better performance by investing in more data and/or more compute (and appropriately scaling the model), which is exactly why large language models tend to improve as they are trained on larger corpora with more compute. Option D captures this: in the power-law region, adding more data (and typically the matching compute/model capacity) yields better results.

Option A is incorrect because the irreducible error region represents a floor driven by noise/ambiguity in the task or data; simply adding more data cannot “beat” that fundamental limit. Option B is incorrect because the small-data regime is usually dominated by underfitting/overfitting and does not let you confidently identify the best achievable model without scaling. Option C is incorrect/misleading because small/medium-data regimes generally do not “approach” big-data performance unless you actually move into the scaling regime by increasing data/compute/model size.

References: Kaplan et al., Scaling Laws for Neural Language Models; Chinchilla (Hoffmann et al.), Training Compute-Optimal Large Language Models.