Q1. Computational Graphs & Gradients

In PyTorch, **Gradient Accumulation** is the process where gradients are summed across multiple backward passes before updating the model weights.

- **The .grad behavior:** When one calls *loss.backward(),* PyTorch calculates the gradients and **adds** them to the existing *.grad* attribute of the leaf tensors (parameters). It does not overwrite the old gradient.
- **The Necessity of optimizer.zero_grad():** Since PyTorch accumulates gradients by default, if one forgets to zero them out, the gradients from the current batch will be added to the gradients of the previous batch.
- **The Mathematical Impact:** If one forgets zero_grad(), their update step becomes: **New Theta = Old Theta - (Learning Rate * Sum of all Gradients from step 1 to now)**.

This causes the gradients to grow incorrectly large, leading to divergent training (the model "explodes") or the model learning based on "stale" data from previous iterations rather than the current batch.

Q2. Tensors: View vs. Reshape

Both methods change the shape of a tensor, but they handle the underlying memory differently.

- **Strict Technical Difference:** * **.view()** creates a new view of the data **only** if the tensor is already "contiguous" in memory (stored in a single, unbroken block in the correct order). If the tensor has been transposed or sliced, .view() will throw an error because it cannot change the shape without moving the data.

.reshape() is more flexible. It returns a "view" if the tensor is contiguous, but if it is not, it automatically **copies** the data into a new contiguous memory block and then changes the shape.

- **Safety Fallback: .reshape()** is the safer choice to use as a fallback if one is unsure about the memory stride or contiguity, as it prevents the code from crashing.

Q3. Device Management (CPU vs. GPU)

- **Cross-Device Operations:** Attempting to perform an operation (like addition) between a Tensor located on the **CPU** and a Tensor located on **cuda:0** (GPU) will result in a **RuntimeError**. PyTorch requires all tensors involved in a single calculation to be on the same physical device.
- **New Tensors in forward:** When one moves a model to the GPU using model.to('cuda'), only the parameters already defined in the model are moved. Any **new** tensors created dynamically inside the forward method (for example, torch.zeros(10)) will default to the **CPU**.
- **Best Practice:** To avoid device mismatch errors, use torch.zeros(10, device=x.device). This ensures that any new tensor one creates automatically inherits the device of the input data x.