

Generative AI for Design and Verification

Haoxing (Mark) Ren, Director of Design Automation Research, NVIDIA

LLM Applications for Chip Design and Verification

Coding Assistance → Generating design and verification code (RTL, SVA, testbenches, EDA scripts, tools scripts, and configs)

- Generate code from specifications
- Generate code for auxiliary design tasks such as assertions, comments, etc.
- Generate lower-level programs from higher-level descriptions
- Generate scripts for specific tasks (VLSI, Verification)
- Transform code for more efficient implementation

Know-how Assistance → Generating insights, knowledge, and ideas. Answer questions about designs, infrastructures, tools, flows, HW domains, etc.

Analysis Assistance → Summarization, report/log analysis, visualization of design and related data, check rule violations, etc.

Debug Assistance → Triage and fix a design problem, e.g. regression failures for logic design and timing failures for physical design.

Optimization Assistance → Optimize for design and verification: coverage closure, recipe optimization, architecture exploration, etc.

NVIDIA



Agenda

LLM for Verilog and SVA Generation

- Benchmarks
- Enhancing LLM capability

Challenges and opportunities

VerilogEval

https://github.com/NVlabs/verilog-eval

156 Verilog design problems converted from HDLBits (https://hdlbits.01xz.net/wiki/Problem_sets)

Covering Verilog concepts

- **Language**: Basics, Vectors, Hierarchy Modules, Procedures
- **Combinational Logic**: Basic gates, multiplexers, arithmetic, Karnaugh map
- **Sequential logic**: Latch/flops, counters, shift registers, cellular automata, FSM
- **Larger circuit**: counter, shift register and FSM applications
- **Verification**: build circuit from waveforms

Prompting variations

NVIDIA.

Simple example

Question

I would like you to implement a module named TopModule with the following interface. All input and output ports are one bit unless otherwise specified.

- input in (1024 bits)
- input sel (8 bits)
- output out (4 bits)

The module should implement a 4-bit wide, 256-to-1 multiplexer. The 256 4-bit inputs are all packed into a single 1024-bit input vector. sel=0 should select bits in[3:0], sel=1 selects bits in[7:4], sel=2 selects bits in[11:8], etc.

Answer

```
module RefModule (
  input [1023:0] in,
  input [7:0] sel,
  output [3:0] out
);

  assign out = {in[sel*4+3], in[sel*4+2], in[sel*4+1], in[sel*4+0]};

endmodule
```

Complex example

I would like you to implement a module named TopModule with the following interface. All input and output ports are one bit unless otherwise specified.

- input clk
- input reset
- input data
- output count (4 bits)
- output counting
- output done
- input ack

The module should implement a timer with one input that:

(1) is started when a particular input pattern (1101) is detected,
(2) shifts in 4 more bits to determine the duration to delay,
(3) waits for the counters to finish counting, and
(4) notifies the user and waits for the user to acknowledge the timer.

The serial data is available on the data input pin. When the pattern 1101 is received, the circuit must then shift in the next 4 bits, most-significant-bit first. These 4 bits determine the duration of the timer delay, referred to as delay[3:0]. After that, the state machine asserts its counting output to indicate it is counting. Once the 1101 and delay[3:0] have been read, the circuit no longer looks at the data input until it resumes searching after everything else is done.

The state machine must count for exactly (delay[3:0] + 1) * 1000 clock cycles. e.g., delay=0 means count 1000 cycles, and delay=5 means count 6000 cycles. Also output the current remaining time. This should be equal to delay for 1000 cycles, then delay-1 for 1000 cycles, and so on until it is 0 for 1000 cycles.

When the circuit isn't counting, the count[3:0] output is don't-care (whatever value is convenient for you to implement). At that point, the circuit must assert done to notify the user the timer has timed out, and waits until input ack is 1 before being reset to look for the next occurrence of the start sequence (1101).

The circuit should reset into a state where it begins searching for the input sequence 1101. The reset signal is active high synchronous. Assume all sequential logic is triggered on the positive edge of the clock.

NVIDIA

Complex example

```
module RefModule (
  input wire clk,
  input wire reset,
  input wire data,
  output wire [3:0] count,
  output reg counting,
  output reg done,
  input wire ack
);

  typedef enum logic[3:0] {
    S, S1, S11, S110, B0, B1, B2, B3, Count, Wait
  } States;

  States state, next;

  reg shift_ena;
  reg [9:0] fcount;
  reg [3:0] scount;
  wire done_counting = (scount == 0) && (fcount == 999);

  always_comb begin
    case (state)
      S: next = States'(data ? S1: S);
      S1: next = States'(data ? S11: S);
      S11: next = States'(data ? S11 : S110);
      S110: next = States'(data ? B0 : S);
      B0: next = B1;
      B1: next = B2;
      B2: next = B3;
      B3: next = Count;
      Count: next = States'(done_counting ? Wait : Count);
      Wait: next = States'(ack ? S : Wait);
      default: next = States'(4'bx);
    endcase
  end

  always @(posedge clk) begin
    if (reset) state <= S;
    else state <= next;
  end

  always_comb begin
    shift_ena = 0; counting = 0; done = 0;
    if (state == B0 || state == B1 || state == B2 || state == B3)
      shift_ena = 1;
    if (state == Count)
      counting = 1;
    if (state == Wait)
      done = 1;
    if (|state === 1'bx) begin
      {shift_ena, counting, done} = 'x;
    end
  end

  // Shift register
  always @(posedge clk) begin
    if (shift_ena)
      scount <= {scount[2:0], data};
    else if (counting && fcount == 999)
      scount <= scount - 1'b1;
  end

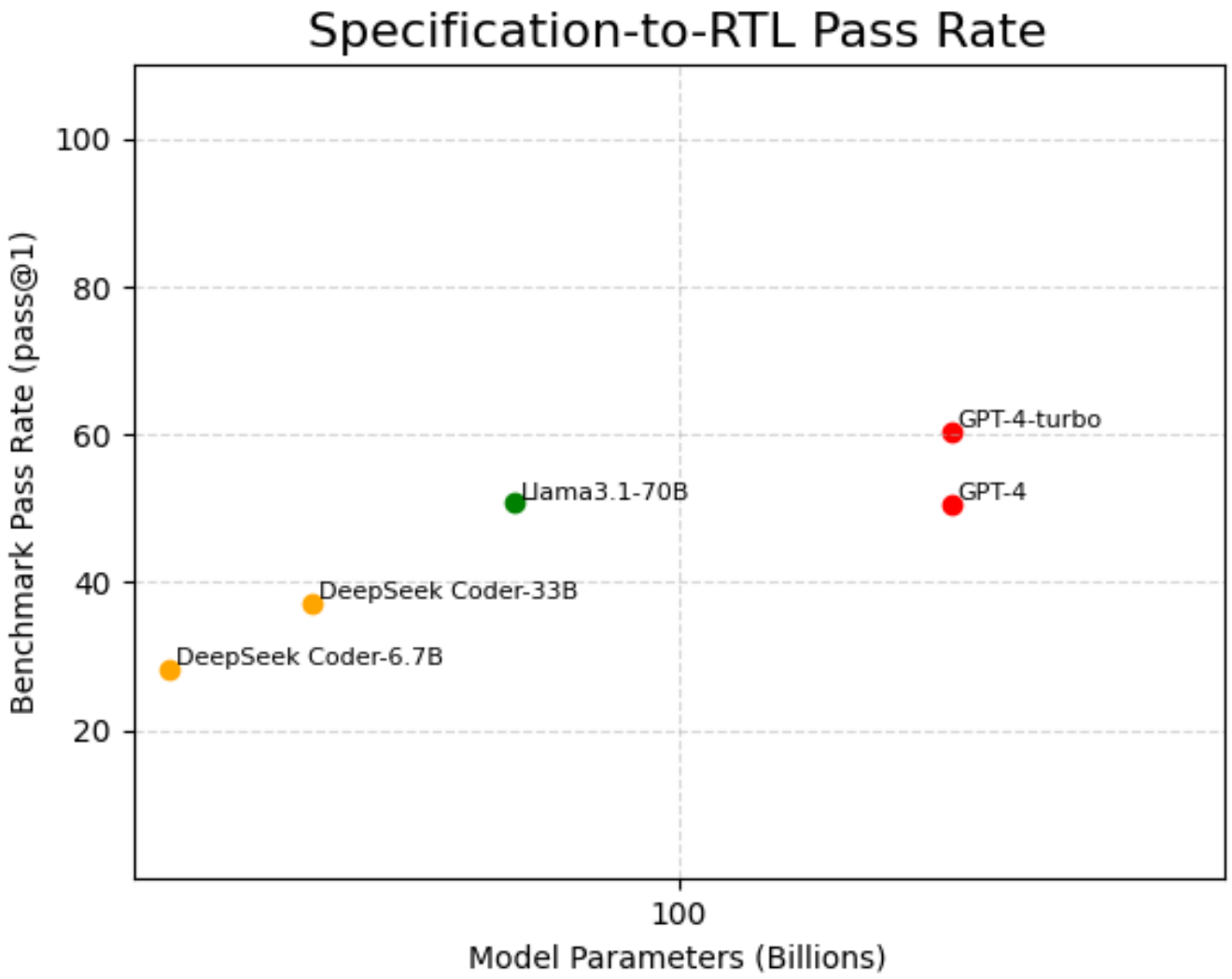
  // Fast counter
  always @(posedge clk)
    if (!counting)
      fcount <= 10'h0;
    else if (fcount == 999)
      fcount <= 10'h0;
    else
      fcount <= fcount + 1'b1;

  assign count = counting ? scount : 'x;

ndmodule
```


NVIDIA

State-of-the-art Models performance



N. Pinckney et al. , Revisiting VerilogEval: Newer LLMs, In-Context Learning, and Specification-to-RTL Tasks

NVIDIA

FVEval

https://github.com/NVlabs/FVEval

NL2SVA-Human (79 problems):

- Testbench + question → SVA
- FIFO, Arbiter, FSM, Counter, RAM

NL2SVA-Machine (300 problems):

- Question → SVA
- Synthetic generated

Design2SVA (192 problems):

- RTL → SVA
- Designs (FSM, Pipeline) generated with generators

Check syntax and function correctness with Jasper

Context:

```
[Design RTL (fsm.sv)]
```

Question:

```
generate a single SVA assertion
for the given design RTL that is
most important to verify.
```

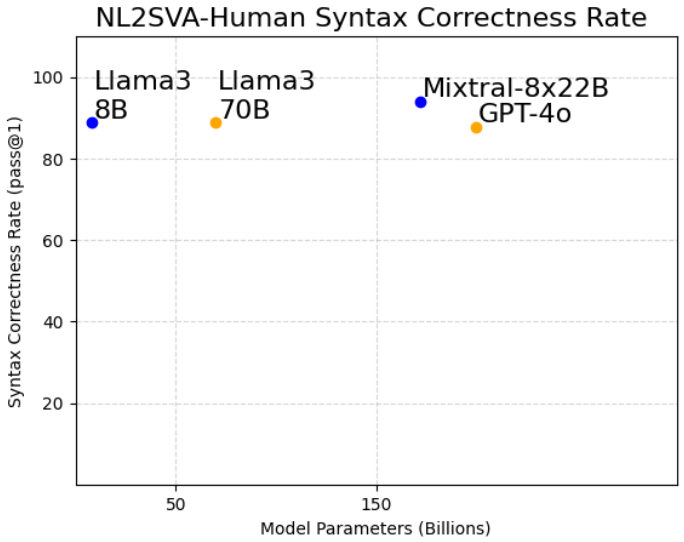
Answer:

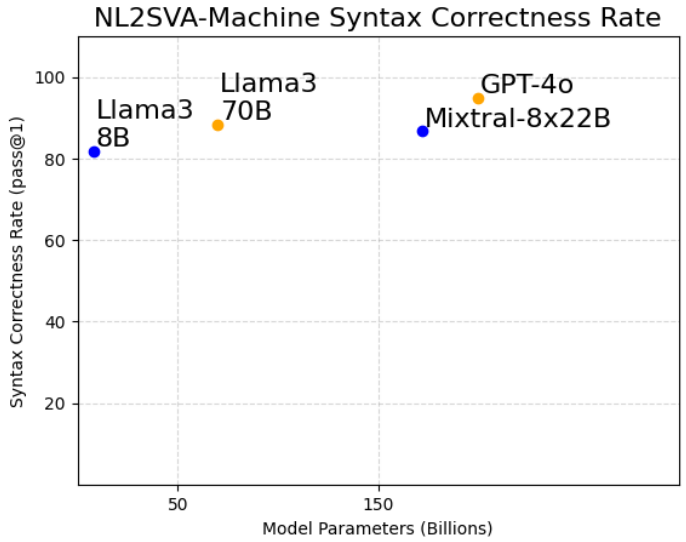
```
logic [3:0] a, b;
assign a = c & d;
assign b = c | d;
assert property (@(posedge clk)
disable iff (tb_reset)(a && b) !=
1'b1;
```

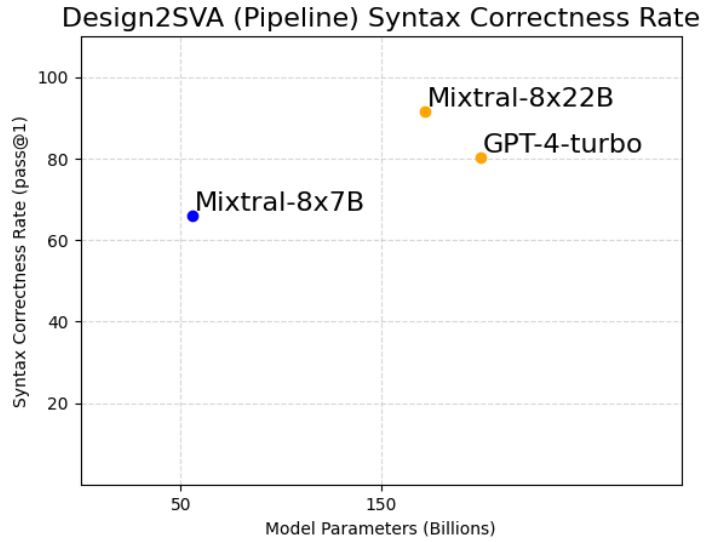
M. Kang et al., FVEval: Understanding Language Model Capabilities in Formal Verification of Digital Hardware

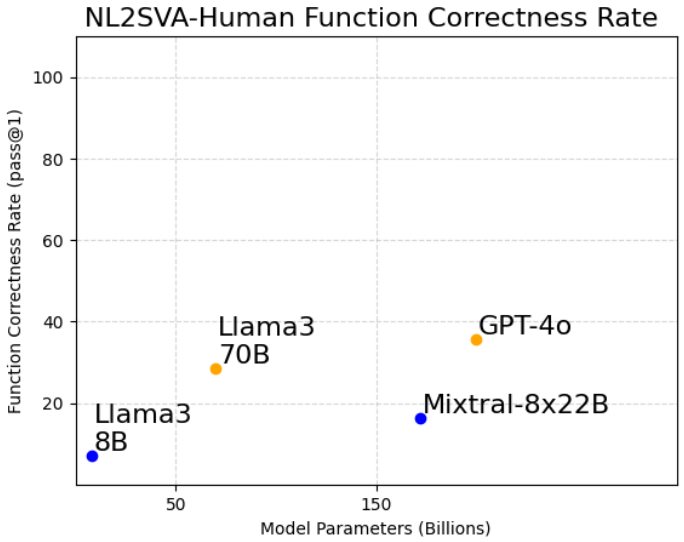
NVIDIA

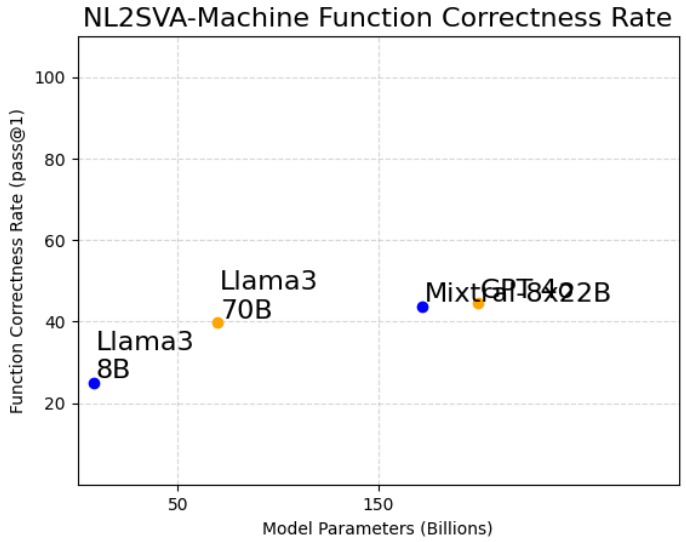
State-of-the-art Models performance

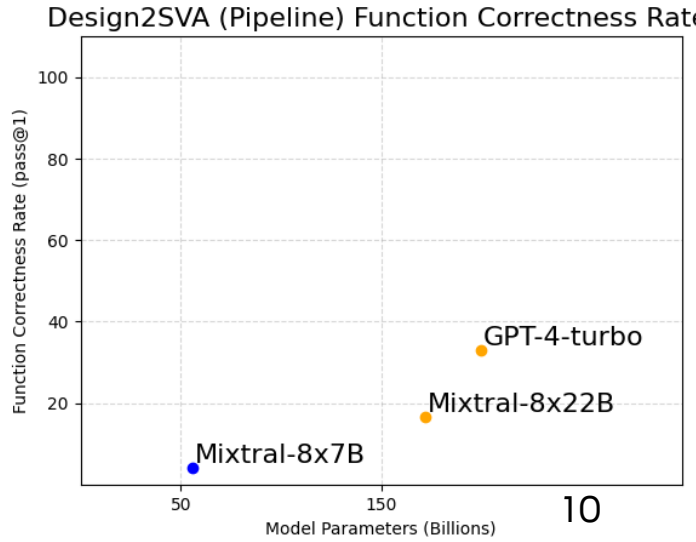












Agenda

LLM for Verilog and SVA Generation

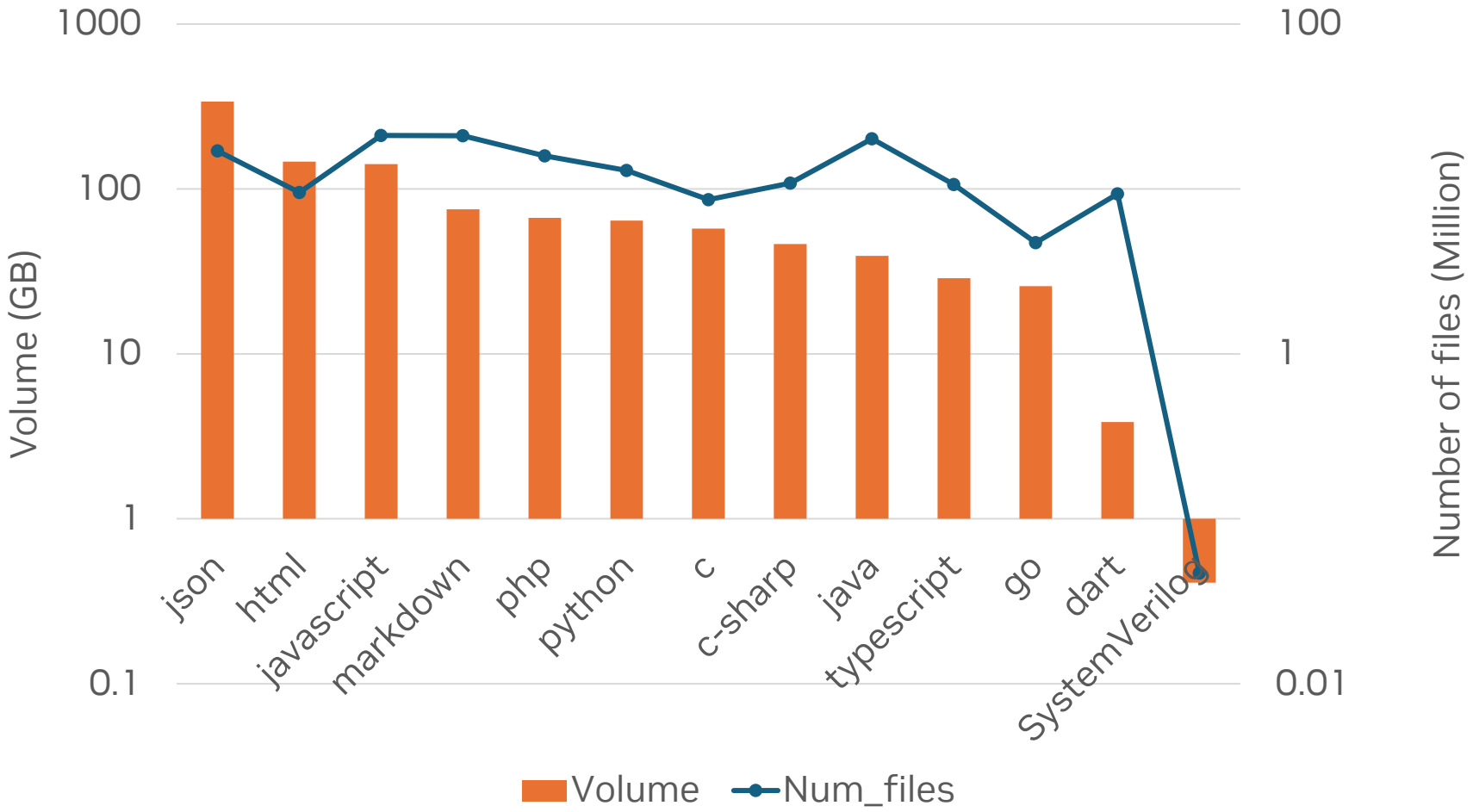
Benchmarks

Enhancing LLM capability

Better model: fine-tune model with synthetic data

Challenges and opportunities

Lack of RTL and SVA training data on github



R. Li et al, StarCoder: may the source be with you!



Welcome to the ICCAD Contest on LLM-Assisted Hardware Code Generation (LLM4HWDesign @ ICCAD 2024)!

Information about the LLM4HWDesign Contest Session @ ICCAD 2024
Time: 10:00 AM - 12:00 PM
Room: Salons A-C
Location: Newark Liberty International Airport Marriott

Large Language Models (LLMs) have demonstrated impressive capabilities in generating high-quality content, sparking interest in their application to hardware design. By assisting in the translation of human instructions into hardware designs (e.g., hardware code), LLMs have the potential to streamline the labor-intensive process of hardware development. Unfortunately, the development of LLMs for hardware design is severely hindered by the scarcity of high-quality, publicly accessible hardware code datasets. Specifically, the lack of adequate datasets prevents effective fine-tuning of LLMs, a critical method for equipping them with hardware domain knowledge and mitigating their limited exposure to hardware-specific data during pretraining. This shortage thus significantly impedes progress in LLM-assisted hardware code generation.

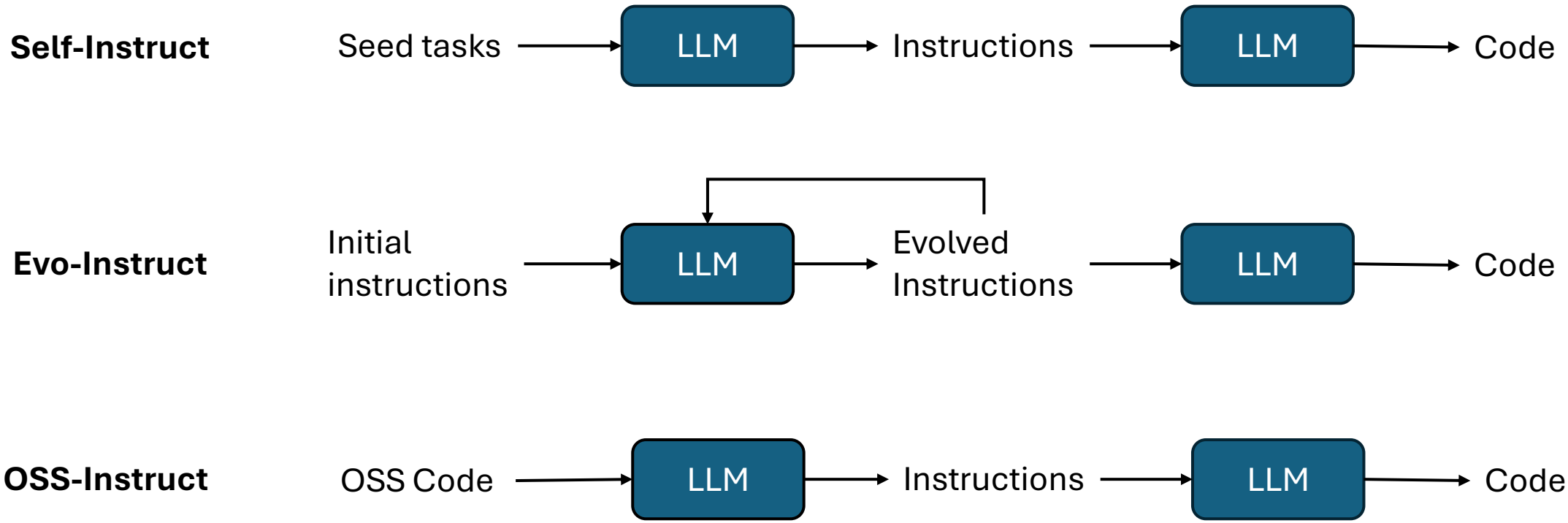
The LLM4HWDesign contest aims to **harness community efforts to develop an open-source, large-scale, and high-quality dataset for hardware code generation**, igniting an ImageNet-like revolution in LLM-based hardware code generation. To achieve this goal, the LLM4HWDesign contest encourages participants to gather data samples and develop innovative data cleaning and labeling techniques that can effectively enhance the scale and quality of datasets for hardware code generation.

The detailed description of the contest problem can be found at Problem page.

If there are any questions that are not addressed in the FAQ page, please feel free to contact us at llm4hwdesign@groups.gatech.edu.

Synthetic Data Generation

How to generate [instruction (specification) →code (Verilog)] pairs



Use NVIDIA nemotron-4-340b-instruct as LLM for SDG

CraftRTL: crafting SDG for VerilogEval

Conventional SDG:

80.1K x [Instruction → code]

Generate correct-by-construction data for non-textual problems such as : Karnaugh map, FSM and waveform:

28.5K x [non-textual problem instruction → code]

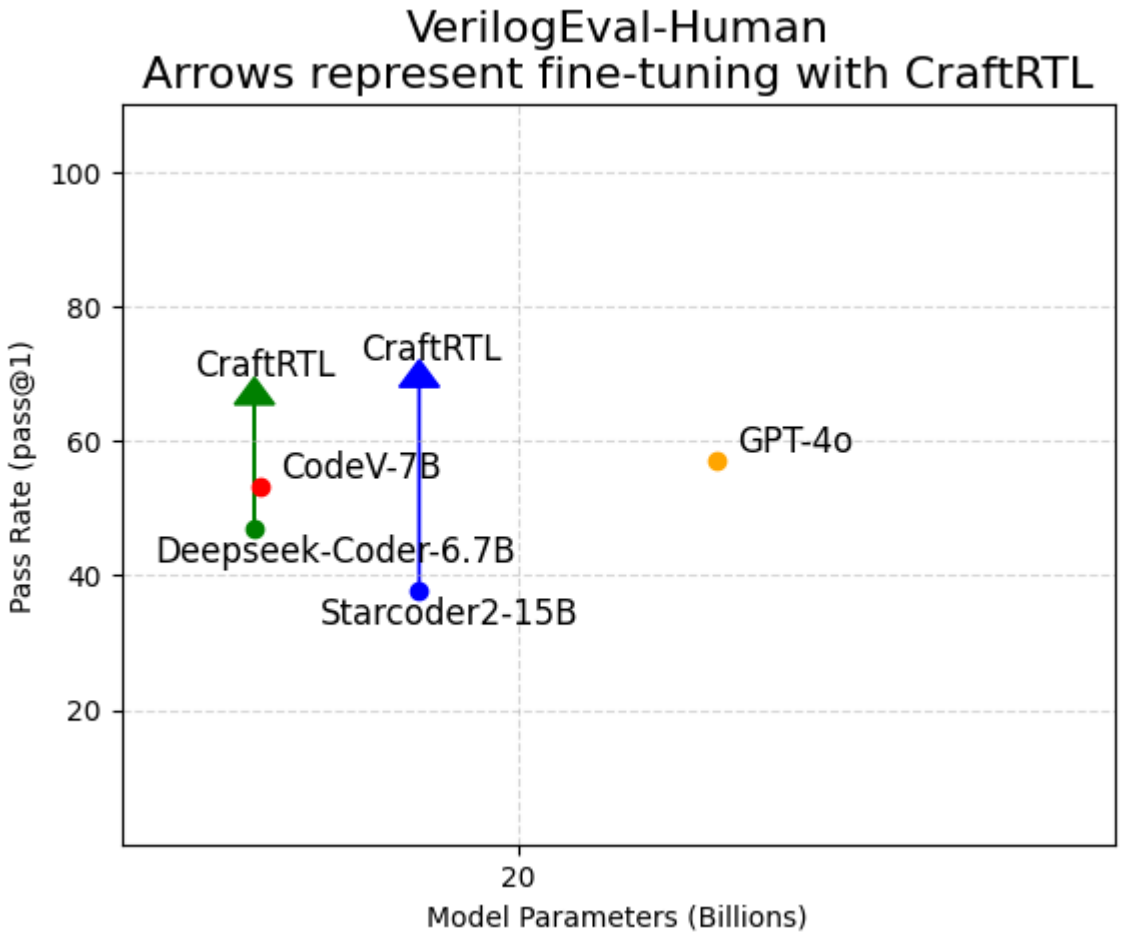
Generate error correction data by automatically analyzing common generation errors and injecting these errors to open-source code snippets:

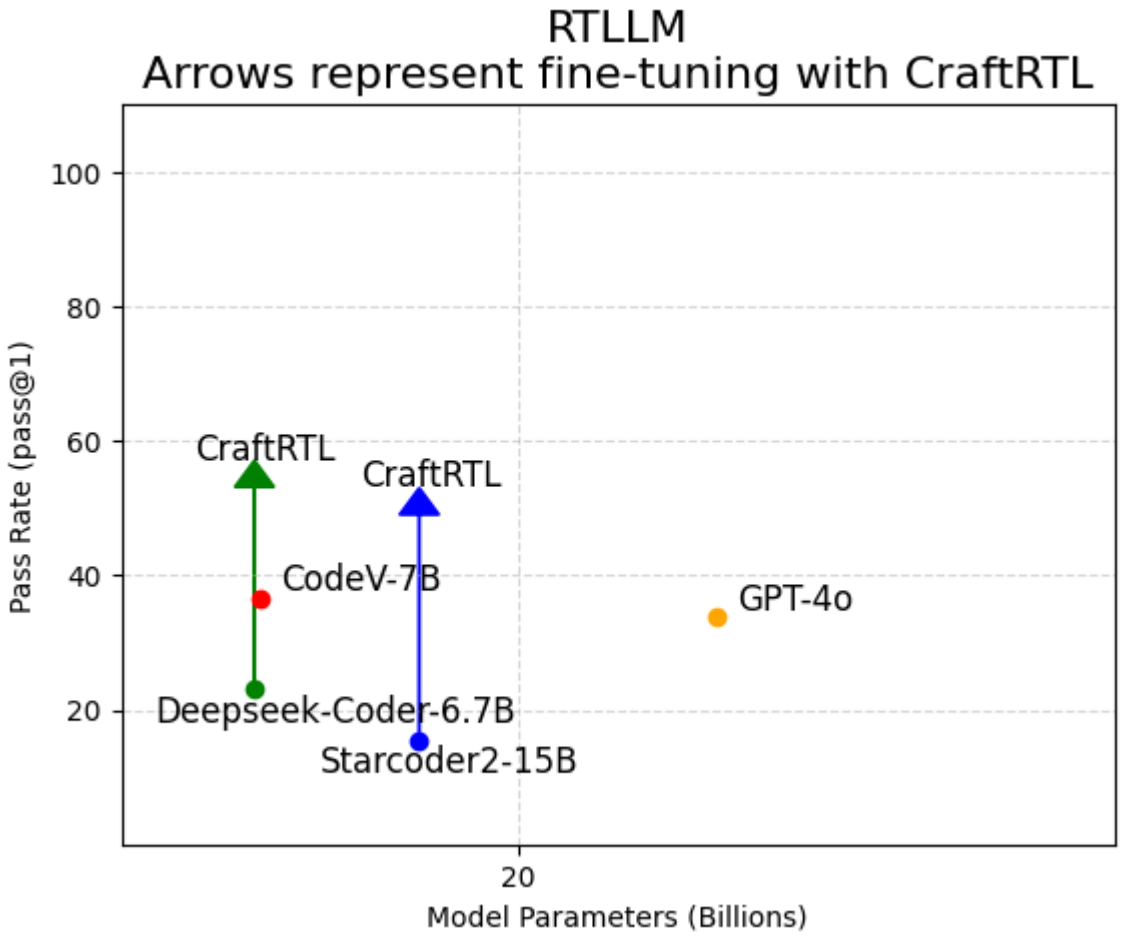
1.4K x [instruction, wrong code → correct code]

M. Liu and Y.-D. Tsai et al., CraftRTL: High-quality Synthetic Data Generation for Verilog Code Models with Correct-by-Construction Non-Textual Representations and Targeted Code Repair

NVIDIA

CraftRTL Models (7B/15B) are the new SOTA for Verilog Benchmarks





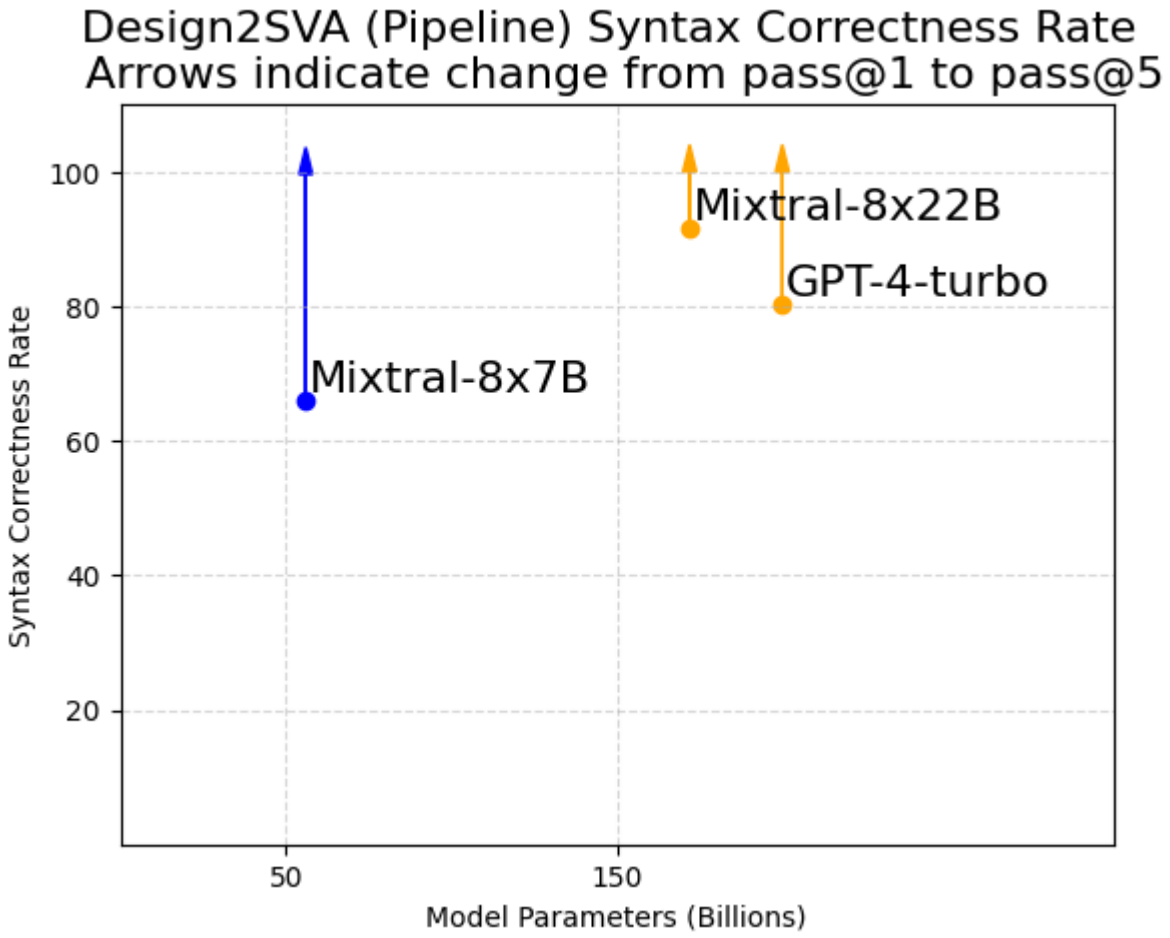


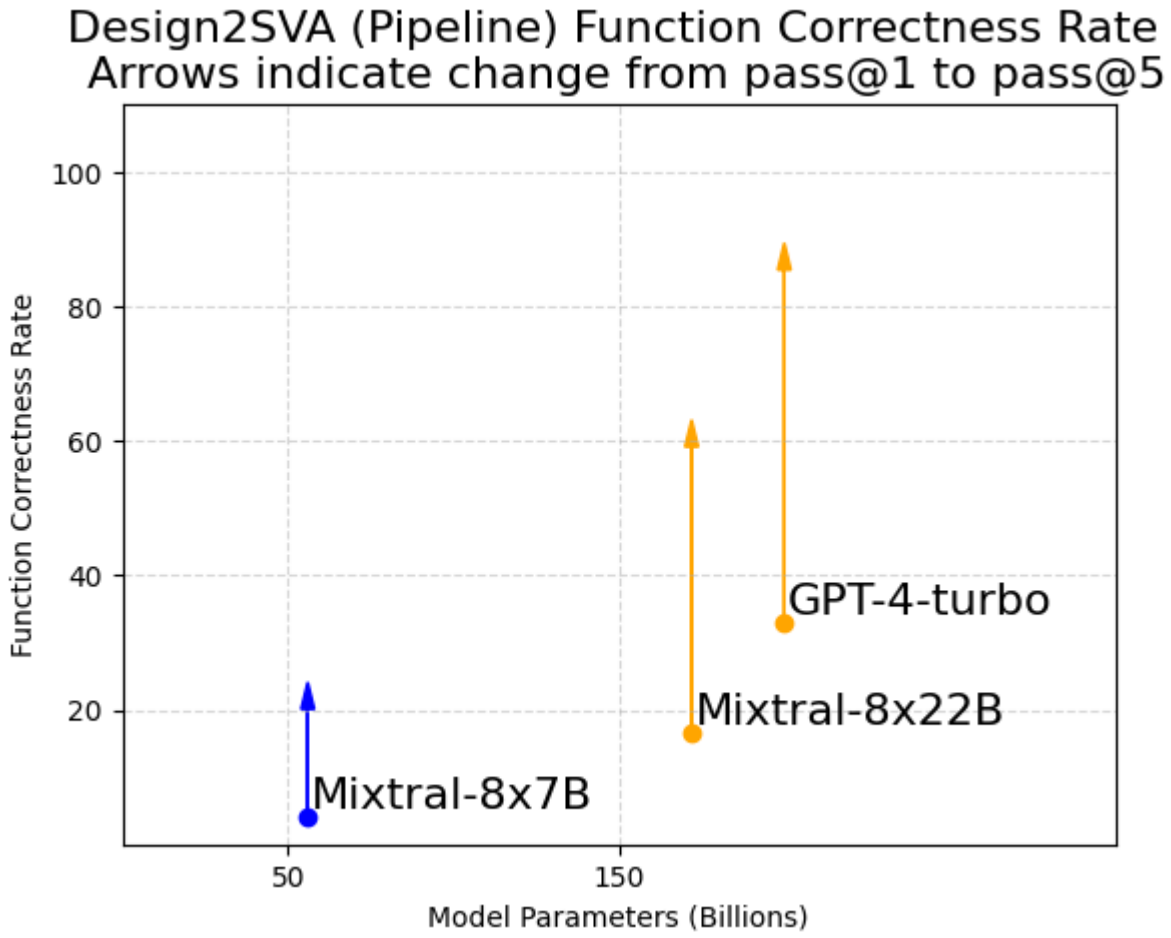
Agenda

LLM for Verilog and SVA Generation

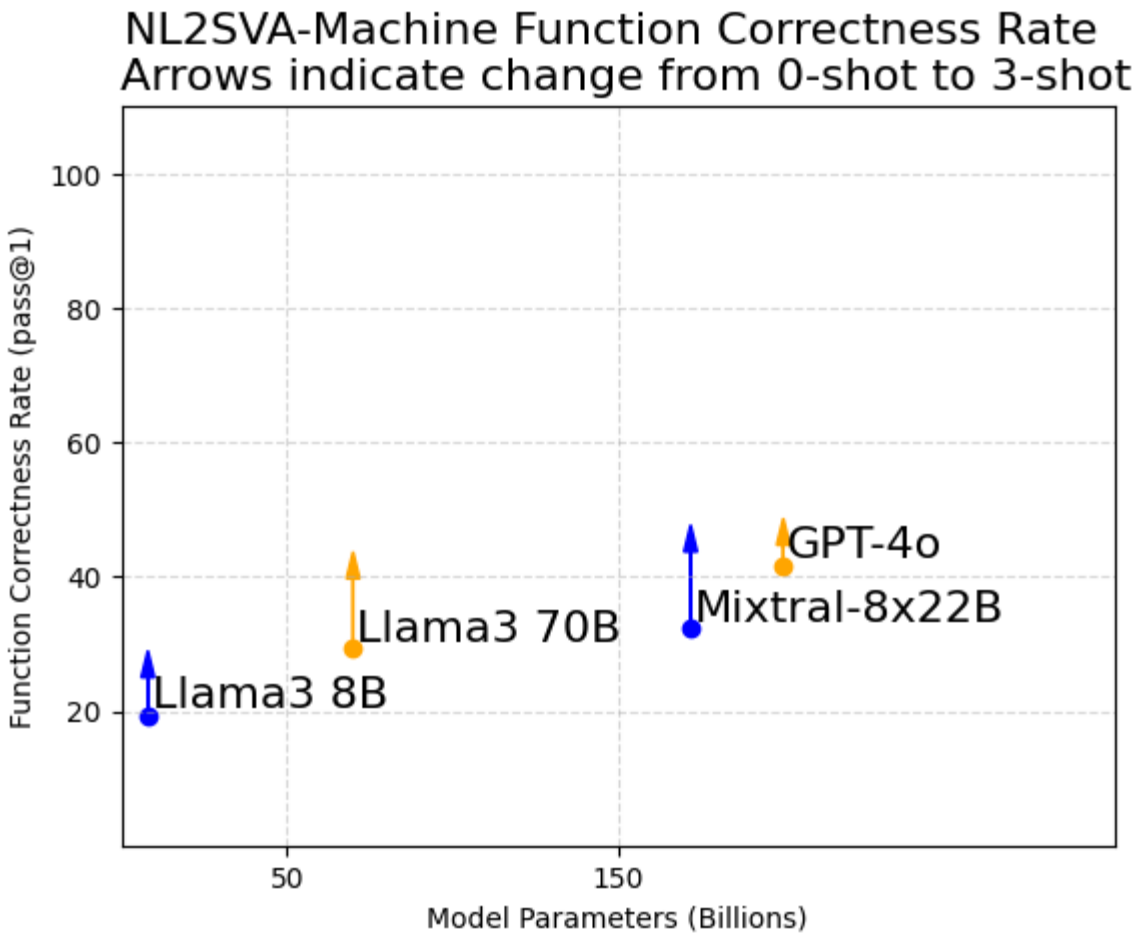
- Benchmarks
- **Enhancing LLM capability**
  - **Better model**: fine-tune model with synthetic data
  - **Better inference:** multi-sampling, few-shot learning, Agents
- Challenges and opportunities

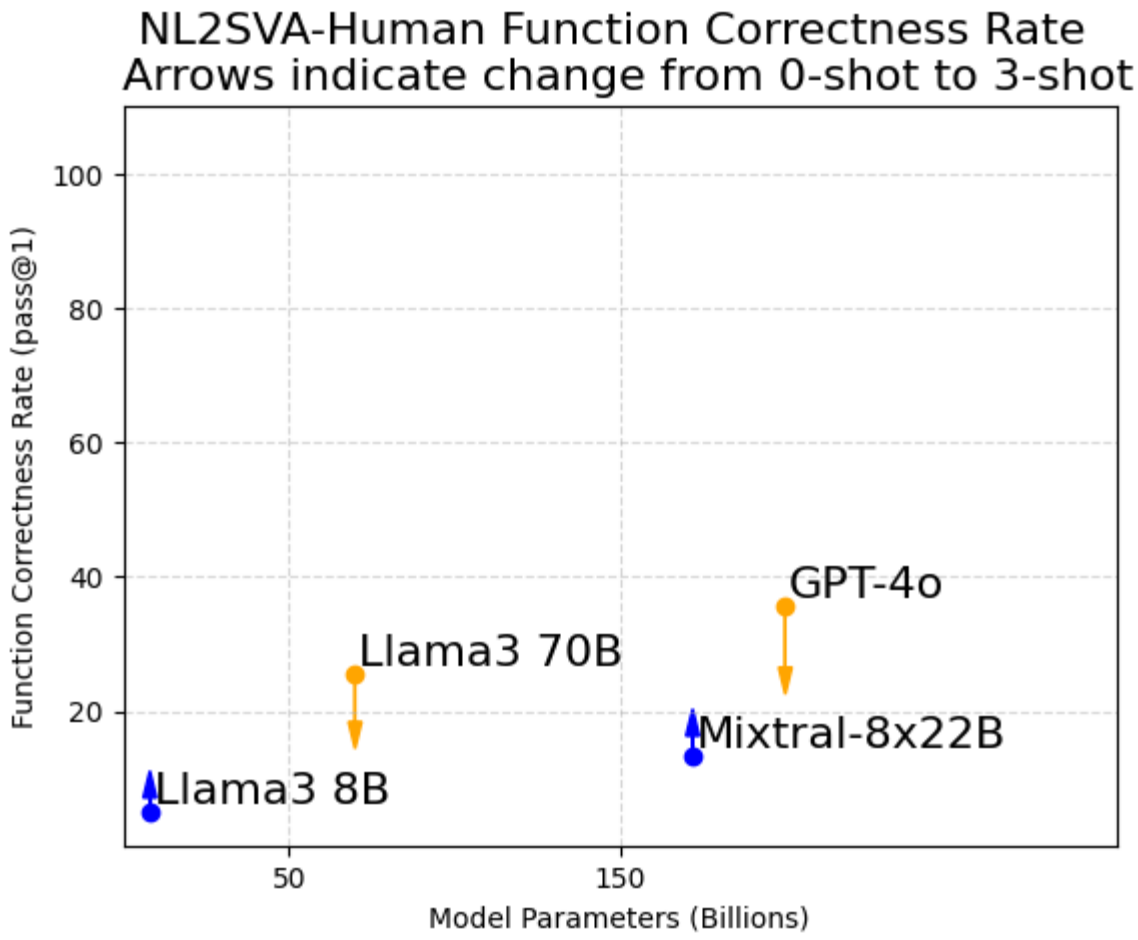
Multi-sampling always help





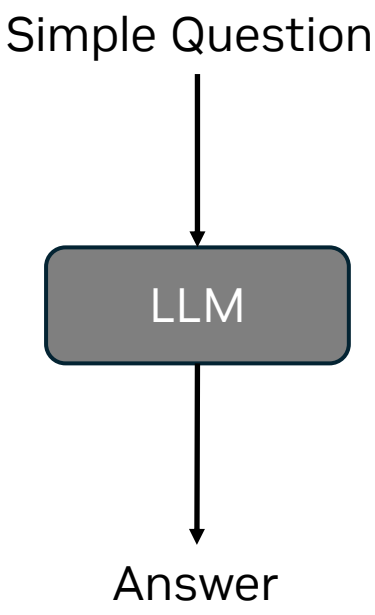
Few-shot learning helps most of the time



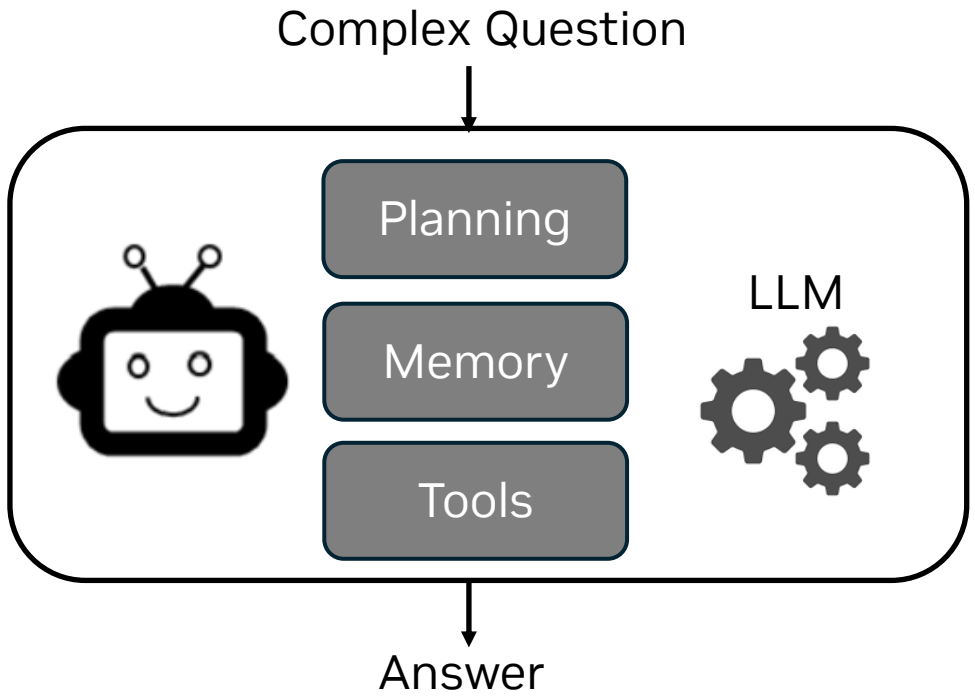


Agentic systems make LLMs much more powerful

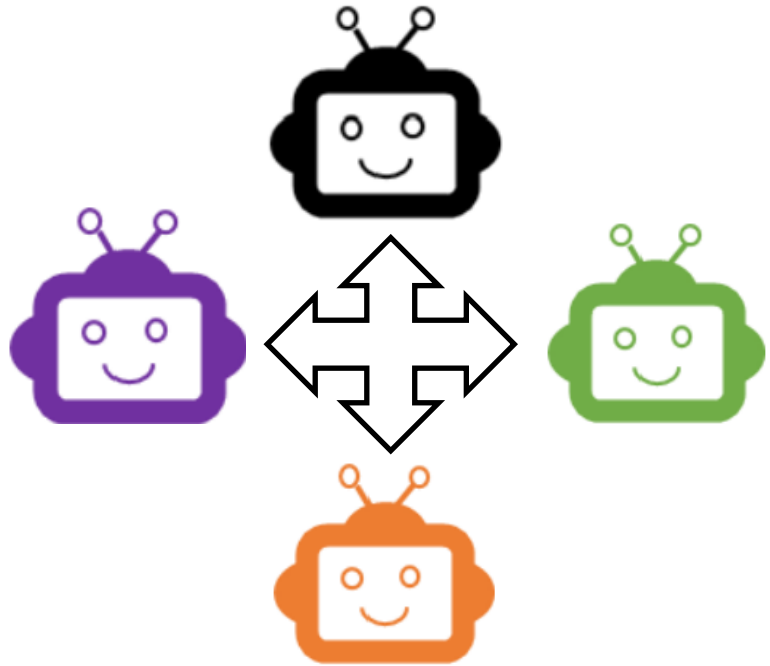
Non-Agentic



Agentic



Single Agent



Multi-Agent

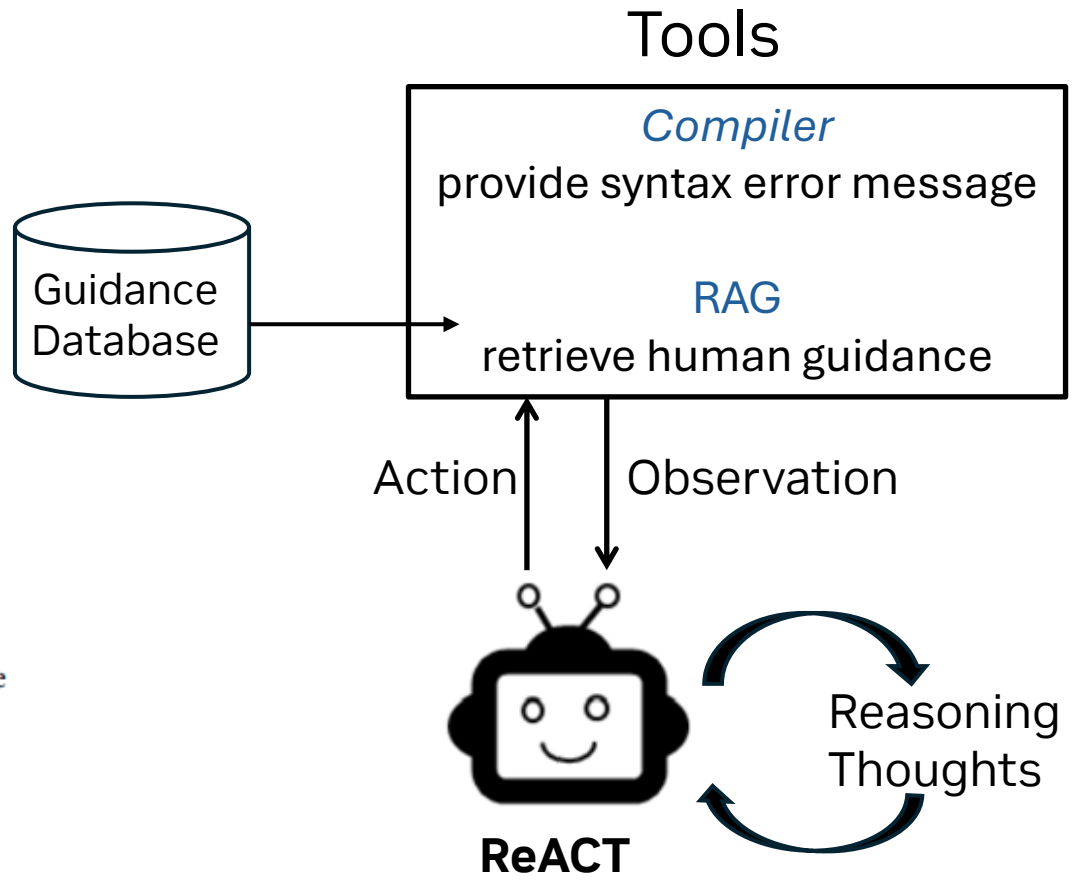
RTLFixer: Fix RTL syntax errors with ReACT Agent

55% of GPT-3.5 Verilog generation errors are Syntax errors

Agent can fix 99% of Syntax Errors

Get feedback/error messages from Verilog compiler

Retrieve human guidance for each syntax error with **RAG**



Compiler Logs:
Object 'clk' is not declared. Verify the object name is correct. If the name is correct, declare the object.
Human Expert Guidance:
Check if 'clk' is an input. If not, and if 'clk' is used within the module, make sure the name is correct. If it's meant to trigger an 'always' block, replace 'posedge clk' with '*'.

Y.-D. Liu et al, RTLFixer: Automatically Fixing RTL Syntax Errors with Large Language Models

FVAgent : Improving NL2SVA with self-learned rules and multi-agents

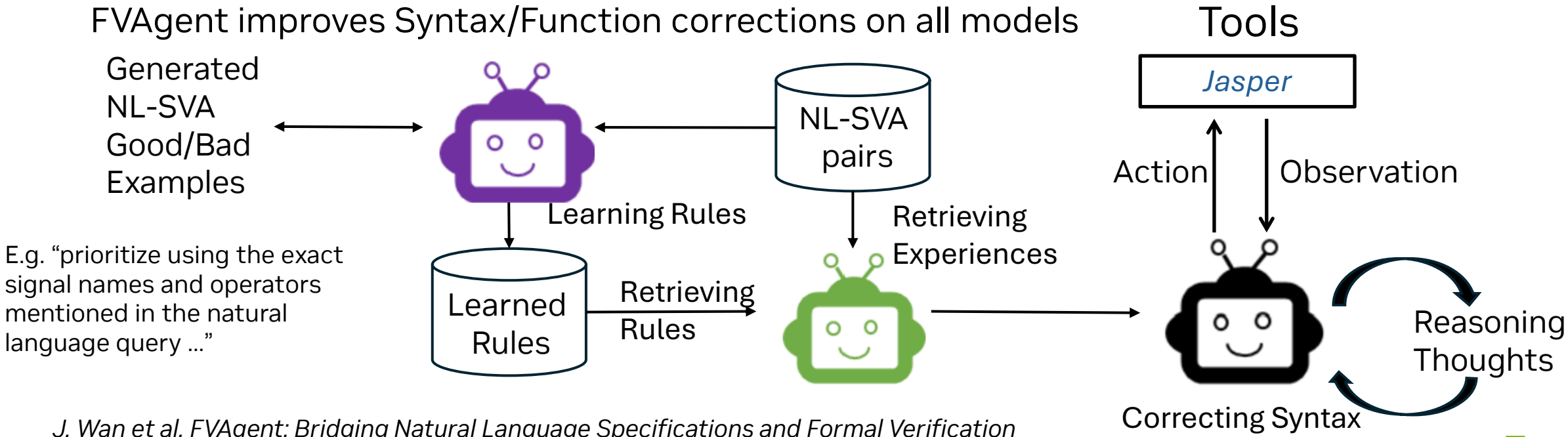
SVA generation is even harder because of less data available

Articulated ‘rules’ in the context improves the accuracy

Self-Learning: Generate rule knowledge base from training data

Multi-Agent task flow : rule generation/retrieval/fix syntax

FVAgent improves Syntax/Function corrections on all models



E.g. “prioritize using the exact signal names and operators mentioned in the natural language query …”

J. Wan et al, FVAgent: Bridging Natural Language Specifications and Formal Verification Assertions with Adaptive Multi-Agent Learning

Learning the Rules

Natural Language: "No grants are made when the arbiter downstream is busy. Use the signals 'tb_gnt', 'busy', and 'last_gnt'."

SVA:

```
asrt: assert property (@(posedge clk) disable iff (tb_reset)
    !(busy && |tb_gnt && tb_gnt != last_gnt));
```

✖ *Functionality error*

BLEU: 0.5859 **Func:** 0.0
Syntax: 1.0 **R. Func:** 1.0

Improvement Knowledge:

- When writing assertions, prioritize using the exact signal names and operators mentioned in the natural language query instead of introducing additional ones.
- When dealing with logical operations, use the simplest and most direct operators (e.g., '&&', '!==') instead of more complex ones (e.g., implication operator '|->') unless necessary.
- Pay attention to the exact wording of the natural language query and avoid overthinking or adding unnecessary signals or conditions.
- When writing assertions, focus on the logical correctness of the statement rather than trying to include all mentioned signals or variables.
- Be mindful of the operator precedence and use parentheses to clarify the order of operations when necessary.

SVA:

```
asrt: assert property (@(posedge clk) disable iff (tb_reset)
    !(busy && |tb_gnt));
```

✔

BLEU:0.6591 **Func:** 1.0
Syntax: 1.0 **R. Func:** 1.0

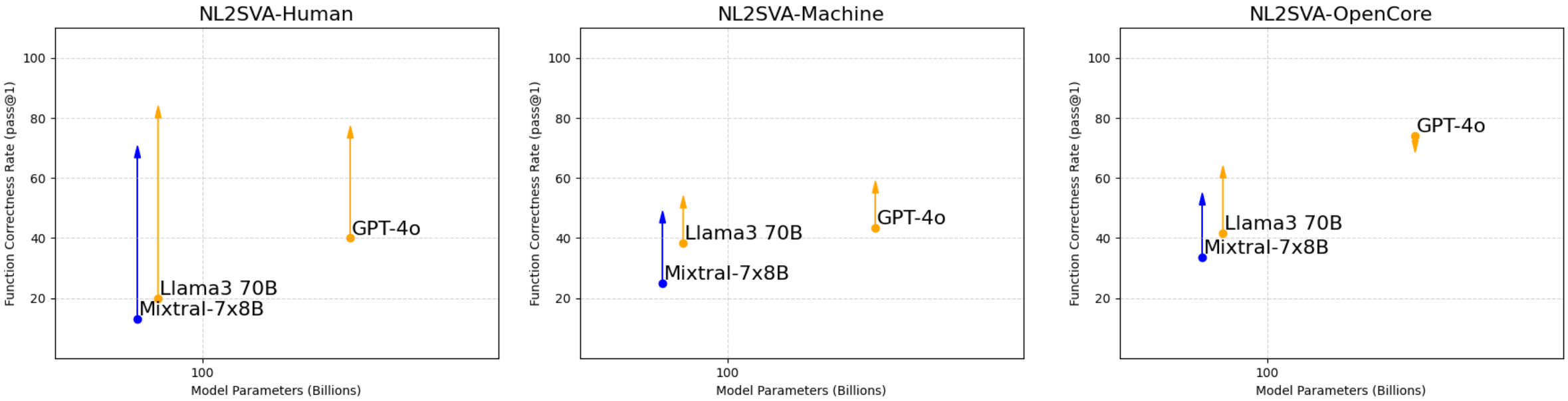
Reference SVA:

```
asrt: assert property (@(posedge clk) disable iff (tb_reset)
    (|tb_gnt && busy) !== 1'b1);
```


NVIDIA

FVAgent significantly improves LLM performance

Arrow indicates improvement of FVAgent vs base models



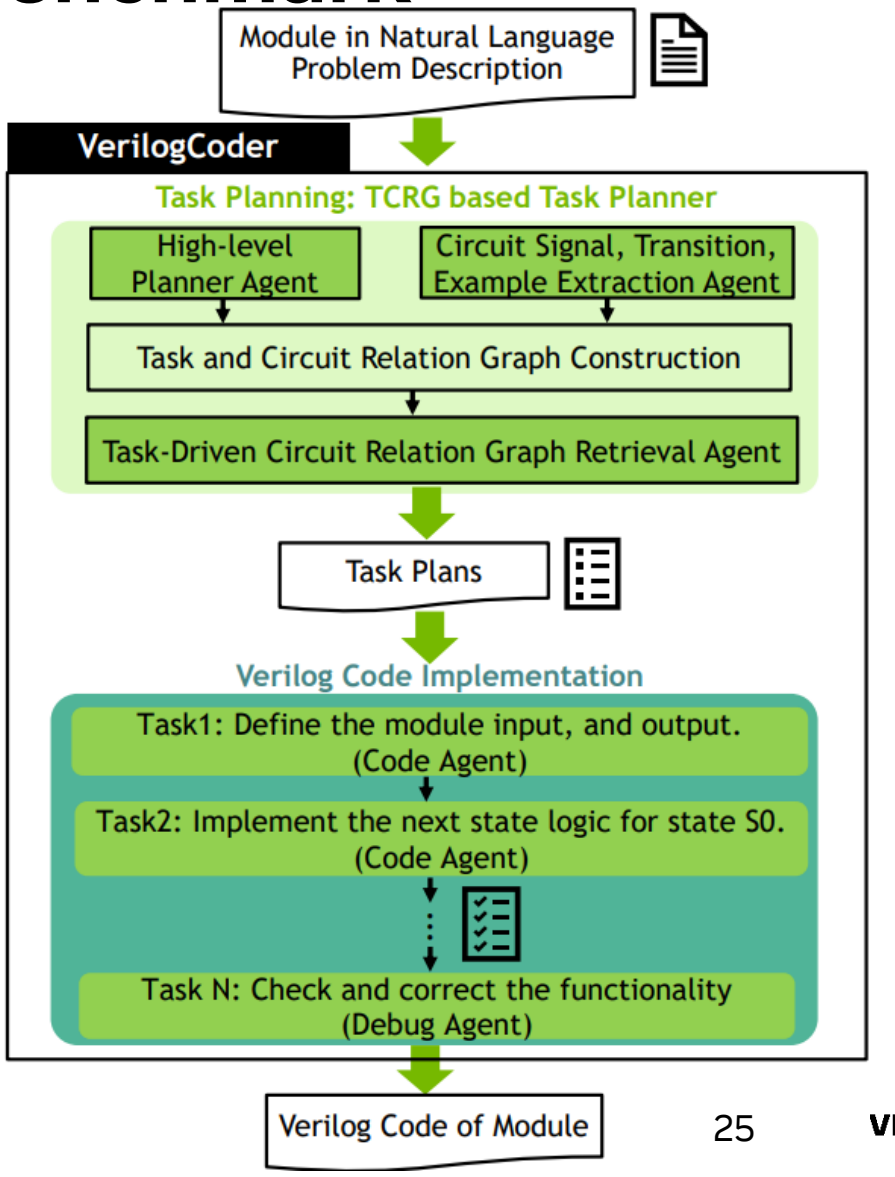
VerilogCoder: Solving VerilogEval benchmark

Two major stages:

The **task planning** stage generates a task plan

The **code implementation** stage writes code for each planned task and debug the code

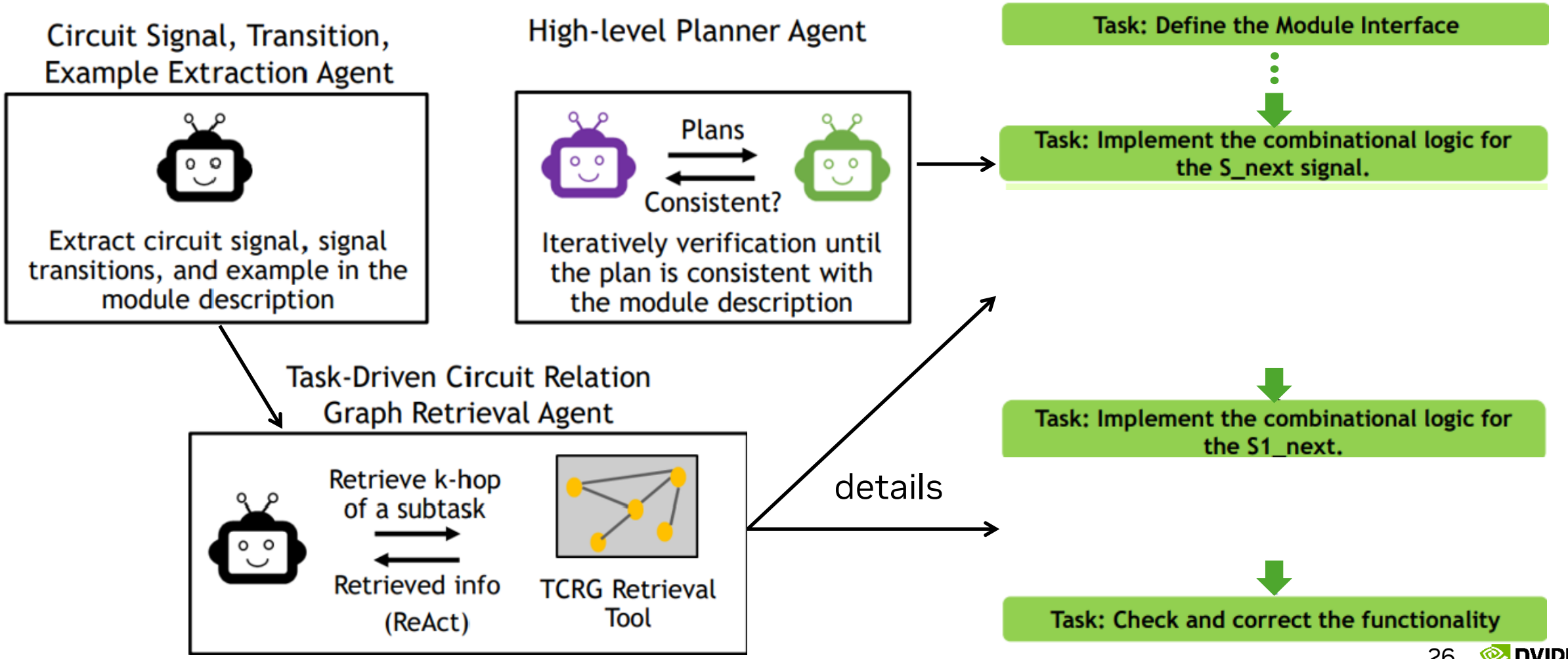
Dynamic task plan generation and execution



C.-T. Ho et al, VerilogCoder: Autonomous Verilog Coding Agents with Graph-based Planning and Abstract Syntax Tree (AST)-based Waveform Tracing Tool

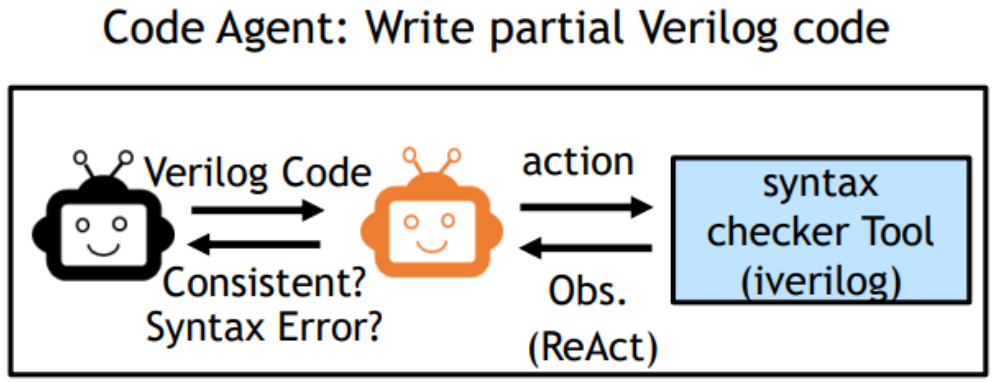
VIDIA

Multi-agent planner builds Knowledge Graph to extract detailed information for task planning

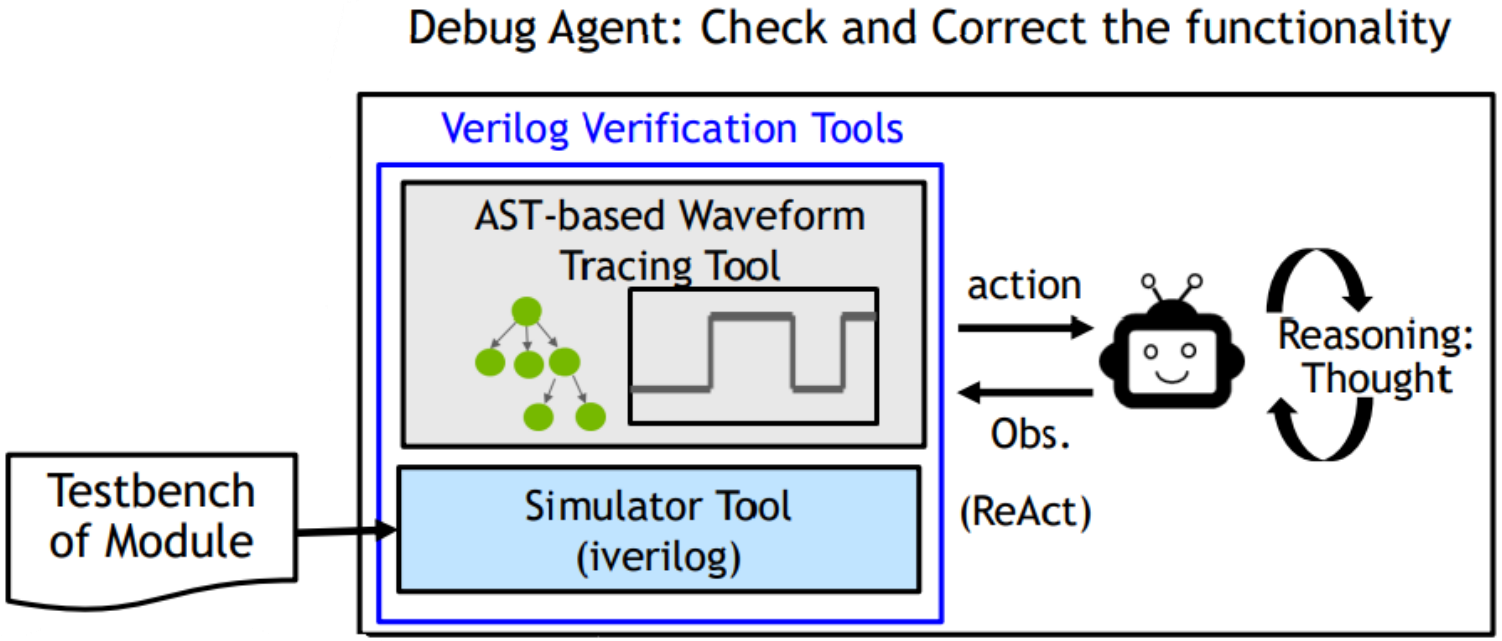


Multi-agent coder leverages syntax checker and waveform tracing tools to write and debug code

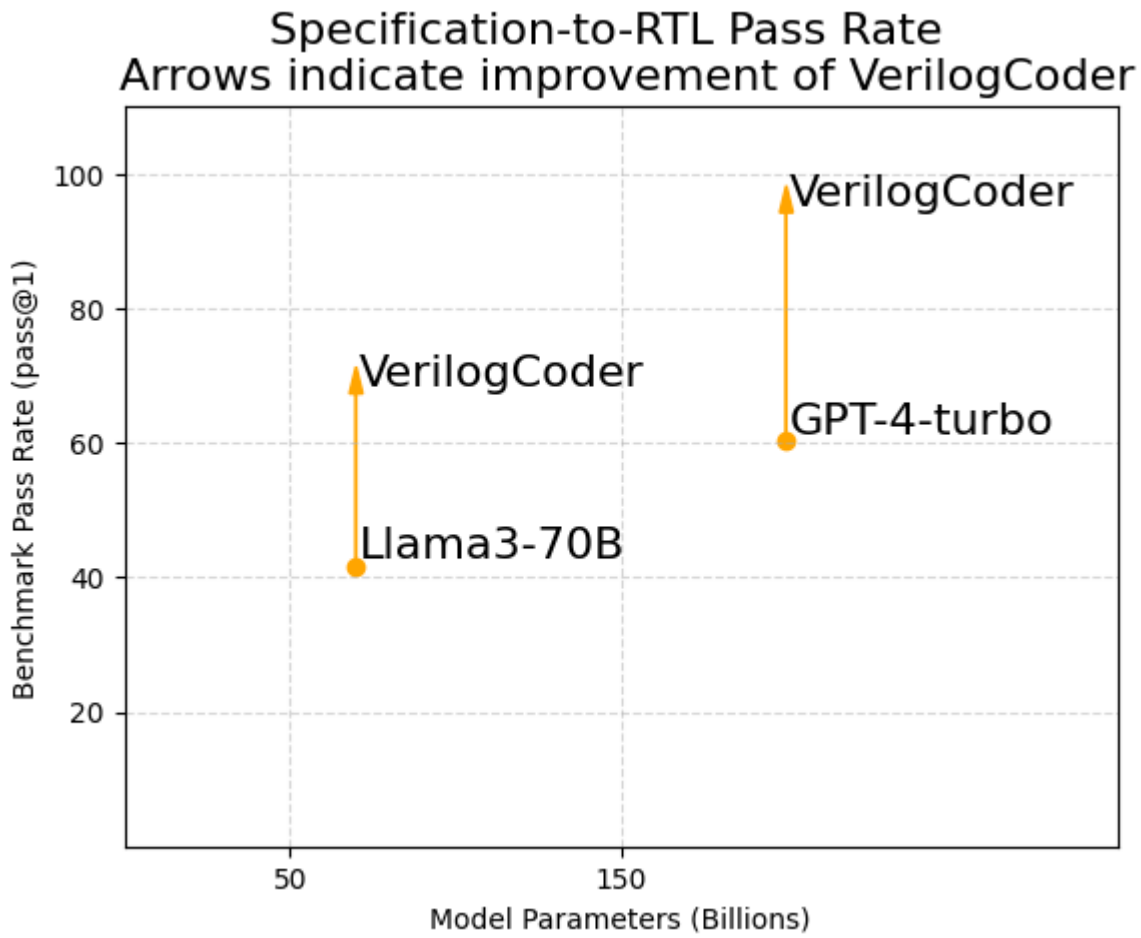
Writing code for each task



Debug final code



VerilogCoder solves most of problems in VerilogEval



C.-T. Ho et al, VerilogCoder: Autonomous Verilog Coding Agents with Graph-based Planning and Abstract Syntax Tree (AST)-based Waveform Tracing Tool

Mate Terminal | Mate Terminal | Mate Terminal

```
(/home/scratch.chiatungh_nvresearch/nvcell_env/llm_env) bash-4.2$ python hardware_agent/examples/verilog_eval_agent/Verilog_eval_task_flow_ver1.py
reading  /home/scratch.chiatungh_nvresearch/hardware-agent-marco/hardware_agent/examples/verilog_testcases/verilog-eval-v2/dataset_dumpall/ Prob139_2013_q2bfsm_prompt.txt
reading  /home/scratch.chiatungh_nvresearch/hardware-agent-marco/hardware_agent/examples/verilog_testcases/verilog-eval-v2/dataset_dumpall/ Prob139_2013_q2bfsm_ref.sv
reading  /home/scratch.chiatungh_nvresearch/hardware-agent-marco/hardware_agent/examples/verilog_testcases/verilog-eval-v2/dataset_dumpall/ Prob139_2013_q2bfsm_test.sv
{'llm_class': <class 'adlrchat.langchain.LLMGatewayChat'>, 'llm_kwargs': {'model_name': 'gpt-4', 'temperature': 0.0, 'top_p': 1.0}}
case manager length =  1
current test is  2013_q2bfsm
current task id is  2013_q2bfsm
current test is  2013_q2bfsm
register tool  sequential_flipflop_latch_identify_tool <function sequential_flipflop_latch_identify_tool at 0x7f65abdac040>  caller:  <autogen.agentchat.assistant_agent.AssistantAgent object at 0x7f65a93ad1b0>  executer:  <autogen.agentc
hat.user_proxy_agent.UserProxyAgent object at 0x7f65a93ad390>
Hardware Agent Initialized 1 proxy, 0 rag proxy, 2 assistants
Hardware Agent Initialized 1 proxy, 0 rag proxy, 1 assistants
user (to chat_manager):

You are a Verilog RTL designer that can break down complicated implementation into subtasks implementation plans.

[Example Begin]

### Problem

I would like you to implement a module named TopModule with the following
interface. All input and output ports are one bit unless otherwise
specified.

 - input  clk
 - input  reset
 - input  in (8 bits)
 - output out (8 bits)

The module should implement an 8-bit registered incrementer. The 8-bit
input is first registered and then incremented by one on the next cycle.

Assume all sequential logic is triggered on the positive edge of the
clock. The reset input is active high synchronous and should reset the
output to zero.

### Solution

module TopModule
(
  input  logic       clk,
  input  logic       reset,
  input  logic [7:0] in,
  output logic [7:0] out
);

  // Sequential logic

  logic [7:0] reg_out;

  always @( posedge clk ) begin
    if ( reset )
      reg_out <= 0;
    else
      reg_out <= in;
  end

  // Combinational logic

  logic [7:0] temp_wire;

  always @(*) begin
    temp_wire = reg_out + 1;
```

Agenda

LLM for Verilog and SVA Generation

Benchmarks

Enhancing LLM capability

Better model: fine-tune model with synthetic data

Better inference: multi-sampling, few-shot learning, Agents

Challenges and opportunities

Specification understanding

Simple spec to real-world specs for both Verilog and SVA generations

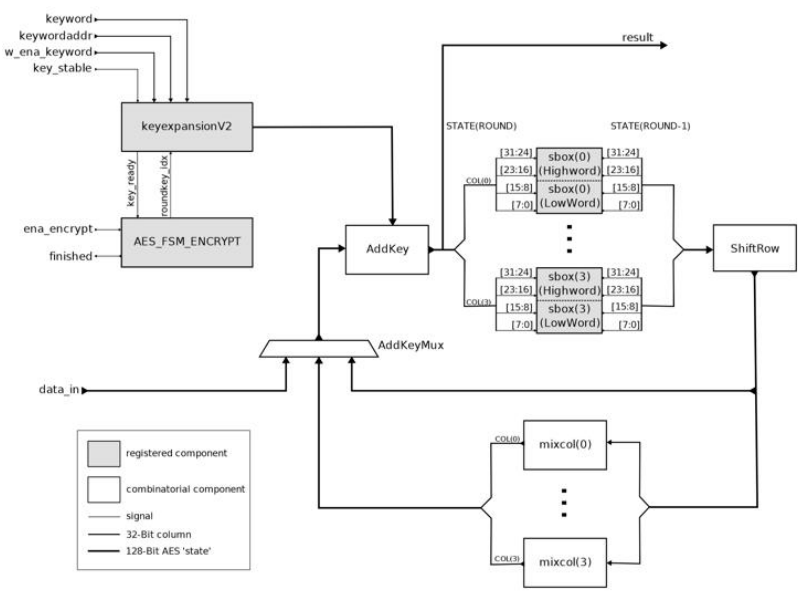
Tens – hundreds of pages, 10Ks - 100Ks tokens

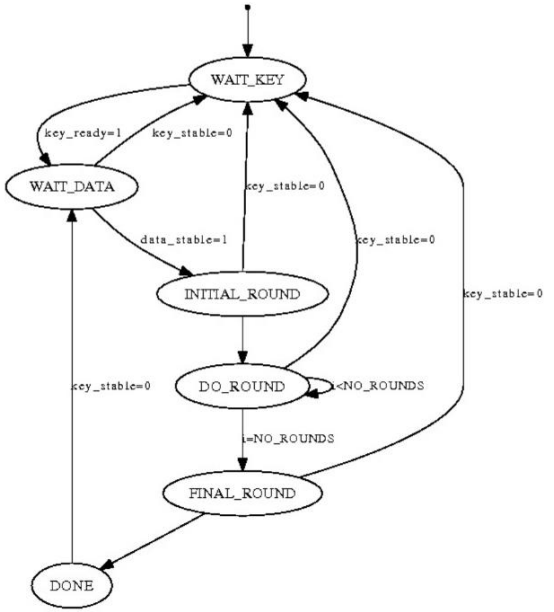
Deal with multi-modal data, generate structured data

Planning is very important: building a dynamic knowledge graph

Refinement approach for implementation of the plan

Signal name	Width	In/Out	Description
clk	1	in	Avalon® bus clock, also used to drive the core.
reset	1	in	*Synchronous* reset signal for Avalon® bus interface. The core itself is designed without need for reset signals.
writedata	32	in	Input data to write to location designated by address. Bit 31 is most significant Bit.
address	5	in	Word offset to the components base address. The memory map of the component for the respective offset is described in 3. Only full 32-Bit words can be addressed no byte addressing is implemented.
write[1]	1	in	If asserted enable write of data at writedata to location designated by address.
read[1]	1	in	If asserted output data at location designated by address to readdata.
readdata	32	out	Data output port for reading data at the location defined by address. Bit 31 is most significant Bit.
waitrequest	1	out	Asserted if writedata was not accepted, this is the case if the keyexpansion is not yet complete and a new is written to the KEY address range without previous deassertion of the KEY_VALID Bit
irq	1	out	If Interrupt behavior is enabled IRQ will be asserted when the operation has terminated. For use of interrupt see 4.1





Towards more comprehensive Benchmarks

Most benchmarks are spec-to-RTL or spec-to-SVA

Many other design problems to tackle, that are suitable for LLM

- Code completion
- Code review
- Code modification
- Code optimization
- QA on Code
- Test generation: stimulus, checker, assertions
- Bug fixing

Need both CoPilot and Agent benchmarks

Opportunities/Challenges for AI in Formal Verification

Assist with test plan and assertion creation: generate test plan from spec and code; generate assertions from spec, code, and test plan

Assist with helper assertion: generates helper assertions to speed up the proof

Assist with abstraction creation: generate abstractions from code

Recommend verification strategy: analyzes which parts of systems are likely to contain errors and focuses the FV efforts on those areas, improving efficiency.

Improve FV tool efficiency: guides heuristics and reduces search state-space in the FV tools to improve efficiency

Assist with debug: automatic debug of FV errors

NVIDIA.

Solving the grand verification challenge: Debug

FV helps reduce the number of bugs, but bugs still exist

Debug is the most time-consuming activity in the verification process

Additional challenges to debug LLM-generated code

LLM-based debug assistant

- Find and analyze relevant design specification, code, log, waveform, and bug report
- Reason about the design logic
- Generate hypothesis
- Leverage models trained with existing documents and code
- Leverage debug tools and FV

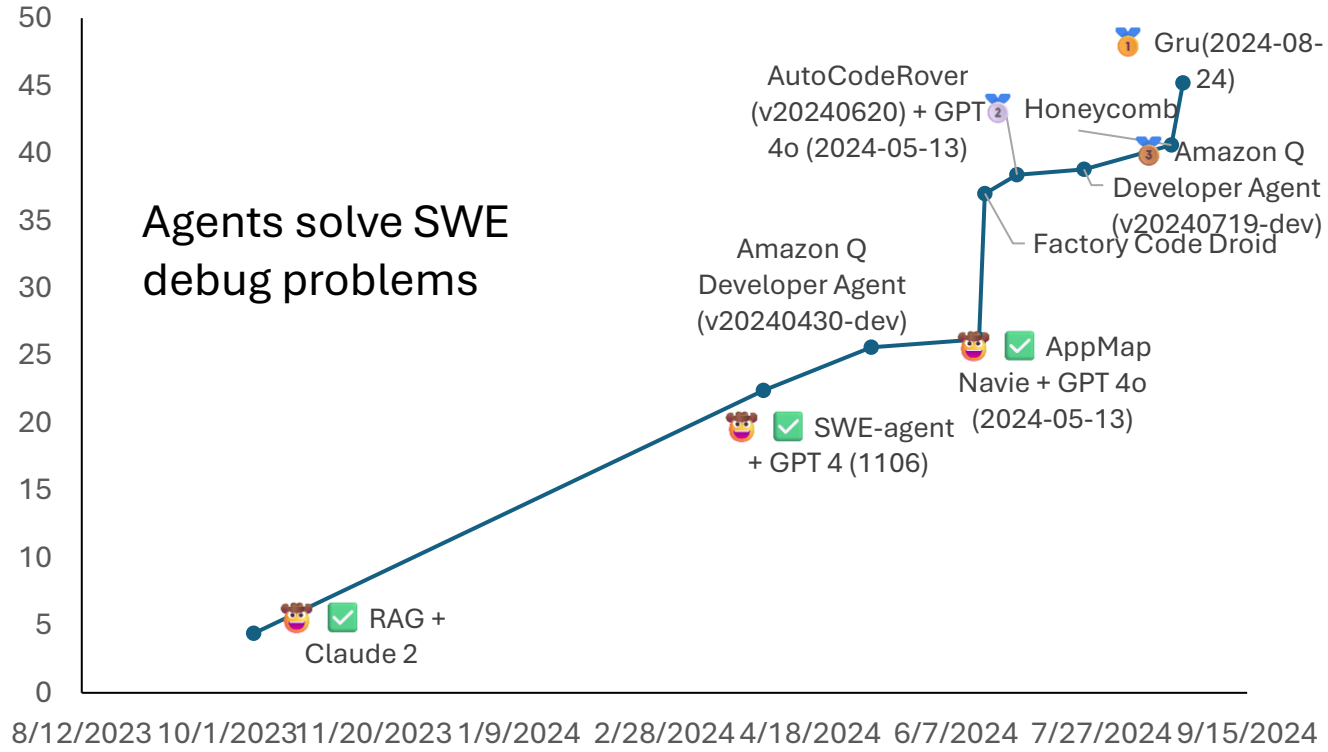
NVIDIA

Agent-based approach for debug

SWE-Bench is a benchmark created from real github debug issues on popular repos like matplotlib

The latest Agent fixes almost half of the problems.

Can this extend to hardware?



https://www.swebench.com/

GPT-4o solves a real and unseen HW issue

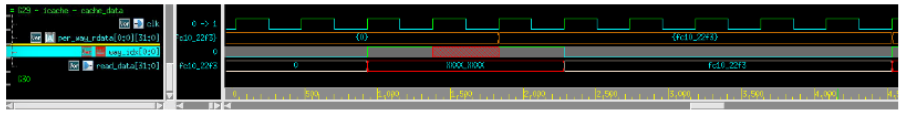
linuxbest commented on May 12 • edited

With the default setting, the NUM_WAYS of icache is 1, so the way_idx should always be 0, but it will change to 1. This behavior is acceptable with Verilator but fails with other simulators.

Verilator:



VCS:



```
597: -bank0 tag-fill: addr=0x80000000, way=1, blk_addr=0, tag_id=0x20000.            <<<< way=1
599: -bank0 mshr-dequeue: addr=0x80000000, id=0 (#0)
599: -bank0 mshr-pop: addr=0x80000000, tag=0x0, req_idx=0 (#0)
599: -bank0 data-fill: addr=0x80000000, way=1, blk_addr=0, data=0x2800bfff002930062900b1203031300000317f
601: -bank0 mshr-table
601: -bank0 tag-hit: addr=0x80000000, way=0, blk_addr=0, tag_id=0x20000 (#0)
603: -bank0 core-rd-rsp: addr=0x80000000, tag=0x0, req_idx=0, data=0xfc1022f3 (#0)
603: -bank0 data-read: addr=0x80000000, way=1, blk_addr=0, data=0xfc1022f3 (#0). <<<<<< way=1, it should
605: I$0 rsp: wid=0, PC=0x80000000, tmask=0001, instr=0xfc1022f3 (#0)
```

linuxbest commented on May 14 Author

The onehot encoder looks wrong to me.

When N==1, the encoder output data should be always 0.

Thanks.

```
diff --git a/hw/rtl/libs/VX_onehot_encoder.sv b/hw/rtl/libs/VX_onehot_encoder.sv
index 92c7d1ea..adaa4183 100644
--- a/hw/rtl/libs/VX_onehot_encoder.sv
+++ b/hw/rtl/libs/VX_onehot_encoder.sv
@@ -29,7 +29,7 @@ module VX_onehot_encoder #(
 );
     if (N == 1) begin

-        assign data_out  = data_in;
+        assign data_out  = 0;
         assign valid_out = data_in;

     end else if (N == 2) begin
```

The issue you're describing seems related to how the `data_out` value is computed for an instruction cache (icache) with `NUM_WAYS = 1`. In this case, `way_idx` should always be 0, but it is changing to 1. This could happen if the logic calculating `data_out` or `valid_out` is not correctly handling the case where `N == 1`.

To fix this issue, we can enforce that when `N == 1`, the `data_out` and `valid_out` behave consistently, ensuring that the output is always zero for `way_idx` regardless of other conditions. Here's the corrected module:

```
if (N == 1) begin

    assign data_out  = 0; // Ensure way_idx is always 0 for N=1
    assign valid_out = data_in;
```

Key Changes:

- For `N == 1`, `data_out` is explicitly set to `0`, ensuring that `way_idx` remains `0`.
- This guarantees the correct behavior when `NUM_WAYS == 1`, avoiding any potential bugs in handling this special case.

This should resolve the issue where `way_idx` unexpectedly changes to 1 when it should always remain 0 in this configuration.

https://github.com/vortexgpgpu/vortex/issues/126

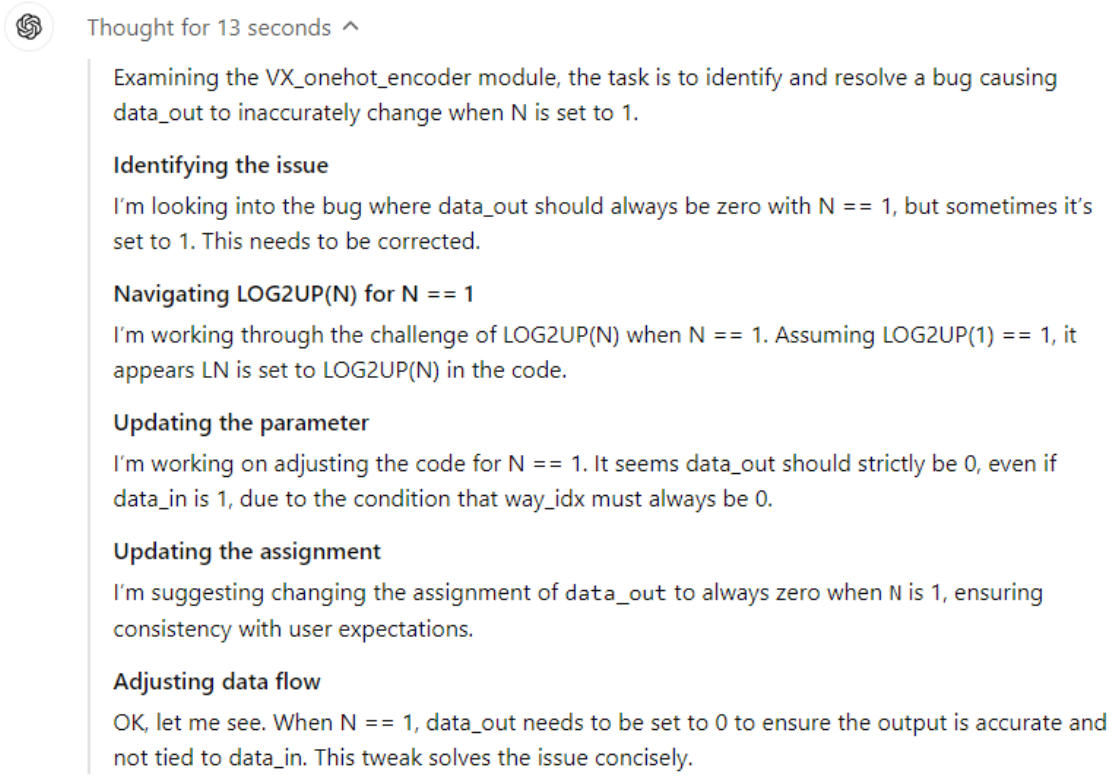
AI learns to reason deeply about chip design

Static plan (predefined CoT, taskflow) work but not flexible

Dynamic plan ('think step by step', VerilogCoder task plan) are more desirable

O1 improves this capability by learning – learning to reason, only good at math/science

Future: Train domain-specific LLM or leverage general-purpose model?



"o1 learns to hone its chain of thought and refine the strategies it uses. It learns to recognize and correct its mistakes. It learns to break down tricky steps into simpler ones. It learns to try a different approach when the current one isn't working. "

OpenAI, Learning to Reason with LLMs

Conclusions

LLMs are capable of generating simple Verilog and SVA today

Fine-tuned model can achieve state-of-the-art results with a fraction of the inference cost with carefully designed synthetic data

Agentic LLM systems can significantly boost code generation accuracy

Stronger reasoning capabilities plus the use of tools will help us overcome the debug challenge

Chip for AI and AI for Chip!

NVIDIA



Huge thanks to Cadence Jasper team’s support for our LLM for FV projects