

# LLM-VeriPPA: Power, Performance, and Area-aware Verilog Code Generation and Refinement with Large Language Models

Kiran Thorat, Jiahui Zhao, Yaotian Liu, Hongwu Peng, Xi Xie, Bin Lei, Jeff Zhang, Caiwen Ding

# Outline

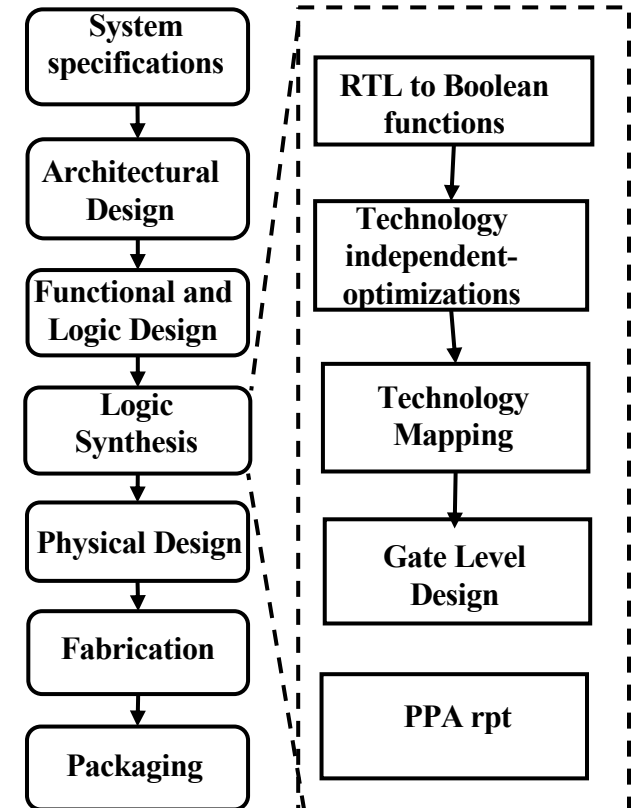
---

- Digital Chip Design
- LLM for EDA
- Related work
- Framework: LLM-VeriPPA
  - Error Rectification
  - Multi-round conversation with Error feedback
- Experiment results
- PPA
  - PPA-optimization
  - PPA Results
- Language construction coverage
- Conclusion

# Digital Chip Design

- Moore's law drives complexity and scalability in modern chip design
- EDA tools used to address the challenges
- EDA- Time consuming, resource intensive, requires domain knowledge.

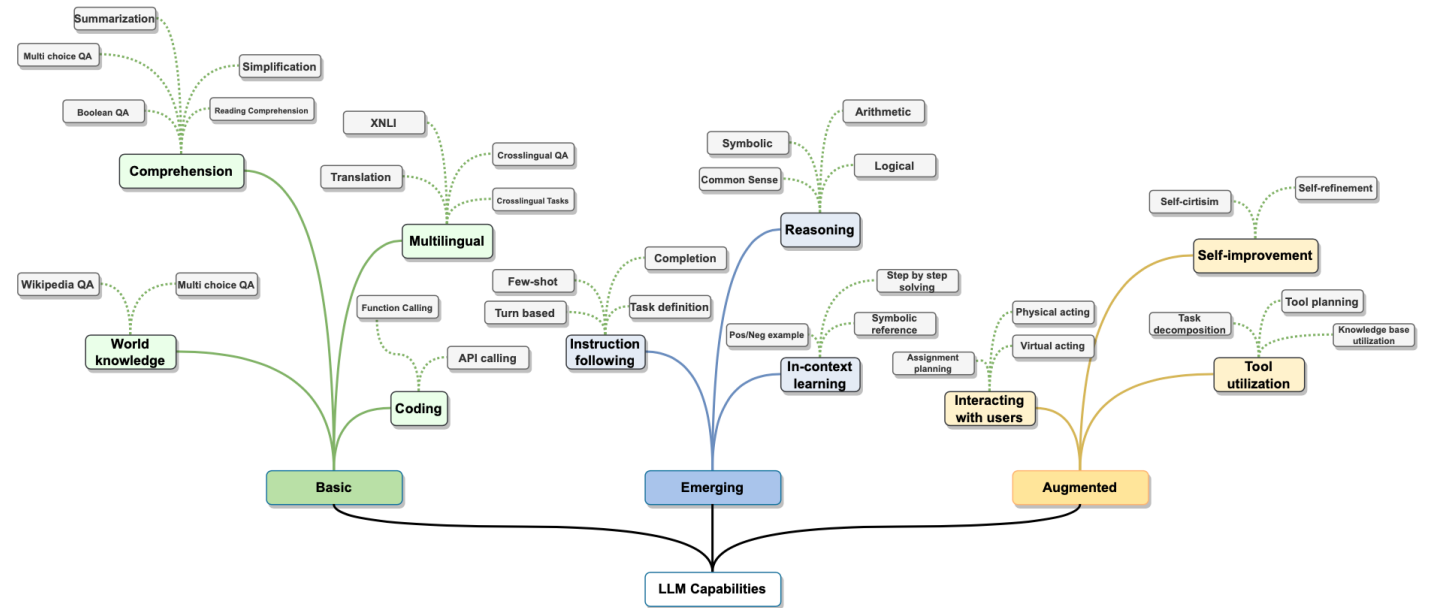
Modern chip design flow



# LLM for EDA

- LLM demonstrated various capabilities [1]

- Comprehension
- Reasoning
- Instruction following
- Coding
- HDL languages:
  - Instruction
  - Intermediate outputs are text format



1. Minaee, Shervin, et al. "Large language models: A survey." *arXiv preprint arXiv:2402.06196* (2024).

# Related Work

---

- RTLLM [2]:
  - RTLLM uses a self-planning process, i.e., designing prompts to ask LLMs to write Verilog and then advice on how to avoid syntax errors to improve correctness.
  - The self-planning does not contain the exact error details to improve the correctness of the generated codes.
  - does not have the capability of PPA optimization to meet the application-specific requirements.
- Verilog Eval [3]:
  - Mainly focus on the dataset generation (156 hardware-designs)
  - Does not provide any PPA results

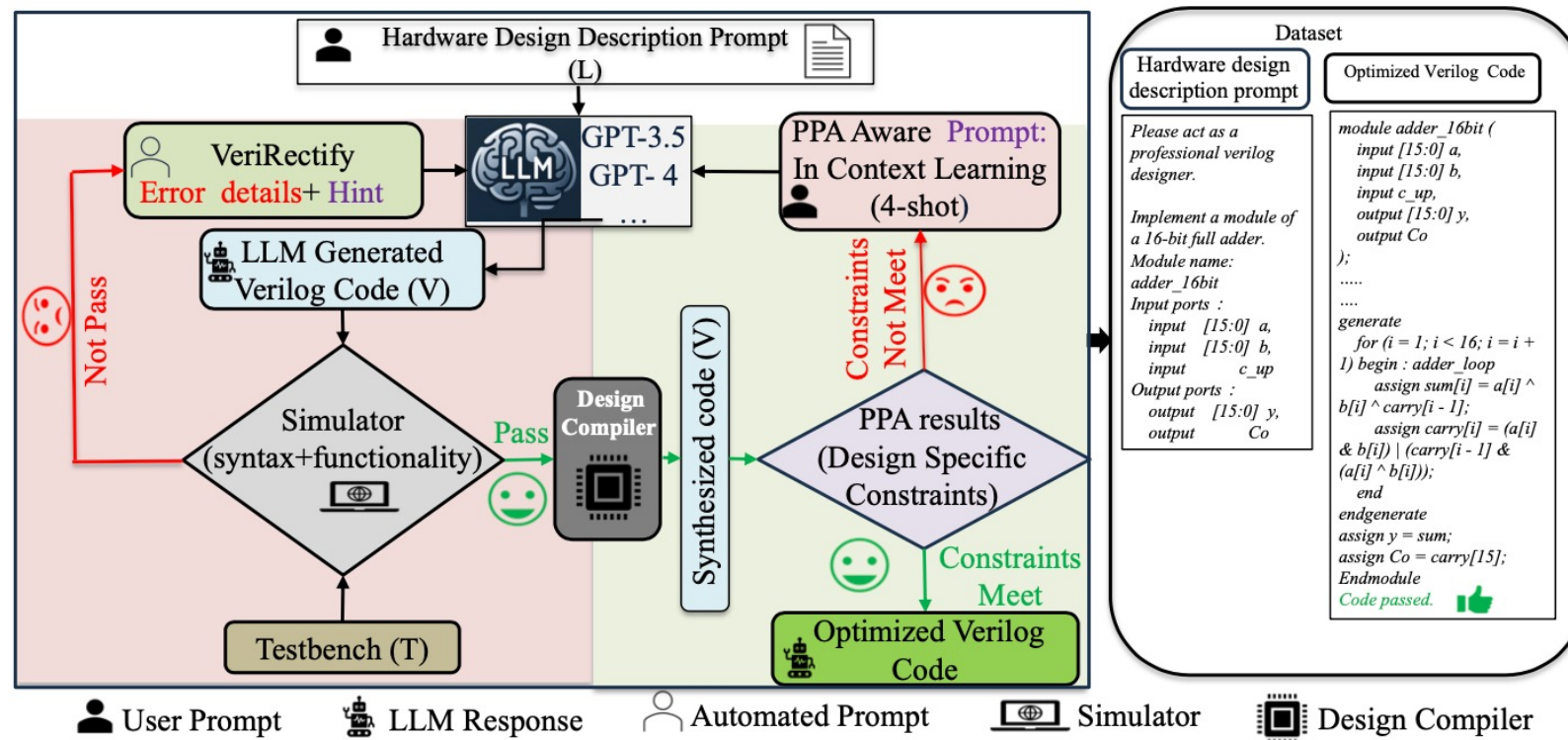
---

2. Lu, Yao, et al. "Rtllm: An open-source benchmark for design rtl generation with large language model." *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2024.

3. Liu, Mingjie, et al. "Verilogeval: Evaluating large language models for verilog code generation." *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 2023

# Framework: Veri-PPA

- Two Stage checks:
  - Syntax and functionality
  - PPA



# Error Rectification Flow

- Error rectification steps:
  - Hardware description prompt
  - LLM generated Verilog code
  - Veri rectify-refinement
  - LLM generated Verilog code

**Hardware Design Description Prompt (L)**

Please act as a professional verilog designer. Implement an 8bit booth-4 multiplier.

During the reset condition (reset signal is high), the module initializes the registers and signals: rdy is set to 0 to indicate that the multiplication operation is not yet complete. p is set to 0 to clear the product register. ctr is set to 0 to reset the counter. multiplier is set to the sign-extended value of the a input, where the most significant bit of a is replicated to the left 8 bits of multiplier. multiplicand is set to the sign-extended value of the b input, where the most significant bit of b is replicated to the left 8 bits of multiplicand.

During the non-reset condition, the module performs the Booth multiplication algorithm: If the counter ctr is less than 16, the module shifts the multiplicand register to the left by 1 bit (multiplicand <= multiplicand << 1). If the current bit of the multiplier at position ctr is 1, the module adds the shifted multiplicand to the p register (p <= p + multiplicand). The counter ctr is incremented by 1 in each clock cycle. Once the counter ctr reaches 16, the multiplication operation is complete, and the rdy signal is set to 1 to indicate readiness.

**Module name:** multi\_booth

**Input ports :** input clk, reset; input [7:0] a, b;

**Output ports :** output [15:0] p; output rdy;

**LLM Generated Verilog Code (V0)**

```

module multi_booth(
    input clk, input reset, input [7:0]
    a,
    input [7:0] b, output [15:0] p,
    output rdy
);
    reg [15:0] p;
    reg [15:0] multiplier;
    reg [15:0] multiplicand;
    reg rdy;
    reg [4:0] ctr;
    always @(posedge clk, posedge
    reset) begin
        if (reset) begin
            ...
        end else begin
            // Non-reset condition
            ...
        end
    endmodule
    
```

**Error details from simulator:**

**Syntax Errors**

multi\_booth\_generated.v:10: error: 'p' has already been declared in this scope.

multi\_booth\_generated.v:6: : It was declared here as a net.

multi\_booth\_generated.v:13: error: 'rdy' has already been declared in this scope.

multi\_booth\_generated.v:7: : It was declared here as a net.

**Functionality Error**

/multi\_booth/testbench.v:67: invalid file descriptor (0x0) given to \$fscanf

**VeriRectify : Error refinement**

**Error Details:**

multi\_booth\_generated.v:10: error: 'p' has already been declared in this scope.

multi\_booth\_generated.v:6: : It was declared here as a net.

multi\_booth\_generated.v:13: error: 'rdy' has already been declared in this scope.

multi\_booth\_generated.v:7: : It was declared here as a net.

multi\_booth/testbench.v:67: invalid file descriptor (0x0) given to \$fscanf.

**Hint:**

Please see the errors in previously generated code. Please write entire code by fixing the errors in previous code. Do not write testbench. Please only give me the code, for anything beside code please properly comment it out.

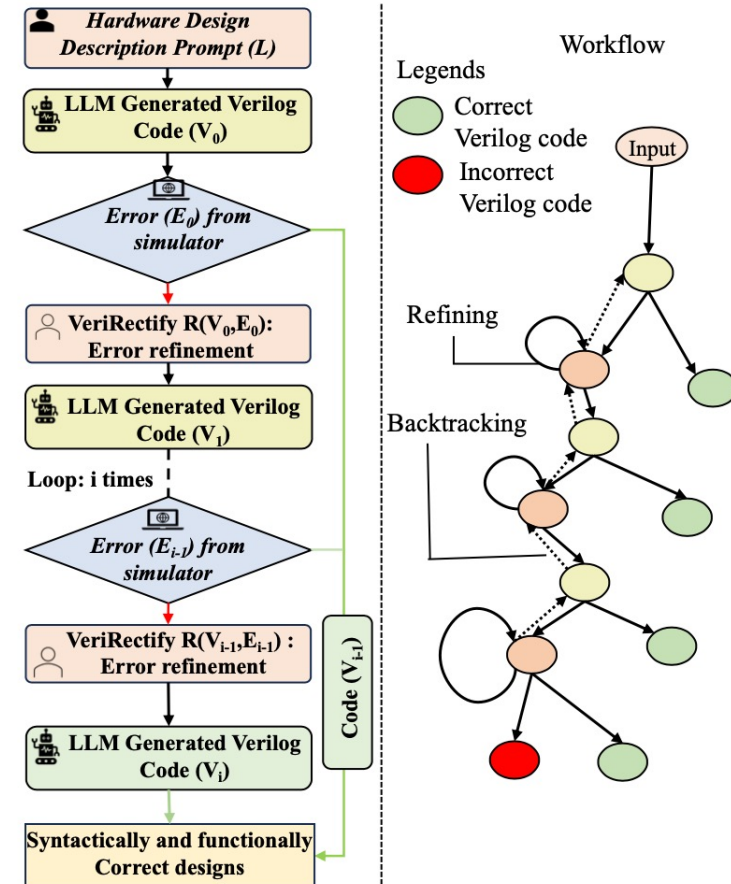
**LLM Generated Verilog Code (V1)**

```

module multi_booth(
    input clk,
    input reset,
    input [7:0] a,
    input [7:0] b,
    output reg [15:0] p,
    output reg rdy
);
    reg [15:0] multiplier;
    reg [15:0] multiplicand;
    reg [4:0] ctr;
    always @(posedge clk, posedge
    reset) begin
        if (reset) begin
            // Reset condition
            rdy <= 0;
            p <= 0;
            ctr <= 0;
            multiplier <= {{8{a[7]}}, a};
            multiplicand <= {{8{b[7]}}, b};
        end else begin
            // Non-reset condition
            if (ctr < 16) begin
                multiplicand <=
                1;
                end
            if (multiplier[ctr] == 1) begin
                p <= p + multiplicand;
            end
            ctr <= ctr + 1;
            if (ctr == 16) begin
                rdy <= 1;
            end end end
        endmodule
    
```

# Multi-round Conversation with Error Feedback

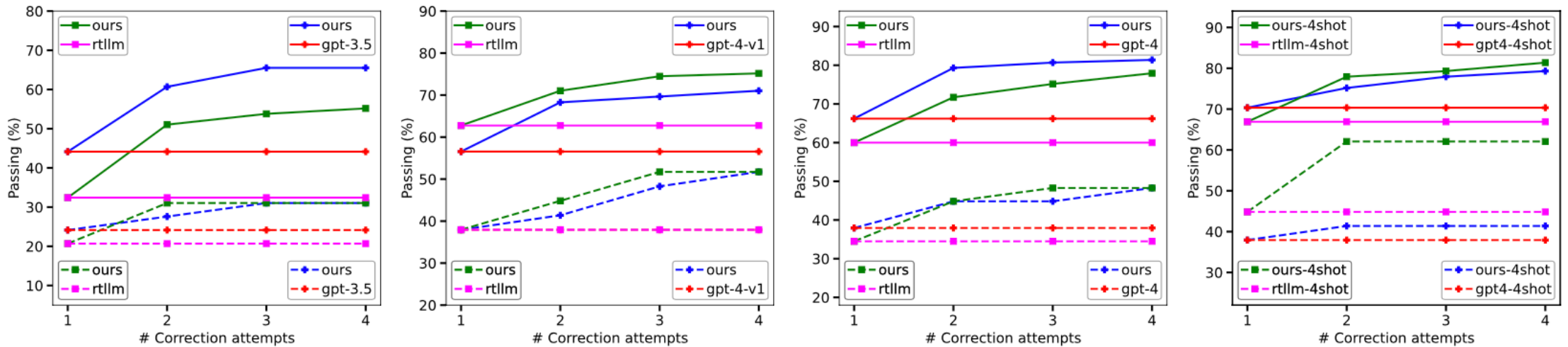
- Process:
  - Hardware description
  - Initial Verilog code and errors ( $V_0, E_0$ )
  - VeriRectify  $R(V_0, E_0)$
  - LLM generated Verilog code ( $V_1$ )
  - Corresponding Error ( $E_1$ )
  - VeriRectify  $R(V_1, E_1)$
  - Loop for until we get correct code or 4 attempts





# Experiment results

- Dataset: RTLLM
  - Commercial-Model: GPT

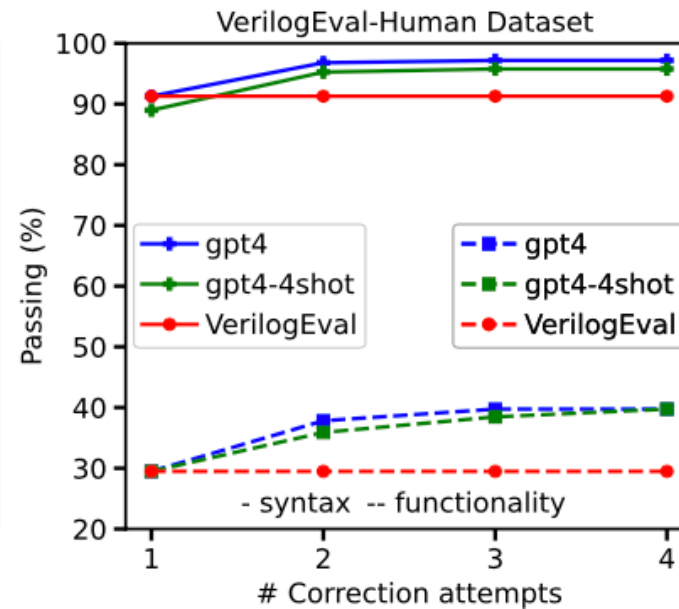
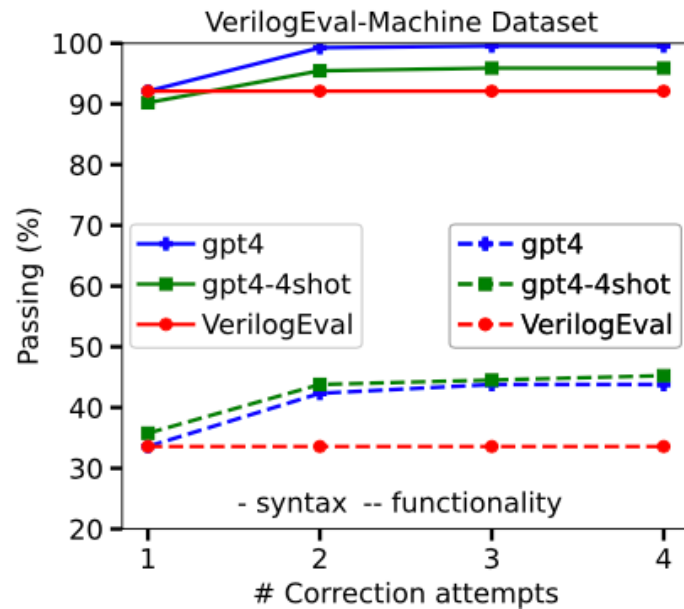


- Open-source: Llama

Model	Without VeriPPA		With VeriPPA	
	Syntax (%)	Functionality (%)	Syntax (%)	Functionality (%)
Llama-2-7B	20.68	0	27.58	0
Llama-3-8B	3.4	0	17.24	0

# Experiment results

- Dataset: Verilog Eval
  - GPT-Model results:



# Power Performance and Area (PPA) in Chip Design

---

- Power: Lowers energy usage and heat, essential for device longevity and performance.
- Performance: Boosts processing speed, crucial for high-demand applications.
- Area: Reduces chip size, cutting costs and enabling more compact designs.
- Need to meet the application specific requirements

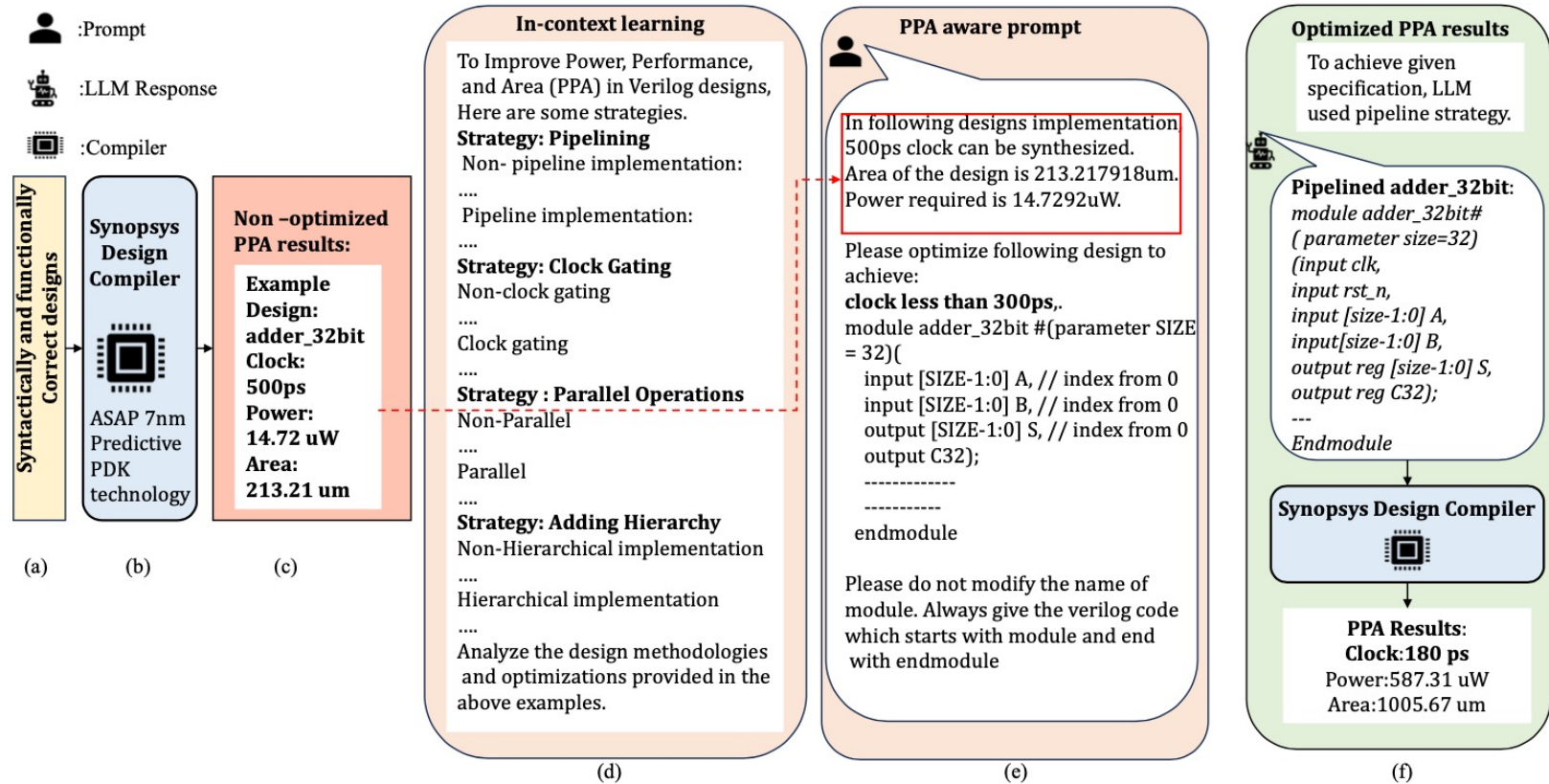
Example: Cryptographic hardware,  
requires fast clock, largely adders.

Adder\_32 bit: 500 ps, 14.7uW,  
213.2  $\mu\text{m}^2$

- PPA needs optimization !

# PPA-Optimizations

- PPA-optimization process
  - Correct designs
  - Non-optimized PPA
  - In-context learning
    - Pipelining
    - Clock Gating
    - Parallel optimization
    - Adding Hierarchy
  - PPA-aware Prompt
  - Optimized PPA results



# Experiment results PPA

- Results

Non-optimized PPA

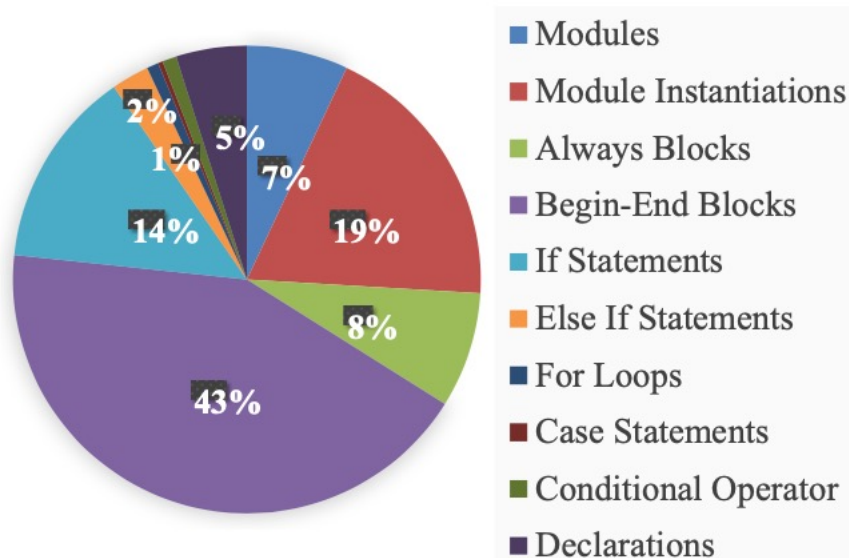
Design Name	GPT-4			GPT-4 (4-shot)		
	Clock (ps)	Power ( $\mu$ W)	Area ( $\mu$ m <sup>2</sup> )	Clock (ps)	Power ( $\mu$ W)	Area ( $\mu$ m <sup>2</sup> )
adder_8bit	318.5	6.3	38.5	333.1	6.1	42.9
adder_16bit	342.2	10.9	104.5	135.1	41.1	152.8
adder_32bit	500.0	14.2	211.6	500.0	14.7	213.2
multi_booth	409.0	112.1	526.0	409.0	112.1	526.0
right_shifter	47.5	144.3	42.9	47.5	144.3	42.9
width_8to16	74.1	223.2	145.8	145.6	128.7	157.2
edge_detect	61.5	49.0	23.3	61.5	49.0	23.3
mux	54.7	215.3	86.1	54.7	215.3	86.1
pe	500.0	552.5	2546.5	500.0	541.0	2488.6
asyn_fifo	295.2	406.4	1279.3	228.3	526.6	1295.4
counter_12	134.4	33.1	40.6	124.5	34.6	36.4
fsm	88.3	32.7	31.5	68.7	49.0	50.2
multi_pipe_4bit	254.7	40.7	131.3	-	-	-
pulse_detect	10.3	187.5	13.5	32.7	59.1	13.5
calendar	-	-	-	208.6	86.6	199.0

Optimized PPA

Design Name	Clock (ps)	Power ( $\mu$ W)	Area ( $\mu$ m)
adder_32bit	180.0	587.31	1005.67
multi_booth	123.2	42.39	42.92
pe	325.0	1206.0	4863.88
asyn_fifo	114.8	988.92	1344.86

# Language construction coverage

- The LLM generated Verilog codes language coverage statistics



Clusters	Labels	Percentage (%)
Timing-Control Dominant	Begin-End Blocks	42.74
	Module Instantiations	18.92
Module-Heavy	Modules	7.01
	Declarations	4.89
Conditional-Intensive	If Statements	13.70
	Else If Statements	2.61
Always-Block-Rich	Always Blocks	7.99
	Conditional Operator	0.98
Complex State Machines	For Loops	0.82
	Case Statements	0.33

# Conclusion

---

- We introduce a novel framework VeriPPA, designed to assess and enhance LLM efficiency in this specialized area
- The first stage focuses on improving the functional and syntactic integrity of the code, while the second stage aims to optimize the code in line with Power-Performance-Area (PPA) constraints, an essential aspect of effective hardware design.
- Our framework achieves 62.0% (+16%) for functional accuracy and 81.37% (+8.3%) for syntactic correctness in Verilog code generation, compared to SOTAs.



**Any Questions**