```
import random
import torch
import numpy as np

class ReplayBufferDQN:
    def __init__(self, buffer_size:int, seed:int=42):
        self.buffer_size = buffer_size
        self.seed = seed
        self.buffer = []
        random.seed(self.seed)

    def add(self, state:np.ndarray, action:int, reward:float, next_state:np.ndarray
            , done:bool):
        """
        Add a new experience to the buffer

        Args:
            state (np.ndarray): the current state of shape [n_c,h,w]
            action (int): the action taken
            reward (float): the reward received
            next_state (np.ndarray): the next state of shape [n_c,h,w]
            done (bool): whether the episode is done
        """
        self.buffer.append((state, action, reward, next_state, done))
        if len(self.buffer) > self.buffer_size:
            self.buffer.pop(0)


    def sample(self, batch_size:int, device='cpu'):
        """
        Randomly sample a batch of experiences from the replay buffer.

        Args:
            batch_size (int): the number of samples to take

        Returns:
            states (torch.Tensor): Tensor of shape (batch_size, n_channels, height, width),
dtype torch.float32.
            actions (torch.Tensor): Tensor of shape (batch_size,), dtype torch.int64
(converted via `.long()`).
            rewards (torch.Tensor): Tensor of shape (batch_size,), dtype torch.float32.
            next_states (torch.Tensor): Tensor of shape (batch_size, n_channels, height,
width), dtype torch.float32.
            dones (torch.Tensor): Tensor of shape (batch_size,), dtype torch.bool.

        Notes:
            1. Use `random.sample` for uniform sampling without replacement.
            2. Convert NumPy arrays to torch tensors with the correct dtype before moving to
`device`.
            3. Use `torch.stack` to combine individual tensors into a batch dimension.
            4. Keep the output shapes and dtypes consistent.
        """

        # ========== YOUR CODE HERE ==========
        # TODO:
        # 1. sample random indices
        # 2. collect experiences using the sampled indices
        # 3. stack and move batches to the specified device, making sure to convert to the
correct dtype
        # ====================================

        # step 1
        batch = random.sample(self.buffer, batch_size)

        # step 2
        states, actions, rewards, next_states, dones = zip(*batch)
        # step 3
        states = torch.stack([torch.tensor(s, dtype=torch.float32) for s in
states]).to(device)
        actions = torch.tensor(actions, dtype=torch.int64).to(device)
        rewards = torch.tensor(rewards, dtype=torch.float32).to(device)
        next_states = torch.stack([torch.tensor(ns, dtype=torch.float32) for ns in
next_states]).to(device)
        dones = torch.tensor(dones, dtype=torch.bool).to(device)

        # ========== YOUR CODE ENDS ==========

        return states, actions, rewards, next_states, dones


    def __len__(self):
        return len(self.buffer)
```