“DQN Setup and Replay Buffer” – name of chatroom in Quant Trading Project

so given my result: and as per the codes, what am i training on and what am i testing on? so i train on the entire dataset and i test on all the dataset, therefore these is leakage? where is the train test split. how do i interpret and understand the poritioning of the data

**ChatGPT said:**

You're absolutely right to ask this — **right now there’s no explicit train/test split in your code**, which does introduce **data leakage** and prevents proper evaluation of generalization.

**✅ What You're Currently Doing**

* You're **looping over the full dataset** from start to finish in one pass:

python

CopyEdit

for t in range(1, len(env\_df) - 1):

* At each step:
  + You compute the current state.
  + Select an action using ε-greedy (using the *current* policy).
  + Store transition in replay buffer.
  + Train on mini-batches sampled from the buffer — these contain samples from **previous parts of the same dataset**.
  + You keep updating and evaluating performance on the same dataset.

**⚠️ Why this is a problem:**

* You are training and validating (implicitly testing) on the same data.
* There’s **no way to know if your agent will perform well on unseen data** (e.g., a new ticker, different market conditions, future time period).

**💡 What You Should Do Instead (Train/Test Split Options)**

**Option 1: Temporal Train/Test Split**

In time-series, you usually split **chronologically**:

python

CopyEdit

split\_index = int(len(df) \* 0.8)

train\_df = df[:split\_index]

test\_df = df[split\_index:]

Then:

* Use train\_df for agent training.
* After training ends, **run the agent on test\_df without updating weights** — purely inference mode.

This tests generalization across future time.

**Option 2: Multiple Tickers (if available)**

If you have many tickers:

* Train on subset of tickers.
* Test on **never-seen tickers**.

This tests whether the agent can generalize across different assets.

**Option 3: Episode-Based Training**

You treat each "episode" as a different environment:

python

CopyEdit

for episode\_df in list\_of\_dfs: # could be by day, week, or ticker

train agent using episode\_df

Then keep one or more aside as test episodes.

**🧠 Suggested Fix**

Here's a simple example to implement a train/test split in your current setup:

python

CopyEdit

split\_idx = int(len(env\_df) \* 0.8)

train\_df = env\_df.iloc[:split\_idx]

test\_df = env\_df.iloc[split\_idx:]

Then change:

python

CopyEdit

# TRAINING

for t in range(1, len(train\_df) - 1):

...

After that, run:

python

CopyEdit

# TESTING (no learning, only agent execution)

wallet = 0

position, entry\_price = 0, 0

for t in range(1, len(test\_df) - 1):

state = torch.tensor(get\_state(t), dtype=torch.float32).unsqueeze(0).to(device)

with torch.no\_grad():

action = torch.argmax(policy\_net(state)).item()

...

# simulate trading but do NOT update weights

**📉 How to Evaluate on Test Set**

* Plot **wallet growth on test set only**
* Compare it to:
  + Buy & Hold
  + Random Agent
  + Rule-based baseline
* Track metrics like:
  + Final PnL
  + Sharpe Ratio
  + Max drawdown

**🧾 Summary**

| **Aspect** | **Current Code** | **Suggested Fix** |
| --- | --- | --- |
| Training data | Entire dataset | Only first 80% (or specific tickers) |
| Testing data | None (same as training) | Final 20% or unseen data |
| Data leakage | ✅ Present | ❌ Avoid with proper split |
| Evaluation method | Wallet on training data | Wallet on *held-out test data* |

Would you like help refactoring your code to include a proper train/test split and inference loop?