ASP-DAC 2024
29th Asia and South Pacific Design Automation Conference

ASIA SOUTH PACIFIC DESIGN AUTOMATION CONFERENCE

香港科技大學 THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

Date: Jan. 22-25, 2024

RTLLM: An Open-Source Benchmark for Design RTL Generation with Large Language Model

Yao Lu[*], Shang Liu[*], Qijun Zhang, Zhiyao Xie[†]

Hong Kong University of Science and Technology

yludf@connect.ust.hk, eezhiyao@ust.hk

RTLLM

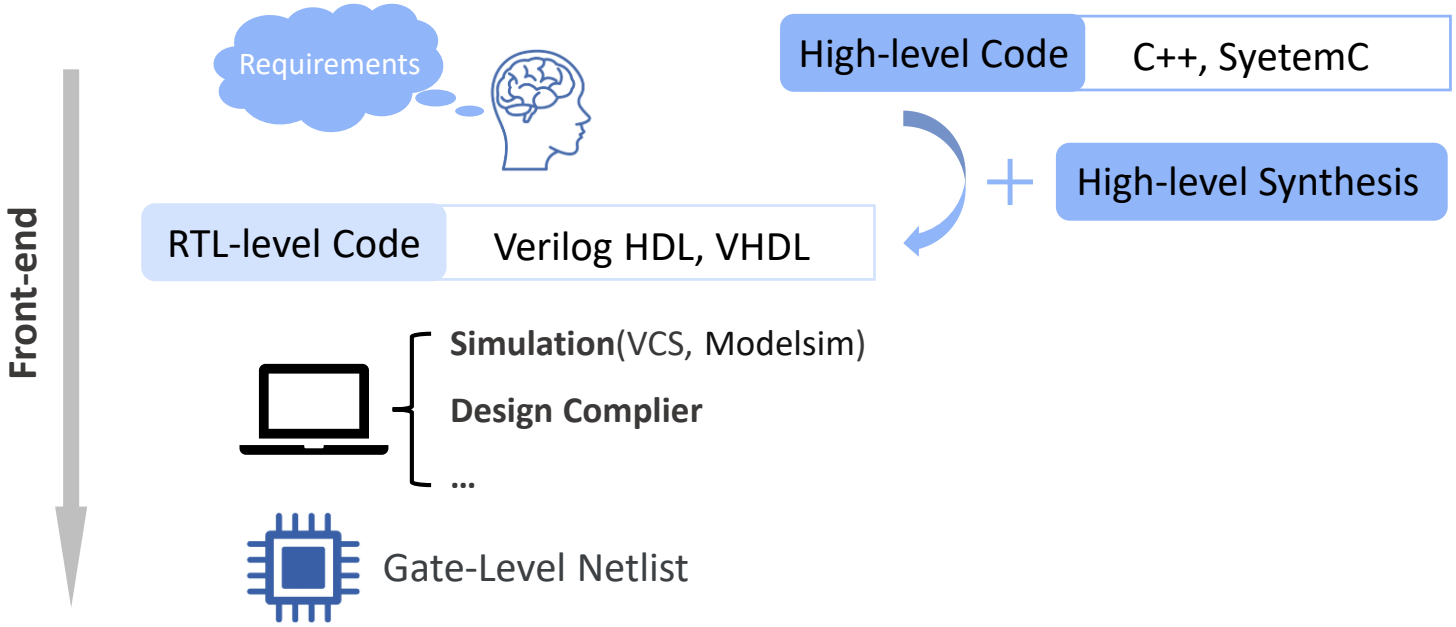
香港科技大學 THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

[*] Equal Contribution [†] Corresponding Author

Outline

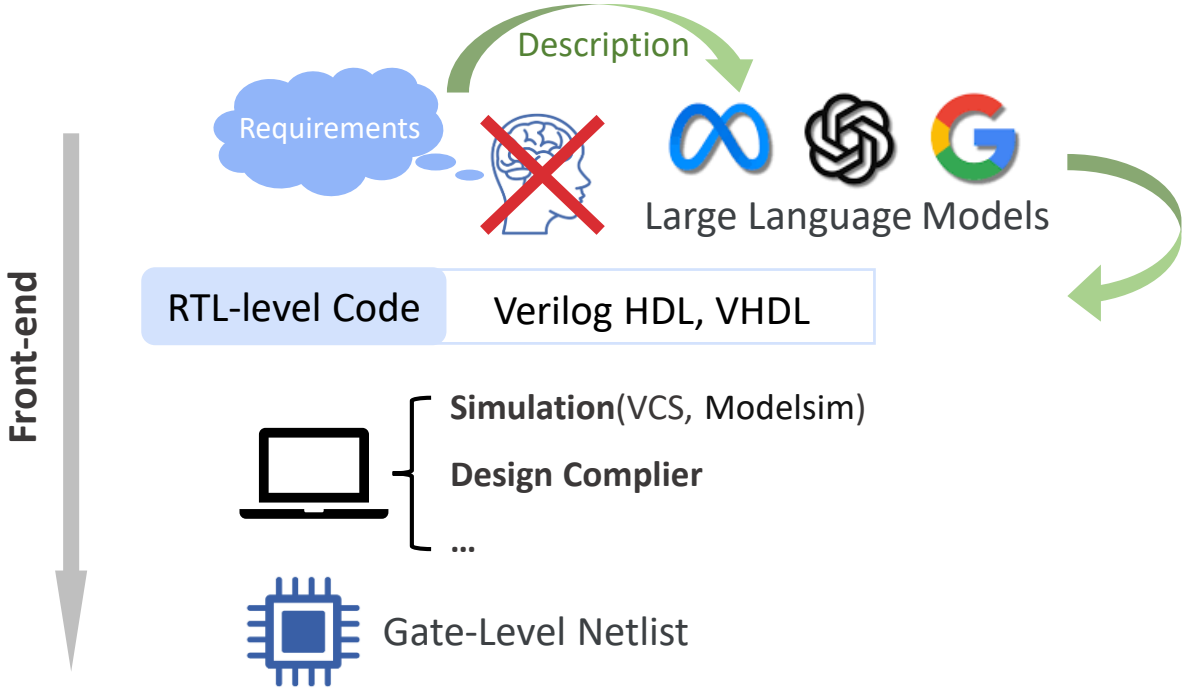
Traditional Hardware Design

- Hardware Description Language (HDL) is used to describe the structure and behavior of digital logic circuits.
  - **Top-Down Electronic Design Automation (EDA) Design Flow**



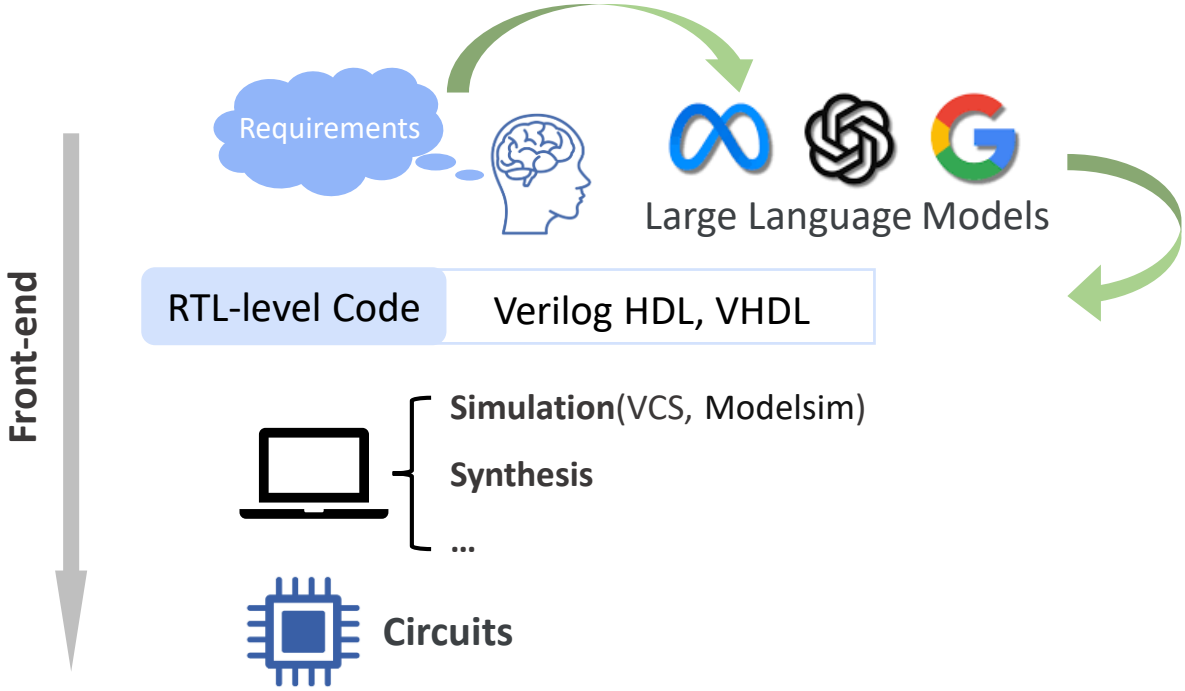
Natural-language-based Design

- Revolution: Adoption of LLMs for hardware design



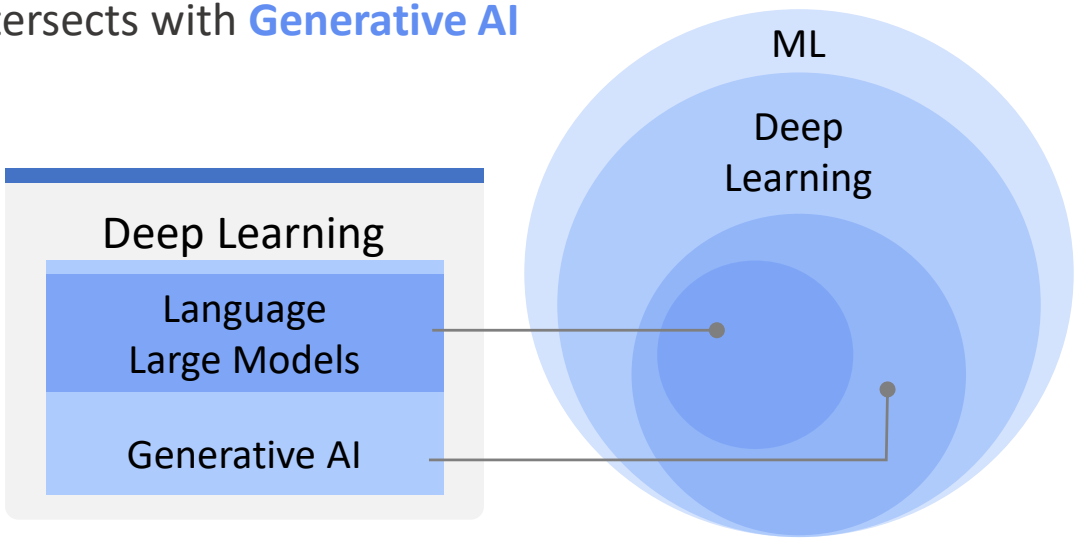
Natural-language-based Design

- Revolution: Adoption of LLMs for hardware design



Large Language Model

- **Large Language Model (LLM)** is a subset of Deep Learning
- LLM also intersects with **Generative AI**

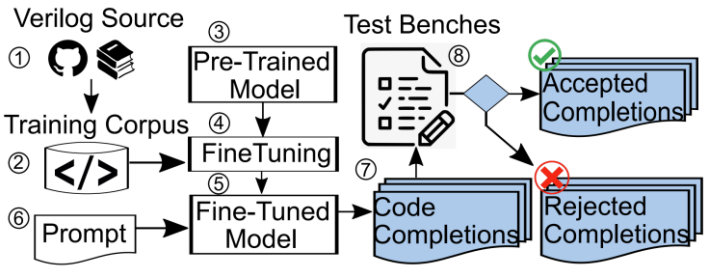


Large, **general-purpose** language models can be **pre-trained** and then **fine-tuned** for specific purposes

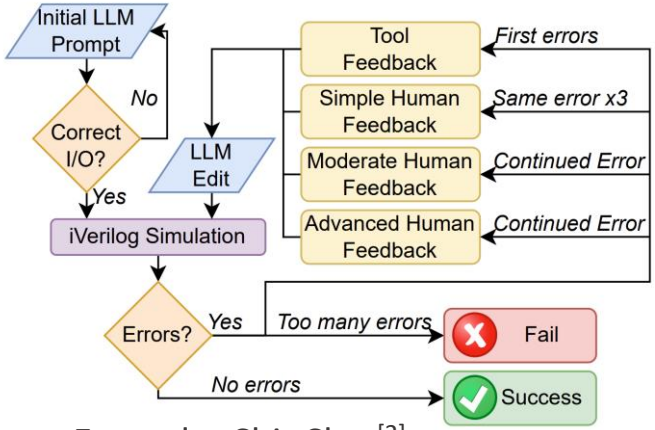
Outline

Large Language Model

- Open-sourced fine-tuned LLMs: StarCoder, CodeGen …
- Commercial Software: ChatGPT 3.5, ChatGPT 4.0…
  - Open-sourced LLMs



  - Example: Thakur et al. [1]
  - GPT



  - Example: ChipChat[2]

The target designs are all relatively simple and in a small circuit scale.

[1] S. Thakur, et al., “Benchmarking large language models for automated verilog rtl code generation,” in DATE, 2023.
[2] J. Blocklove, et al., “Chip-chat: Challenges and opportunities in conversational hardware design,” arXiv:2305.13243, 2023.

Data size of different works

- The statistics of designs evaluated in prior works and in **RTLLM**.

Works	Num of Designs	Num of HDL Lines {Medium, Mean, Max, Total}	Num of Cells in Netlist[1] {Medium, Mean, Max, Total}
Thakur et al. [1]	17	{16, 19, 48, 0.3K}	{9.5, 45, 335, 0.7K}
Chip-Chat [2]	8	{42, 42, 72, 0.3K}	{37, 44, 110, 0.4K}
Chip-GPT [3]	8	Not released to public	
RTLLM	30	{52, 86, 518, 2.5K}	{121, 408, 2435, 11.8K}

- Prior works use **ad-hoc** and simple test designs
- RTLLM is a **more comprehensive** benchmark compared with datasets in prior works.
- A latest work **VerilogEval** from Nvidia provides another benchmark

[3] K. Chang, Y. Wang, H. Ren, M. Wang, S. Liang, Y. Han, H. Li, and X. Li, "Chipgpt: How far are we from natural language hardware design," arXiv:2305.14019, 2023.

Outline

How to Evaluate RTL Generation Performance?

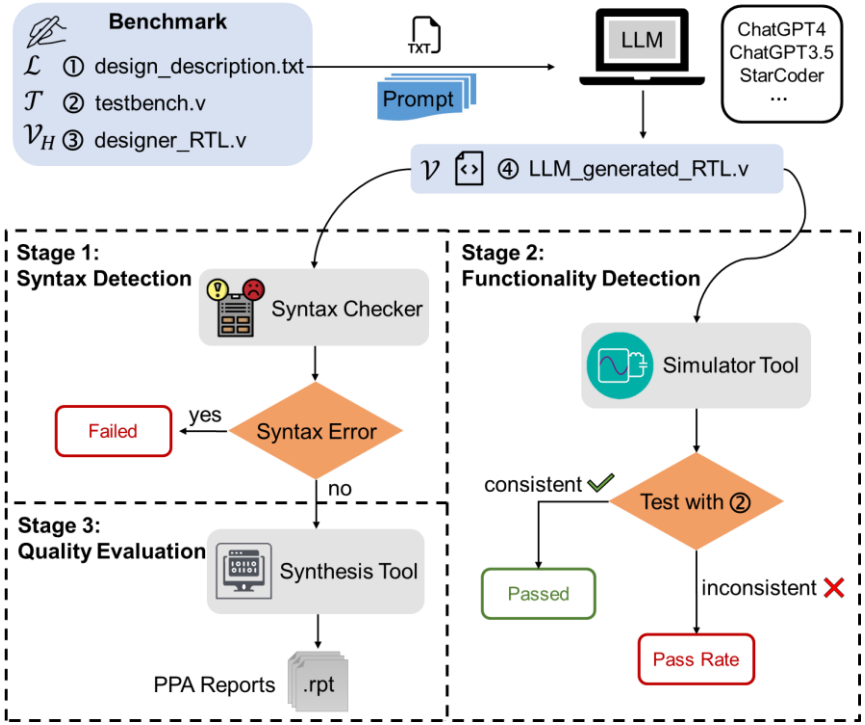
香港科技大學
THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

For a generated RTL design $\mathcal{V}$:

1. Syntax goal
   - The **syntax** of generated RTL design $\mathcal{V}$ should at least be correct
2. Functionality goal
   - The **functionality** of generated RTL design V should be exactly the same as designers' expectation.
3. Quality goal
   - Design **qualities** (e.g., PPA) should also be desirable

Prior works mostly only consider goal 1.

RTLLM: A comprehensive open-source benchmark for design RTL generation



- $\mathcal{L}$: **Description**
  - A natural language description of the target design's functionality.
  - Define module names, I/O names
- $\mathcal{T}$: **Testbench**
  - A testbench with multiple test cases, each with input values and correct output values
- $\mathcal{V}$: **Design to be tested**
  - Automatically generated designs from LLMs
- $\mathcal{V}_H$: **Correct RTL Design**
  - A reference design Verilog hand-crafted by human designers.

RTL Designs in RTLLM

香港科技大學
THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

	Design	Description	Lines of Code
Arithmetic	accu	Accumulates 8-bit data and output after 4 inputs	64
	adder_8bit	An 8-bit adder	26
	adder_16bit	A 16-bit adder implemented with full adders	137
	adder_32bit	A 32-bit carry-lookahead adder	181
	adder_64bit	A 64-bit ripple carry adder based on 4-stage pipeline	197
	multi_8bit	An 8-bit booth-4 multiplier	84
	multi_16bit	An 16-bit multiplier based on shifting and adding operation	65
	multi_pipe_4bit	A 4-bit unsigned number pipeline multiplier	43
	multi_pipe_8bit	An 8-bit unsigned number pipeline multiplier	92
	div_8bit	An 8-bit radix-2 divider	72
	div_16bit	A 16-bit divider based on subtraction operation	45

30 RTL designs

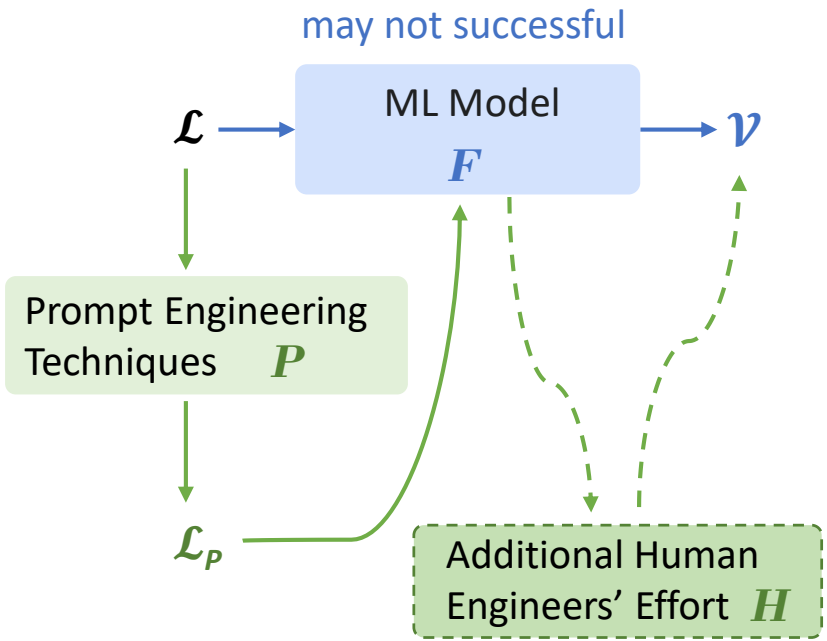
- **11 arithmetic circuits**
- 19 logic circuits

RTL Designs in RTLLM

	Design	Description	Lines of Code
Logic	JC_counter	4-bit Johnson counter with specific cyclic state sequence	22
	right_shifter	Right shifter with 8-bit delay	17
	mux	Multi-bit mux synchronizer	46
	counter_12	Counter module counts from 0 to 12	37
	freq_div	Frequency divider for 100M input clock, outputs 50MHz, 10MHz, 1MHz	51
	signal_generator	Signal generator produces square, sawtooth, and triangular waveforms	52
	serial2parallel	1-bit serial input and output data after receiving 6 inputs	62
	parallel2serial	Convert 4 input bits to 1 output bit	41
	pulse_detect	Extract pulse signal from the fast clock and create a new one in the slow clock	38
	edge_detect	Detect rising and falling edges of changing 1-bit signal	39
	FSM	FSM detection circuit for specific input	77
	width_8to16	First 8-bit data placed in higher 8-bits of the 16-bit output	50
	traffic_light	Traffic light system with three colors and pedestrian button	106
	calendar	Perpetual calendar with seconds, minutes, and hours	37
	RAM	8x4 bits true dual-port RAM	50
	asyn_fifo	An asynchronous FIFO 16×8 bits	149
	ALU	An ALU for 32bit MIPS-ISA CPU	111
	PE	A Multiplying Accumulator for 32bit integer	27
	risc_cpu	Simplified RISC_CPU with clock generator, instruction register, accumulator, arithmetic logic unit, data controller, state controller, etc.	518

- **19 logic circuits**

RTLLM: A comprehensive open-source benchmark for design RTL generation



- $\mathcal{V} = F(\mathcal{L})$
  - Given a natural language description of desired design functionality $\mathcal{L}$
  - The target is to develop an ML model F to generate the RTL of this design $\mathcal{V}$
- $\mathcal{L}_P = P(\mathcal{L})$
  - Apply prompt engineering techniques P to revise $\mathcal{L}$, generating $\mathcal{L}_P$
- $\mathcal{V} = H(F(\mathcal{L}_P))$
  - LLM output may be further manually revised by human engineers H

Self-planning Technique

香港科技大學
THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

Enhancing LLM Capabilities and streamlining RTL Design Generation

- Highly effective prompt engineering technique
- **Two-step** process for design RTL generation
- **No additional human efforts** or design data needed

Step 1: Planning the Design

- **LLM instructed** to plan the target design $\mathcal{V}$
- Input is the original description $\mathcal{L}$
- Output includes **natural language-level analysis and reasoning,** $\mathcal{L}_P$
- Model identifies **syntax errors to avoid**

Step 2: Final Design Generation

- Using the detailed design description $\mathcal{L}_P$
- Generates the final design RTL $\mathcal{V}$
- **Reduced error-prone design RTL** compared to direct generation

Self-planning Technique

香港科技大學
THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

Sample – Step 1

- We request the LLM to first generate
  - the reasoning steps **(line 5)**
  - advice to avoid errors **(line 6)**

```
1 #Implement the design of unsigned 16bit multiplier
      based on shifting and adding opration.
2 module multi_16bit(
3 // ...I/O details omitted...
4 );
5 #Please act as a professional verilog designer, try
      to understand the requirements above and give
      reasoning steps in natural language to achieve
      it.
6 #In addition, try to give advice to avoid syntax
      error.
```

Part of the **input** of the first step from self-planning

```
1  The task is to implement ... The modules has an ...
2  To achieve this, we will use a state machine to
       control the multiplication process. The steps
       involved are as follows:
3  IDLE: The initial state where the module waits for
       the start signal to be asserted.
4  MULTIPLY: The state where the multiplication
       operation is performed.
5  FINISH: The final state indicating the
       multiplication operation is complete.
6  Create the necessary registers to hold the state,
       the inputs (ain and bin), and the intermediate
       results during the multiplication process.
7  // ...details omitted...
8  Potential Syntax Error Advice:
9  Ensure that all the signals ... are declared ...
10 Make sure to use blocking ... non-blocking ...
11 Ensure that you use the correct parameter size for
       states. Since we have three states, we need at
       least two bits to represent them ...
12 Check for any typos or misspellings ...
```

Part of the **output** of the first step from self-planning

Self-planning Technique

香港科技大學
THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

Samples – Step 2

- Wrong RTL designs generated by GPT-3.5 **without self-planning**
  - multi_16bit

```
module multi_16bit(
// ...I/O details omitted...
);
// ...details omitted...
always @(posedge clk or negedge rst_n) begin
    integer i;
    reg [15:0] a_shifted;
    for (i = 0; i < 16; i = i + 1) begin
        // ......
    end
end
// ......
```

Syntax error: It wrongly defines new variables in the always block.

- After using self-planning with GPT-3.5, the syntax error in this example is **solved**.

Self-planning Technique

香港科技大學
THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

Samples – Step 2

- Wrong RTL designs generated by GPT-3.5 **without self-planning**
  - adder_32bit

```
module adder_32bit(
// ...I/O details omitted...
);
// ...details omitted...
    for (i = 1; i <= 32; i = i + 1) begin :
    cla_block
        wire P, G, carry;
        //......
        assign carry = (i == 1) ? 1'b0 : (P & {
    carry, G});
        assign S[i] = A[i] ^ B[i] ^ {carry, G};
        //......
    end
end
```

Functionality error: Reflected in its usage of the ***carry*** variable.

- After using self-planning with GPT-3.5, both the **syntax error** and the **functionality error** in two examples are **solved**.

Outline

Experiment

- **Our proposed RTL generation benchmark**
  - 11 arithmetic circuits
  - 19 logic circuits
- Logic synthesis with **Synopsys Design Compiler®**
- Functionality verification, the RTL simulation with **Synopsys VCS®**

Experiment

- Evaluated **4 LLMs** with our proposed RTL generation benchmark
  - **GPT-3.5**: the free commercial solution.
  - **GPT-4**: the state-of-the-art commercial solution.
  - **Thakur et al.**: an academic model with 16 billion parameters developed by fine-tuning the **CodeGen** model with Verilog data.
  - **StarCoder**: a recent general academic model with 15 billion parameters for code generation, without being fine-tuned for Verilog.
  - **GPT-3.5 + self-planning**: adopting our proposed self-planning technique when using GPT-3.5.
  - **GPT-4 + self-planning**: adopting our proposed self-planning technique when using GPT-3.5.

Syntax and Functionality Results

香港科技大學
THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

- Syntax and Functionality Correctness Verification for Different LLMs

Clear metric for assessing

  - Quantitative assessment of syntax correctness
  - Counts the number of generated design RTLs $\mathcal{V}$ with correct **syntax** out of **5 trials**
  - The **Func.** column counts a success ✔ if **at least one** generated RTL passes the testbench $\mathcal{T}$
  - Functionality evaluation is based on designs with correct syntax

Syntax and Functionality Results

- The Syntax and Functionality Correctness Verification for Different LLMs

Part I – Logic Circuits (total 19 designs)

Design	GPT-3.5		GPT-4		Thakur et al. [5]		StarCoder [10]		GPT-3.5 + SP		GPT-4 + SP	
	Syntax	Func.	Syntax	Func.	Syntax	Func.	Syntax	Func.	Syntax	Func.	Syntax	Func.
JC_counter	5	✔	5	✔	5	✘	5	✘	4	✔	5	✔
right_shifter	5	✔	5	✔	5	✔	0	-	5	✔	5	✔
mux	0	-	4	✔	5	✔	0	-	4	✘	5	✔
counter_12	5	✔	5	✔	5	✘	5	✘	4	✔	5	✔
freq_div	5	✘	5	✘	5	✘	0	-	5	✘	5	✔
signal_generator	5	✔	5	✔	0	-	5	✘	5	✔	5	✔
serial2parallel	5	✔	5	✔	0	-	5	✘	5	✔	4	✔
parallel2serial	5	✘	4	✘	0	-	0	-	3	✘	5	✘
pulse_detect	1	✘	5	✘	5	✘	0	-	5	✘	5	✘
edge_detect	5	✔	5	✔	5	✘	0	-	5	✔	5	✔
FSM	5	✘	5	✘	5	✘	0	-	5	✘	5	✘
width_8to16	5	✔	5	✔	0	-	5	✔	5	✔	5	✔
traffic_light	5	✘	5	✔	0	-	0	-	5	✘	5	✔
calendar	0	-	5	✘	0	-	0	-	5	✔	5	✔
RAM	0	-	0	-	5	✔	5	✔	0	-	3	✔
asyn_fifo	0	-	0	-	0	-	0	-	2	✘	0	-
ALU	0	-	5	✘	0	-	0	-	0	-	5	✔
PE	4	✔	5	✔	5	✘	5	✔	5	✔	4	✔
risc_cpu	0	-	0	-	0	-	0	-	0	-	0	-
Total Success rate	55%	10/30	81%	15/30	40%	5/30	27%	5/30	73%	14/30	90%	19/30

Syntax and Functionality Results

- The Syntax and Functionality Correctness Verification for Different LLMs

Part II – Arithmetic Circuits (total 11 designs)

Design	GPT-3.5		GPT-4		Thakur et al. [5]		StarCoder [10]		GPT-3.5 + SP		GPT-4 + SP	
	Syntax	Func.	Syntax	Func.	Syntax	Func.	Syntax	Func.	Syntax	Func.	Syntax	Func.
accu	4	✔	5	✔	0	-	0	-	4	✔	5	✔
adder_8bit	4	✔	5	✔	0	-	0	-	4	✔	5	✔
adder_16bit	5	✘	5	✔	5	✔	0	-	5	✔	5	✔
adder_32bit	5	✘	5	✘	0	-	0	-	5	✔	5	✘
adder_64bit	2	✘	3	✘	0	-	0	-	4	✘	5	✘
multi_8bit	3	✘	4	✘	0	-	0	-	5	✘	5	✘
multi_16bit	0	-	5	✔	5	✔	0	-	2	✔	2	✔
multi_pipe_4bit	0	-	2	✔	0	-	5	✔	1	✘	5	✘
multi_pipe_8bit	0	-	4	✘	0	-	5	✔	3	✘	4	✔
div_8bit	0	-	0	-	0	-	0	-	4	✘	0	-
div_16bit	0	-	5	✘	0	-	0	-	0	-	5	✘
Total Success rate	55%	10/30	81%	15/30	40%	5/30	27%	5/30	73%	14/30	90%	19/30

- **The performance rank is :**

  GPT-4 + self-planning > GPT-4 > GPT-3.5 + self-planning > GPT-3.5 > Thakur et al.

Design Qualities Results

- Evaluation of design qualities: **power, timing, and area**

  Clear metric for assessing

  - Syntax correctness as a prerequisite
  - **Logic synthesis** for designs with correct syntax
  - Color coding to identify different categories：
    - **Green** for the Best Qualities
    - **Red** for the Worst Qualities

Design Qualities Results

- This table summarizes the design qualities of generated RTL from different LLMs

Design	Designer Reference ($\mathcal{V}_H$)			ChatGPT-3.5			ChatGPT-4			Thakur et al. [1]			GPT-3.5 + Self-planning		
	Area (μm^2)	Power (μW)	Timing (ns)	Area (μm^2)	Power (μW)	Timing (ns)	Area (μm^2)	Power (μW)	Timing (ns)	Area (μm^2)	Power (μW)	Timing (ns)	Area (μm^2)	Power (μW)	Timing (ns)
accu	239	19K	-0.42	298	24K	-0.43	304	21K	-0.39	-	-	-	231	18K	-0.37
adder_8bit	65	34	-0.62	38	14	-0.14	15	5.8	-0.12	-	-	-	74	42	-0.63
adder_16bit	128	68	-1.21	157	91.0	-0.33	126	68	-1.19	189	106	-0.31	163	94	-0.33
adder_32bit	571	298	-0.72	58	17	-0.04	65	26	-0.13	-	-	-	337	199	-0.43
adder_64bit	2.9K	296K	-0.48	2.5K	242K	-0.60	2.4K	187K	-0.48	-	-	-	2.3K	220K	-0.32
multi_8bit	52	6.1K	-0.08	640	45K	-0.43	494	33K	-0.49	-	-	-	259	23K	-0.27
multi_16bit	749	75K	-0.91	-	-	-	531	79K	-0.50	7.5K	384K	-1.76	-	-	-
multi_pipe_4bit	198	19K	-0.34	-	-	-	193	22K	-0.33	-	-	-	146	17K	-0.30
multi_pipe_8bit	961	78K	-0.65	-	-	-	1.1K	80K	-0.99	-	-	-	443	42K	-0.14
div_8bit	158	8.4K	-0.38	-	-	-	-	-	-	-	-	-	-	-	-
div_16bit	1.8K	2.4K	-4.20	-	-	-	1.5K	1.8K	-4.84	-	-	-	-	-	-
JC_counter	380	45K	-0.13	380	45K	-0.13	42	4.7K	-0.26	29	4.6K	-0.23	195	21K	-0.22
right_shifter	42	4.2	-0.14	40	3.8K	-0.12	46	5.7K	-0.13	40	3.8K	-0.12	40	3.8K	-0.12
mux	68	6.5	-0.08	-	-	-	90	9.5	-0.08	64	13	-0.08	144	14	-0.08
counter_12	49	4.3K	-0.31	79.0	8.0K	-0.25	46	4.4K	-0.26	35	4.0K	-0.24	76	8.4K	-0.26
freq_div	124	16K	-0.29	911	66K	-0.45	118	16K	-0.32	226	16K	-0.4	667	53K	-0.41
signal_generator	178	14K	-0.36	72	9.2K	-0.23	98	11K	-0.26	-	-	-	101	11K	-0.27
serial2parallel	135	13K	-0.29	168	16K	-0.30	100	9.8K	-0.28	-	-	-	155	14K	-0.33
parallel2serial	55	8.6K	-0.23	35	6.2K	-0.21	20	3.8K	-0.19	-	-	-	1.06	0	0
pulse_detect	25	2.8	-0.13	42	2.8	-0.12	40	4.3	-0.08	25	2.8	-0.12	28	3.4	-0.08
edge_detect	19	2.6K	-0.14	24	3.3K	-0.16	19	2.6K	-0.14	1.06	0	0	19	2.6K	-0.14
FSM	44	3.5K	-0.18	26	2.7K	-0.21	34	2.7K	-0.25	27	2.7K	-0.24	45	4.1K	-0.2
width_8to16	219	23K	-0.26	214	21K	-0.20	219	23K	-0.26	-	-	-	144	14K	0.24
traffic_light	178	18K	-0.35	147	14K	-0.34	138	11K	-0.38	-	-	-	-	-	-
calendar	199	16K	-0.36	-	-	-	460	31K	-0.51	-	-	-	227	16K	-0.37
RAM	3.5K	248K	-0.35	-	-	-	-	-	-	353	27K	-0.26	-	-	-
asyn_fifo	1.3K	107	-0.23	-	-	-	-	-	-	-	-	-	0	0	0
ALU	2.4K	1.0K	-0.76	-	-	-	3.3K	1.4K	-0.71	-	-	-	-	-	-
PE	2.4K	363K	-1.03	2.5K	359K	-1.08	2.6K	366K	-1.06	2.2K	275K	-0.07	2.5K	358K	-1.08
risc_cpu	634	6.2K	-0.30	-	-	-	-	-	-	-	-	-	-	-	-
Best Quality Num	3	7	5	2	2	5	8	5	6	2	1	2	5	7	5

Outline

1. Introduction
2. Background
3. RTLLM
4. Evaluation
5. **Discussion**

Conclusion

香港科技大學
THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

- Proposed three reasonable metrics for auto RTL generation
- Proposed **a more comprehensive open-source benchmark** for design RTL generation
- Benchmark **features**: a reasonable flow, larger dataset, increased design complexity
- Reported the performance of existing LLM solutions
- **Effective** self-planning prompt engineering technique

Future Directions

香港科技大學
THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

- Ongoing expansion and maintenance of the benchmark
- Continued **validation and refinement** of self-planning
- **Fine-tuning open-source models** for improved RTLLM benchmark performance

We have updated RTLLM 1.1, and RTLLM 2.0 is coming soon…

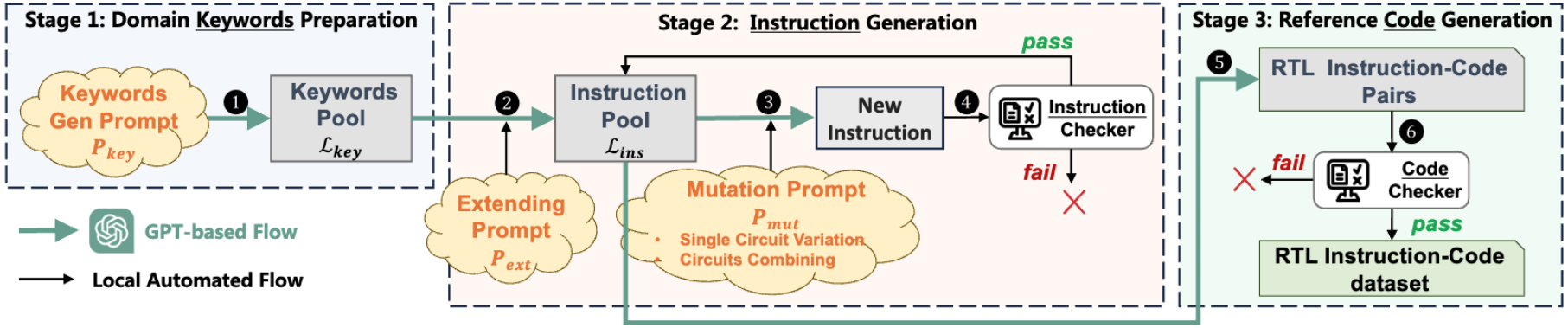
Our Lastest Work:

- **RTL Code Generation Model** with the **fully open-source dataset** and **lightweight solution**
  - Generate a new ‘circuit design’ dataset with >10K samples
  - Only 7B parameters
  - Outperforming GPT-3.5, can be used locally, without privacy concerns
  - Further quantized into 4 bits, consumes only 4GB of memory
  - Will be fully open-sourced

Our Lastest Work:

Design	GPT-3.5		GPT-4		Thakur et al. [17]		StarCoder [7]		RTLCoder		RTLCoder-4bit	
	Syntax	Func.	Syntax	Func.	Syntax	Func.	Syntax	Func.	Syntax	Func.	Syntax	Func.
accu	4	✔	5	✔	0	-	0	-	4	✘	3	✘
adder_8bit	4	✔	5	✔	0	-	0	-	5	✔	4	✔
adder_16bit	5	✘	5	✔	5	✔	0	-	5	✔	5	✔
adder_32bit	5	✘	5	✘	0	-	0	-	4	✘	3	✘
adder_64bit	2	✘	3	✘	0	-	0	-	2	✘	1	✘
multi_8bit	3	✘	4	✘	0	-	0	-	1	✘	4	✔
multi_16bit	0	-	5	✔	5	✔	0	-	0	✘	0	✘
multi_pipe_4bit	0	-	2	✔	0	-	5	✔	3	✔	4	✘
multi_pipe_8bit	0	-	4	✘	0	-	5	✔	3	✔	4	✔
div_8bit	0	-	0	-	0	-	0	-	3	✔	1	✔
div_16bit	0	-	5	✘	0	-	0	-	0	-	0	-
.....												
edge_detect	5	✔	5	✔	5	✘	0	-	5	✔	4	✘
FSM	5	✘	5	✘	5	✘	0	-	5	✘	4	✘
width_8to16	5	✔	5	✔	0	-	5	✔	5	✘	0	-
traffic_light	5	✘	5	✔	0	-	0	-	3	✔	2	✔
calendar	0	-	5	✘	0	-	0	-	5	✘	3	✘
RAM	0	-	0	-	5	✔	5	✔	5	✘	5	✘
asyn_fifo	0	-	0	-	0	-	0	-	0	-	0	-
ALU	0	-	5	✘	0	-	0	-	1	✘	0	-
PE	4	✔	5	✔	5	✘	5	✔	5	✔	5	✔
risc_cpu	0	-	0	-	0	-	0	-	0	-	0	-
Success rate	63%	10/30	87%	15/30	40%	5/30	27%	5/30	83%	11/30	80%	9/30

Our Lastest Work:



Our proposed automated **dataset generation flow**

- **For further details regarding this project, please visit and follow our lab's GitHub page! You can star the repo you interested.**

HKUST Zhiyao

香港科技大學
THE HONG KONG
UNIVERSITY OF SCIENCE
AND TECHNOLOGY

THANK YOU!

HKUST
Zhiyao