In [15]:
```
#https://www.w3schools.com/python/python_ml_decision_tree.asp
```

In [16]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report

df = pd.read_csv("cancerAllv3.csv")
```

In [17]:
```
features=['radius','texture','perimeter','area','s','c','concavity','cp','sym']

import numpy as np
X = np.array(df)
y = X[:,30]
X = X[:,0:9]

X_train_full, X_test, y_train_full, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_train_full, y_train_full, test_size=0.2, random_state=42)



clf = DecisionTreeClassifier(criterion='gini')
clf = clf.fit(X_train, y_train)
# Evaluate the Decision Tree on the validation set
dt_val_preds = clf.predict(X_val)
print(f"Decision Tree Validation Accuracy: {accuracy_score(y_val, dt_val_preds):.2f}")
```

```
Decision Tree Validation Accuracy: 0.91
```

In [23]:
```
class RLAgent:
    def __init__(self, actions):
        self.q_table = {}  # Q-table for storing rewards
        self.actions = actions  # Actions: ["maintain", "override"]
        self.learning_rate = 0.1
        self.discount_factor = 0.9
        self.epsilon = 0.1  # Exploration rate

    def get_state(self, weather_conditions, dt_prediction):
        # Simplify state representation by discretizing feature values
        state = tuple(np.digitize(weather_conditions, bins=np.linspace(0, 1, 5))) + (dt_prediction,)
        return state

    def choose_action(self, state):
        if np.random.rand() < self.epsilon or state not in self.q_table:
            return np.random.choice(self.actions)
        return max(self.q_table[state], key=self.q_table[state].get)

    def update_q_table(self, state, action, reward, next_state):
        if state not in self.q_table:
            self.q_table[state] = {a: 0 for a in self.actions}
        if next_state not in self.q_table:
            self.q_table[next_state] = {a: 0 for a in self.actions}

        # Update Q-value
        current_q = self.q_table[state][action]
        max_future_q = max(self.q_table[next_state].values())
        self.q_table[state][action] = current_q + self.learning_rate * (
```

```
            reward + self.discount_factor * max_future_q - current_q
        )

# Improved Reward Function
def reward_function(final_prediction, true_label):
    if final_prediction == true_label:
        return 1
    elif final_prediction == 1 and true_label == 0:  # False positive
        return -0.5
    else:  # False negative
        return -1.0

# Train RL Agent
agent = RLAgent(actions=["maintain", "override"])
for epoch in range(100):
    for i in range(len(X_val)):
        # Access row as a list
        weather_conditions = X_val.iloc[i].tolist()
        dt_prediction = dt_val_preds[i]
        true_label = y_val.iloc[i]

        state = agent.get_state(weather_conditions, dt_prediction)

        # Agent chooses an action
        action = agent.choose_action(state)

        # Simulate action result
        if action == "maintain":
            final_prediction = dt_prediction
        else:  # "override"
            final_prediction = 1 - dt_prediction  # Flip the prediction

        # Calculate reward
        reward = reward_function(final_prediction, true_label)

        # Get next state (same environment for simplicity)
        next_state = state

        # Update Q-table
        agent.update_q_table(state, action, reward, next_state)

    # Decay epsilon
    agent.epsilon = max(0.01, agent.epsilon * 0.99)


# Test the Hybrid Model
final_preds = []
for i in range(len(X_test)):
    # Access the row using .iloc and convert it to a list
    weather_conditions = X_test.iloc[i].tolist()
    dt_prediction = dt_test_preds[i]

    state = agent.get_state(weather_conditions, dt_prediction)
    action = agent.choose_action(state)

    if action == "maintain":
        final_preds.append(dt_prediction)
    else:
        final_preds.append(1 - dt_prediction)
```

```python
# Final evaluation of the hybrid model
print(f"Hybrid Model Test Accuracy: {accuracy_score(y_test, final_preds):.2f}")
```

```
Hybrid Model Test Accuracy: 0.90
```

In [ ]:

```python
```