Challenge #10: Identify Computational Bottlenecks

Step 1: FrozenLake Q-Learning Code Summary

The FrozenLake Q-learning code uses nested loops to update a Q-table. Major operations per step include:
- Random action selection
- Action execution via OpenAI Gym
- Q-table update using: Q[state, action] = Q[state, action] + alpha * (reward + gamma * max(Q[new_state]) - Q[state, action])

Step 2: Identified Bottlenecks

The primary computational bottleneck is the repeated Q-table update. The max and arithmetic operations inside loops dominate runtime.

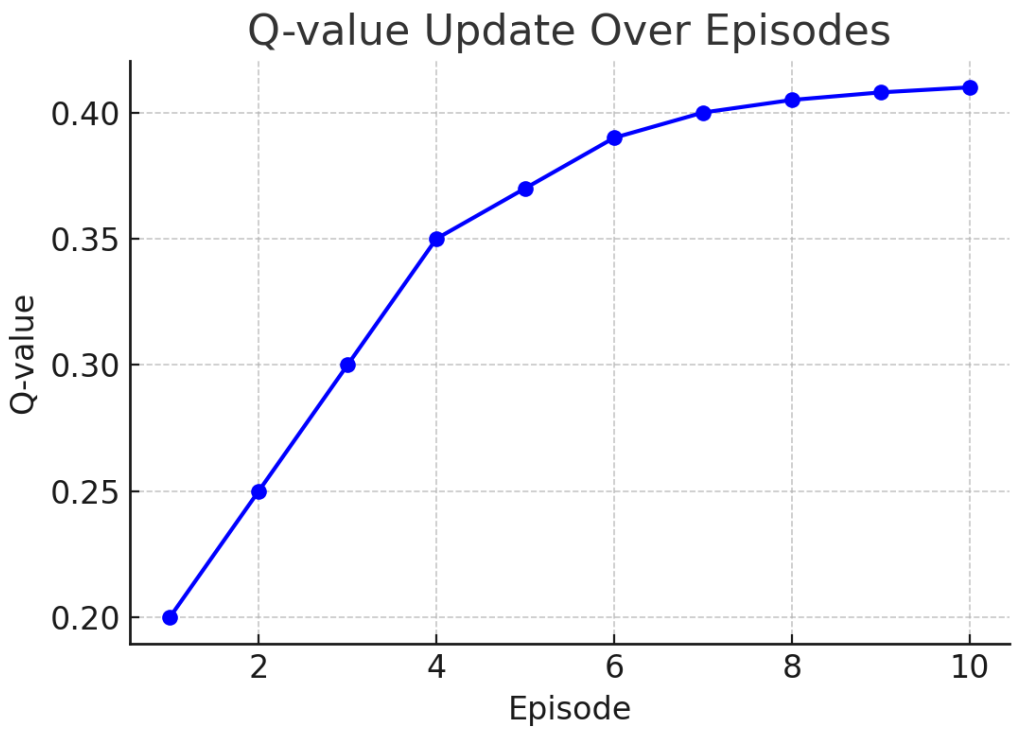
Step 3: Hardware Implementation Proposal

A dedicated hardware unit for Q-value updates can accelerate training. The hardware takes current_q, reward, next_q_max, alpha, and gamma to compute the updated Q-value.

Step 4: SystemVerilog Code

```
module q_update_unit (
    input logic clk,
    input logic rst,
    input logic [31:0] current_q,
    input logic [31:0] reward,
    input logic [31:0] next_q_max,
    input logic [31:0] alpha,
    input logic [31:0] gamma,
    output logic [31:0] updated_q
);
    logic [31:0] temp1, temp2, temp3;
    always_ff @(posedge clk or posedge rst) begin
        if (rst) begin
            updated_q <= 0;
        end else begin
            temp1 = gamma * next_q_max;
            temp2 = reward + temp1 - current_q;
            temp3 = alpha * temp2;
            updated_q <= current_q + temp3;
        end
    end
endmodule
```

Step 5: Visualization


Q-value Update Over Episodes
Q-value
0.40
0.35
0.30
0.25
0.20
2
4
6
8
10
Episode