# LLM Reflection Report – Challenge #4

LLM Used: ChatGPT-4o

Challenge: Challenge-4

## 1. Overview

This report summarizes the process of replicating the paper 'Designing Silicon Brains using LLM' using ChatGPT-4o. It includes original and AI-generated queries, HDL code, testbenches, debugging efforts, and improvements. The goal was to evaluate the effectiveness of LLMs in hardware design and reflect on their capabilities and limitations.

## 2. Replication Summary

ChatGPT was able to replicate the Johns Hopkins paper by:

- Generating a Verilog module for a Leaky Integrate-and-Fire (LIF) neuron
- Simulating a spiking neural array in Verilog
- Designing a parameterized neuron array
- Connecting modules via SPI and a top-level system

## 3. Key Prompts and Generated Code

### 3.1 Generate a Verilog module for a Leaky Integrate-and-Fire neuron

Verilog code:

```
module lif_neuron (
    input clk,
    input reset,
    input [7:0] current_in,
    input [7:0] threshold,
    input [7:0] leak_rate,
    input [7:0] refractory_period,
    output reg spike
);
    reg [7:0] membrane_potential = 0;
    reg [7:0] refrac_counter = 0;
    reg in_refrac = 0;
```

```
  always @(posedge clk or posedge reset) begin
    if (reset) begin
      membrane_potential <= 0;
      refrac_counter <= 0;
      in_refrac <= 0;
      spike <= 0;
    end else begin
      spike <= 0;
      if (in_refrac) begin
        refrac_counter <= refrac_counter - 1;
        if (refrac_counter == 0)
          in_refrac <= 0;
      end else begin
        if (membrane_potential < leak_rate)
          membrane_potential <= current_in;
        else if (membrane_potential + current_in < membrane_potential)
          membrane_potential <= threshold;
        else
          membrane_potential <= membrane_potential + current_in - leak_rate;
        if (membrane_potential >= threshold) begin
          membrane_potential <= 0;
          spike <= 1;
          in_refrac <= 1;
          refrac_counter <= refractory_period;
        end
      end
    end
  end
endmodule
```

## 3.2 How do I simulate a spiking neural array in Verilog?

Testbench code:

```
module tb_neuron_array;
  reg clk, reset;
  reg [7:0] input_current [2:0];
  wire [2:0] spikes;
  genvar i;
  generate
    for (i = 0; i < 3; i = i + 1) begin : neuron_array
      lif_neuron u_neuron (
        .clk(clk),
        .reset(reset),
        .current_in(input_current[i]),
            .threshold(8'd100),
            .leak_rate(8'd2),
            .refractory_period(8'd5),
            .spike(spikes[i])
        );
    end
  endgenerate
  always #5 clk = ~clk;
  initial begin
    $dumpfile("dump.vcd");
    $dumpvars(0, tb_neuron_array);
    clk = 0; reset = 1;
    input_current[0] = 0;
    input_current[1] = 0;
    input_current[2] = 0;
    #10 reset = 0;
    repeat (20) begin
      input_current[0] = $random % 10 + 5;
      input_current[1] = $random % 10 + 10;
      input_current[2] = $random % 10 + 15;
      #10;
    end
    $finish;
  end
endmodule
```

### 3.3 Design a parameterized neuron array in Verilog

Verilog code:

```
module neuron_array #(
  parameter N = 4
)(
  input clk,
  input reset,
  input [8*N-1:0] current_in,
  output [N-1:0] spikes
);
  genvar i;
  generate
    for (i = 0; i < N; i = i + 1) begin : neuron_block
      lif_neuron u_neuron (
        .clk(clk),
        .reset(reset),
        .current_in(current_in[i*8 +: 8]),
            .threshold(8'd100),
            .leak_rate(8'd1),
            .refractory_period(8'd10),
            .spike(spikes[i])
        );
    end
  endgenerate
endmodule
```

### 3.4 ReLU Neuron Module

Verilog code:

```
module relu_neuron (
  input clk,
  input reset,
  input signed [7:0] input_current,
  output reg signed [7:0] output_current
);
  always @(posedge clk or posedge reset) begin
    if (reset)
      output_current <= 0;
    else if (input_current < 0)
      output_current <= 0;
    else
      output_current <= input_current;
  end
endmodule
```

## 4. Reflection and Analysis

ChatGPT was able to replicate the Johns Hopkins paper closely. It generated working HDL for all core components and mimicked many of the same bugs described in the original study. Fixes were applied through iterative prompting.

The experience highlighted the importance of prompt engineering, simulation and verification practices, and human-guided iteration. It also demonstrated how LLMs can serve as valuable design co-pilots for hardware engineers.

## 5. Questions Asked and Lessons Learned

| Question | Answer |
|---|---|
| Can ChatGPT generate the complete HDL with no errors? | No, but it gets very close. |
| What kinds of errors appear most often? | Syntax errors (wire vs reg), misunderstanding ports, SystemVerilog features in Verilog. |
| Can it fix its own mistakes when prompted? | Yes, especially for simpler structural or syntax issues. |
| Does it understand HDL architecture and hierarchy? | To a degree — struggles with modular thinking and proper instantiation. |
| Can it produce synthesizable code? | Yes, but requires human verification. |
| Is this useful for beginners? | Only if guided — otherwise debugging might be frustrating. |

## 6. Suggested Improvements

- Add neuron tunability via parameters.

- Add reset/bias functionality.

- Use a different encoding scheme for spikes.

- Modularize SPI protocol logic for reuse.

## 7. Key Learnings

- LLMs are powerful, but need guidance:

ChatGPT can generate complex Verilog modules, but often introduces syntax and logic errors (e.g., improper use of wires, state machine bugs). It required manual debugging and iteration from my end.

- Replication demands critical understanding:

Rather than just copying, replicating the paper involved analyzing logic, debugging LLM outputs, and correcting errors—like it almost came close to its implementation but in a different way.

- Prompt engineering shapes results:

Well-structured and specific prompts led to better HDL generation. Iterative refinement proved essential to correct and **enhance LLM outputs over time and also helped me improve framing of prompts.**

- Going beyond the paper enhances value:

By replacing LIF with ReLU, adding parameterization, and proposing extensions, the project demonstrated how LLMs can support innovation—not just replication.

- Verification is just as crucial as generation:

LLM-generated HDL must be verified through simulation, waveform analysis, and testbenches to catch hidden bugs—something LLMs don't handle independently. **I have not generated testbench for these but will start implementing this strategy for the upcoming weeks**

- Comparisons show both capability and flaws:

ChatGPT could replicate most of the paper's code but repeated many of its errors. Fixes still rely on human HDL knowledge, which **essentially proves that the data on which the current model has been trained is not sufficient to generate bug free RTL**

- Human-in-the-loop is essential:

LLMs accelerate design, but users must guide prompts, fix bugs, and validate functionality. **I believe it is more helpful in cases where commonly available RTL components like counters and registers can be generated with the help of LLMs, and especially valuable in scenarios that require the use of generate blocks and the complex task of connecting modules and ports.**