2025 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION CHINA
accellera SYSTEMS INITIATIVE

SLEG: A LLM-based SVA Evaluation and Generation System

Sichao Yang*, Baoqi Zhang, Jun Liu
GalaxFV, X-EPIC

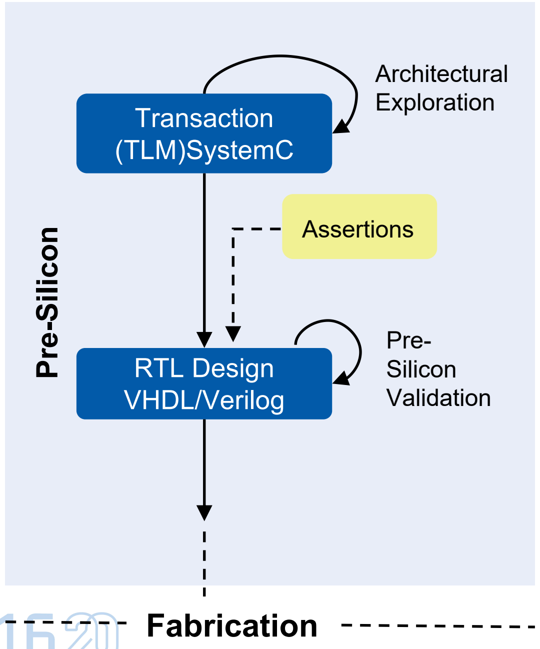
Sihang Shang*, Junjian Peng, Gaihao Liu
SANECHIPS TECHNOLOGY

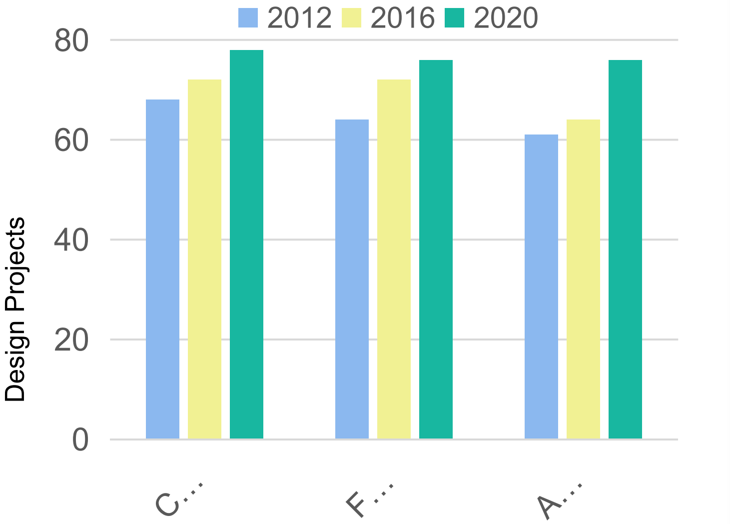
16 Apr 2025 上海 SHANGHAI

Content

Motivation

- SVAs are essential for Design Verification.
- LLM-based tools are under rapid commercialization.





[1] In 2020, more than 75% ASIC designs utilized assertion-based validation

tabnine qodo replit
Ask Codi GitHub Copilot Intellicode
Cody Codiga CodeGeeX A Free Code Generation and Completion Tool
DEEPCODE Amazon Code Whisperer Figstack

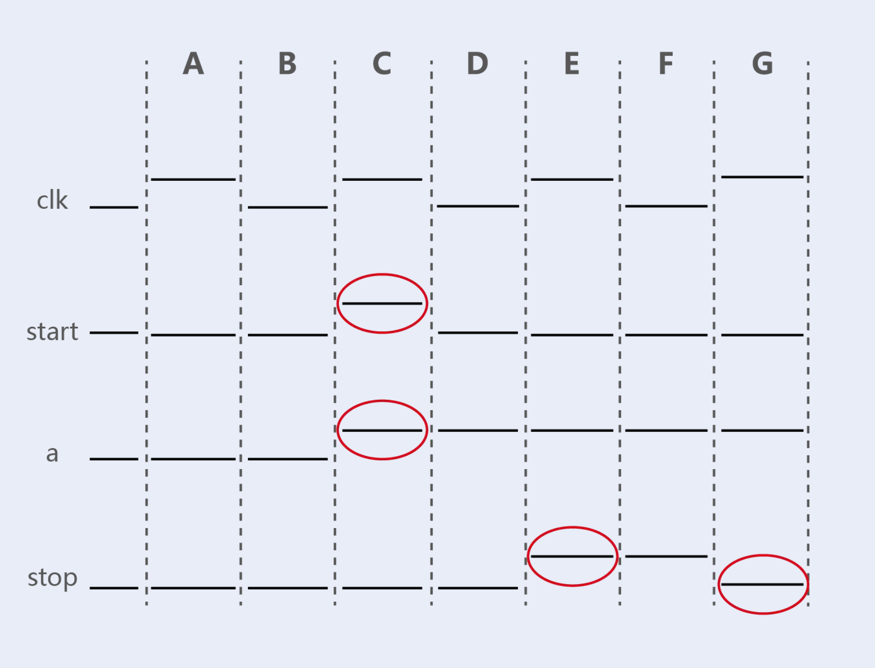
Claude deepseek
OpenAI Grok
Gemini MISTRAL AI_

Challenge

- **SVA can be complicated and difficult to describe.**

Human Description	SVA
When the posedge of signal <clk> is triggered, if signal <start> rises, it implies that after 2 clk cycles, signal <a> should be high for three repetitions non-consecutively and after one time delay signal <stop> should be active, and after another one time delay <stop> should be low.	(@(posedge clk) $rose(start) |-> ##2 (a[=3]) ##1 stop ##1 !stop)
The concatenation of signal <rbF> and <rbE> should have only 1 bit high.	$onehot({rbF, rbE})
When the posedge of signal <clk> is detected, the value of <tmp_ic_miss_state> is equal to 7'b0000001 or 7'b00000010 or 7'b0000100 or 7'b0001000.	@(posedge clk) tmp_ic_miss_state==7'b0000001 |tmp_ic_miss_state==7'b0000010 |tmp_ic_miss_state==7'b0000100 |tmp_ic_miss_state==7'b0001000
……	

@(posedge clk) $rose(start) |-> a ##1 stop ##1 !stop



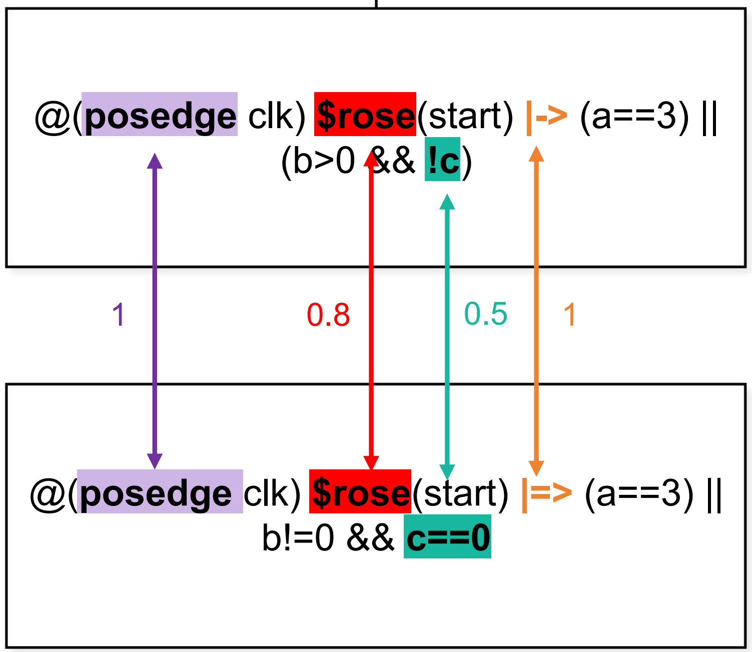
➢ Challenge

- **Traditional methods for SVA evaluation lack preciseness.**

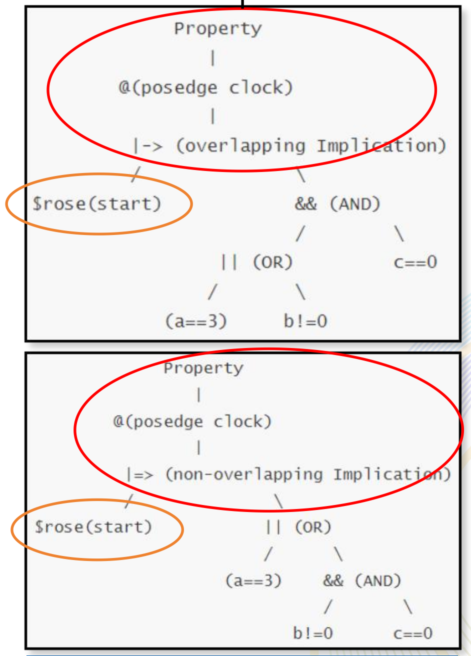
Manual check | **Similarity measures** | **Functional testing**

Golden SVA

Predicted SVA



Weighted N-gram Match [2]



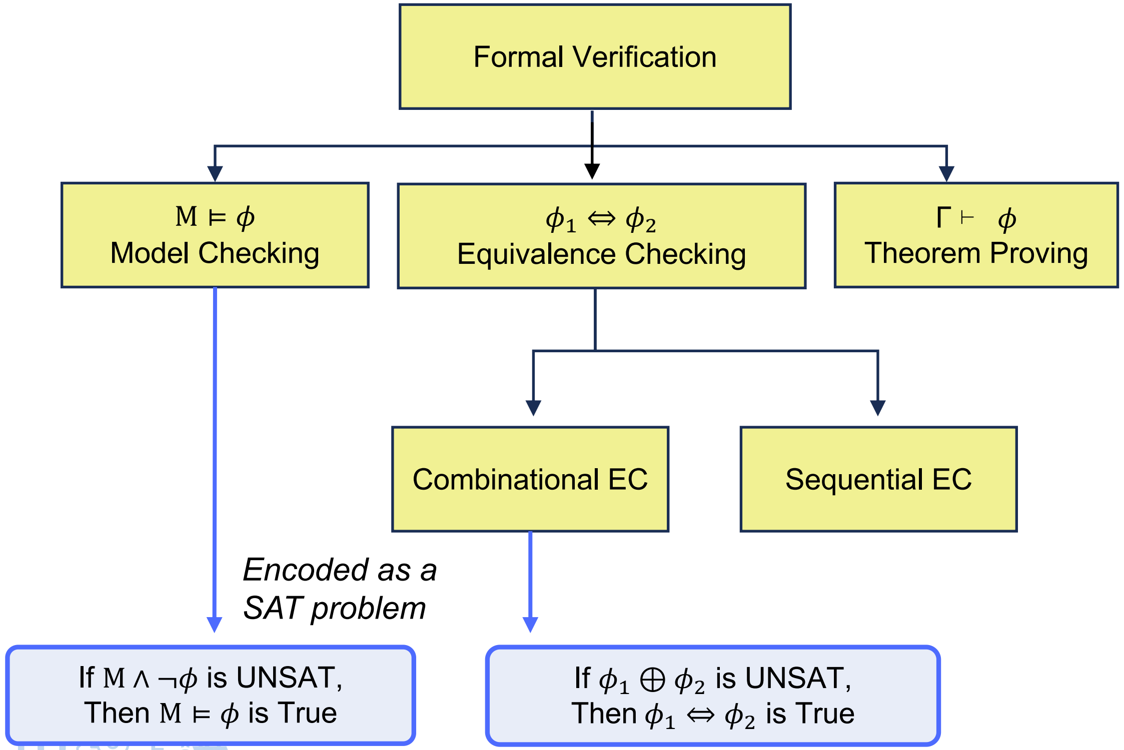
AST Match

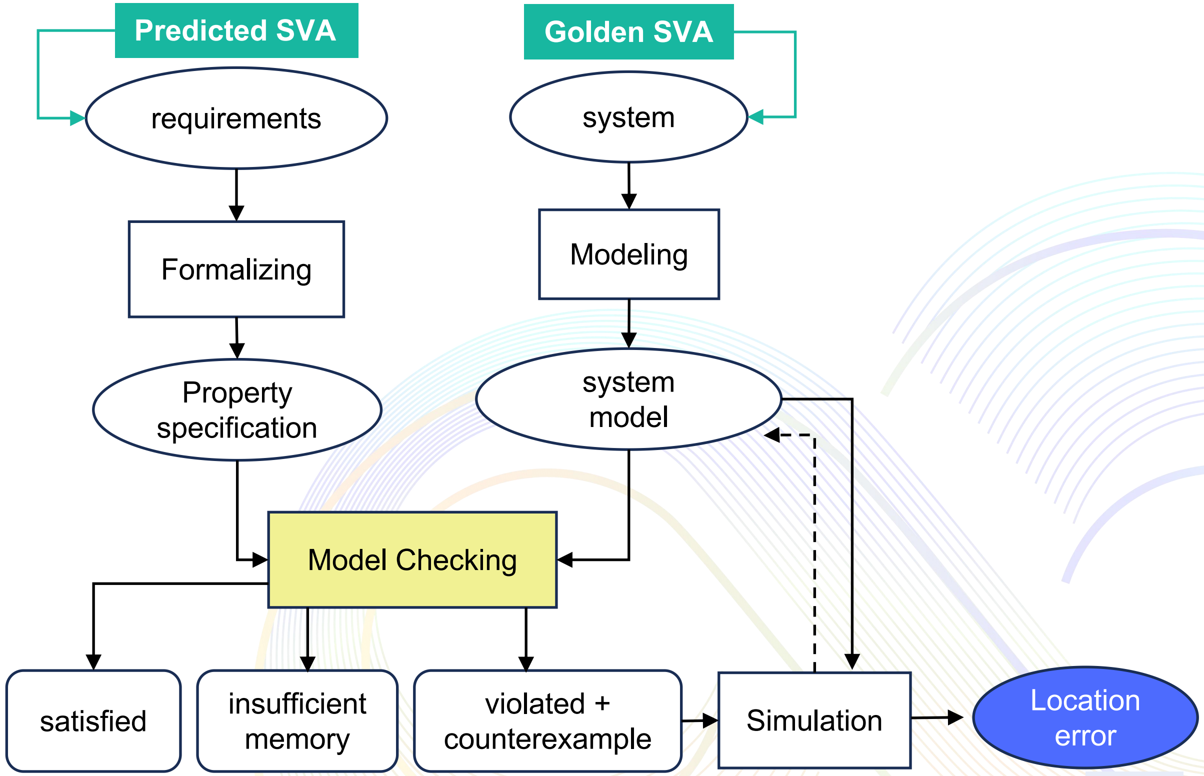
```
SVA:
    @(posedge clk) $rose(start) |-> (a==3) || (b>0 && !c)

Test cases:
case 1:
    start(0)=0, a(0)=0, b(0)=0, c(0)=1
    a(1)=3, b(1)=0, c(1)=1, start(1)=1
    expected result: pass
case 2:
    start(0)=0, a(0)=0, b(0)=0, c(0)=1
    a(1)=2, b(1)=5, c(1)=0, start(1)=1
    expected result: pass
case 3:
    start(0)=0, a(0)=0, b(0)=0, c(0)=1
    a(1)=2, b(1)=0, c(1)=1, start(1)=1
    expected result: fail
case 4:
    start(0)=0, a(0)=0, b(0)=0, c(0)=0
    a(1)=0, b(1)=0, c(1)=0, start(1)=0
    a(2)=0, b(2)=0, c(2)=0, start(2)=1
    expected result: fail
case 5:
    start(0)=0, a(0)=0, b(0)=0, c(0)=0
    a(1)=2, b(1)=1, c(1)=0, start(1)=1
    expected result: pass
...
```

➢ Method – Evaluation by Formal Checking

- **We proposed a Formal-Checking-based evaluation method that is: Sound, Complete and Efficient.** [4]





[7]

Method – Evaluation by Formal Checking

- An app (SVAC) in GalaxFV was developed for SVA checking.

Tcl script

```
set_app_mode SVAC
svac_read_sig signals.txt
svac_map_prop -specFile specPropFile.txt -implFile implPropFile.txt
svac_elaborate
create_clock -clock clk -period 100
check_goals -block
```

Spec&impl prop

```
spec_p1; @(posedge clk) $changed(a) |-> b[->1]; impl_p1; @(posedge clk) $changed(a)1 -> !b[*0:$] ##1 b
spec_p2; @(posedge clk) $rose(a) I=> b until !c; impl_p2; @(posedge clk) $rose(a) |-> ##1 b until_with !c
```

signals.txt

```
clk
srC
Dst
[1:0] a
[1:0] b
[1:0] c
```

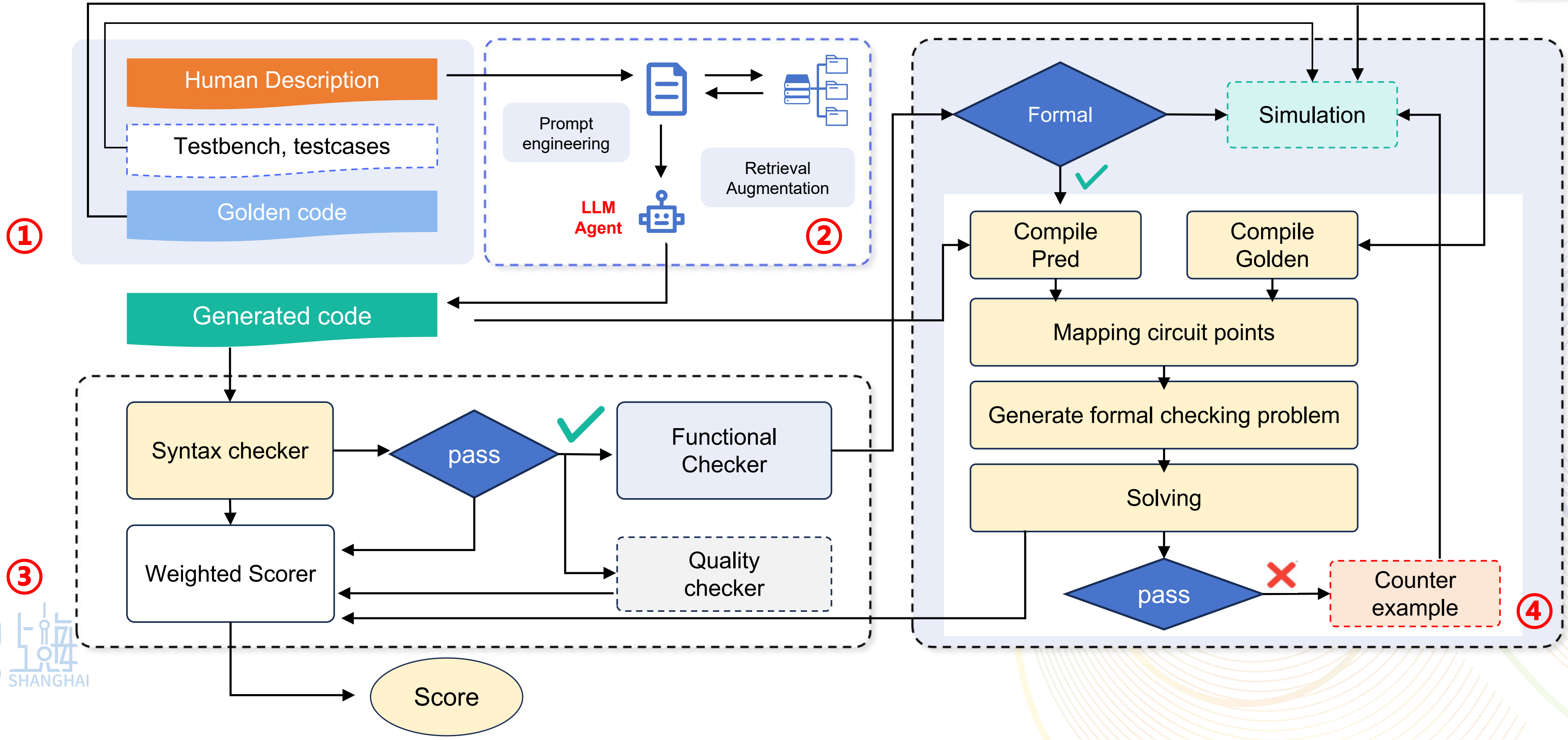
Result

```
---[ SVAC  ]-------------------------------------
Assertion
[0 ]svac_top.spec_p1_comp_impl_p1          proven
[1 ]svac_top.spec_p2_comp_impl_p2          falsified
[2 ]svac_top.spec_p3_comp_impl_p3          proven
galaxfv_shell> _
```

Method – Evaluation by Formal Checking

1. Input data
2. Prediction
3. Evaluation
4. SVAC workflow

- **SLEG includes SVAC and schedule it with other checkers.**



Method – Evaluation by Formal Checking

- **SVAC can evaluate prompting techniques.**

Here are functions and operators in SVA and their descriptions:

FUNCTION	DESCRIPTION
`$rose(expression)`	returns true if the expression changed to 1. Otherwise, it returns false
...	

OPERATOR	DESCRIPTION
`##n`	Delay operators - Fixed n cycles time delay. e.g.: "one time delay" is `##1`
`##[m:n]`	Delay operators - from m to n cycles

...

Here are some rules you should follow when writing SVA code:

- make sure the left bracket number is equal to the right bracket;
- ...

step1: analyze the description's structure, check if it is an immediate or concurrent assertion.

step2: check if it contains sequence, asynchronous reset (`disable iff`) or clocking (`posedge clk`) part ?

...

←Knowledge Appending

↑CoT

↓Few-Shot Learning

Here are a few examples for the task:

[Description]
When the posedge of signal <clk> is detected, signal <sys> should be greater than 10.
[Assertion]
`@(posedge clk) cyc >10`

...

EVAL

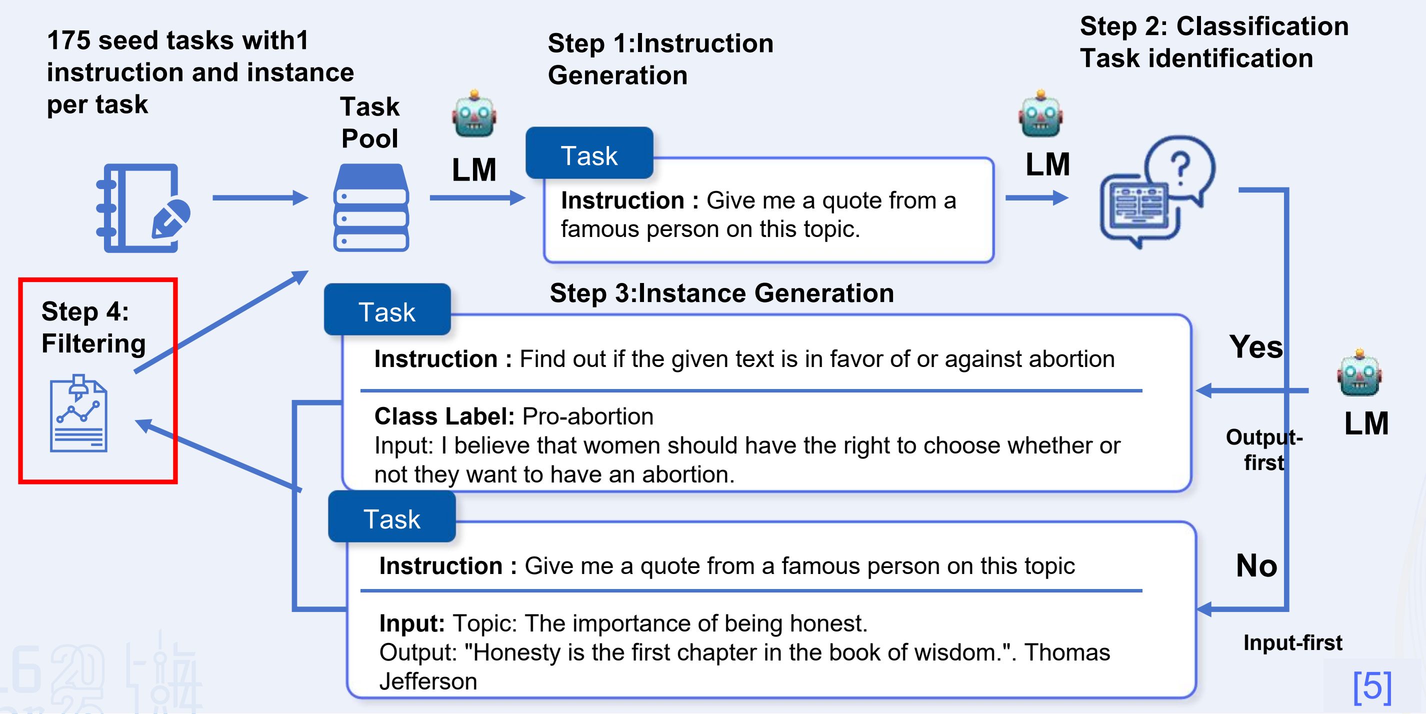
Techniques	Snytax	Functionality	Score
basic prompt	10	6	0.3
syntax_guarding	17	6	0.3
knowledge_appending	17	6	0.3
chain_of_thought	16	4	0.2
fewshot_learning	20	17	0.85
gpt4	20	20	1

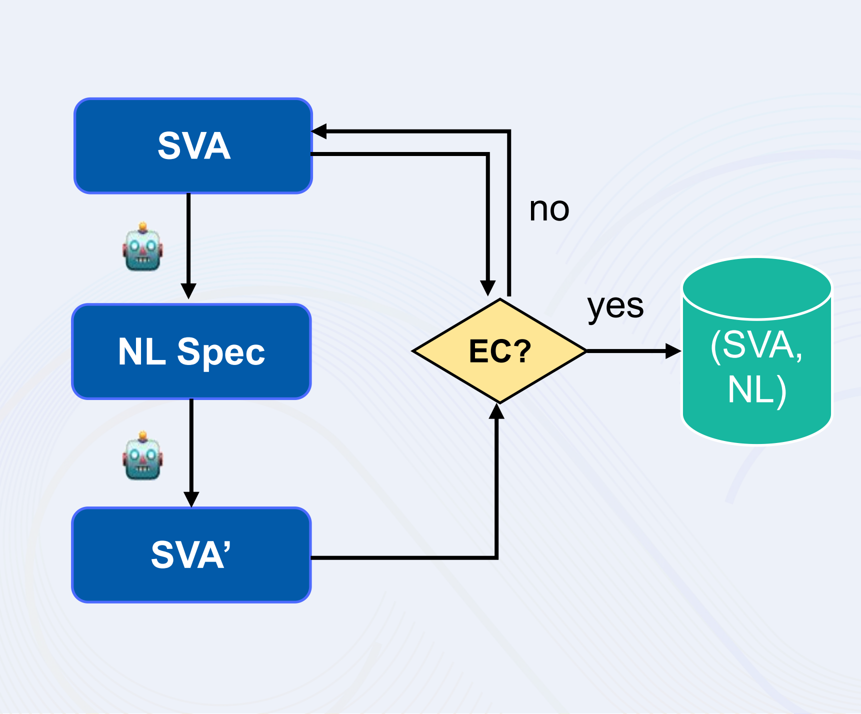
TEST

Techniques	Snytax	Functionality	Score
basic prompt	15	12	0.6
syntax_guarding	17	9	0.45
knowledge_appending	18	13	0.65
chain_of_thought	18	11	0.55
fewshot_learning	20	20	1
gpt4	20	20	1

➢ Method – Evaluation by Formal Checking

- **SVAC can check label correctness while bootstrapping.**
  - Self-instruct with LM is the common way to synthesize new data or label;
  - But traditional filtering techniques include only heuristics.

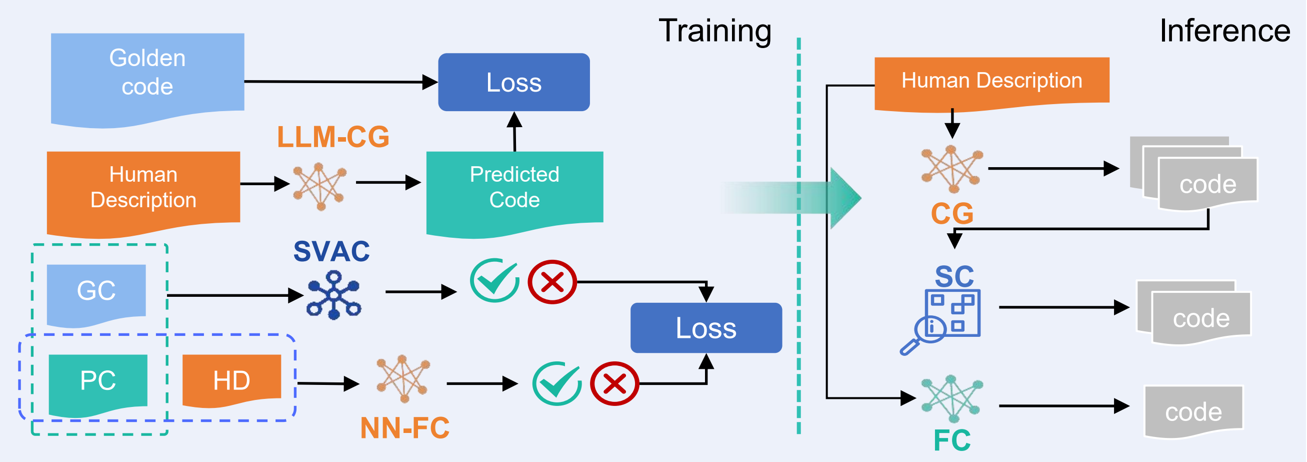




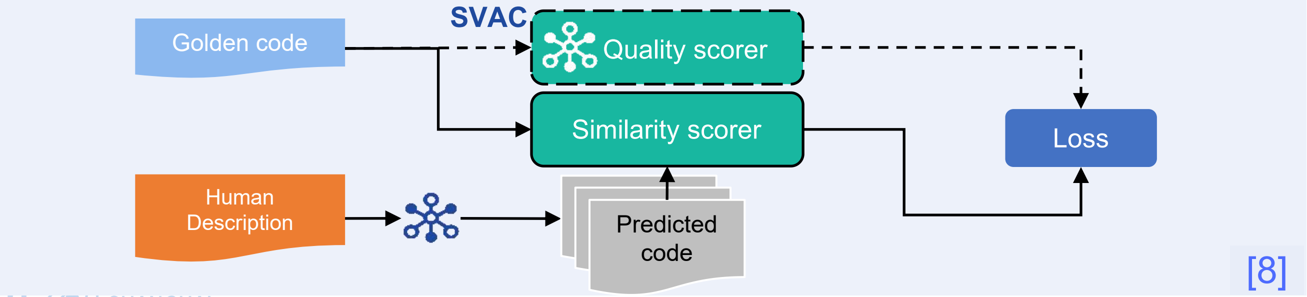
➢ Method – Evaluation by Formal Checking

◆ SVAC can enhance LLM performance.

A. Explicit Learning

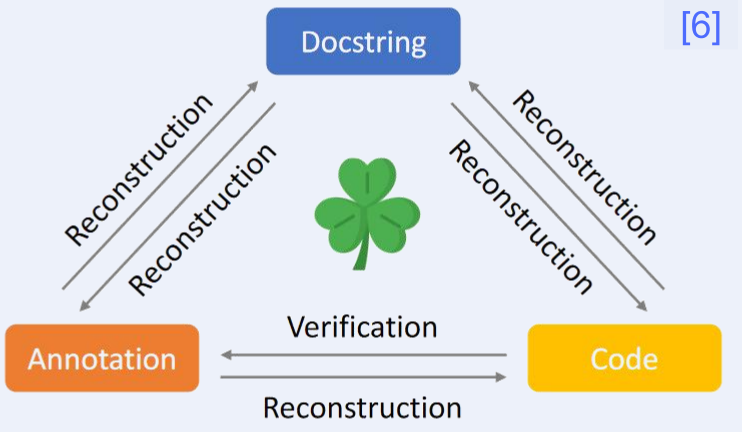


B. Implicit Learning

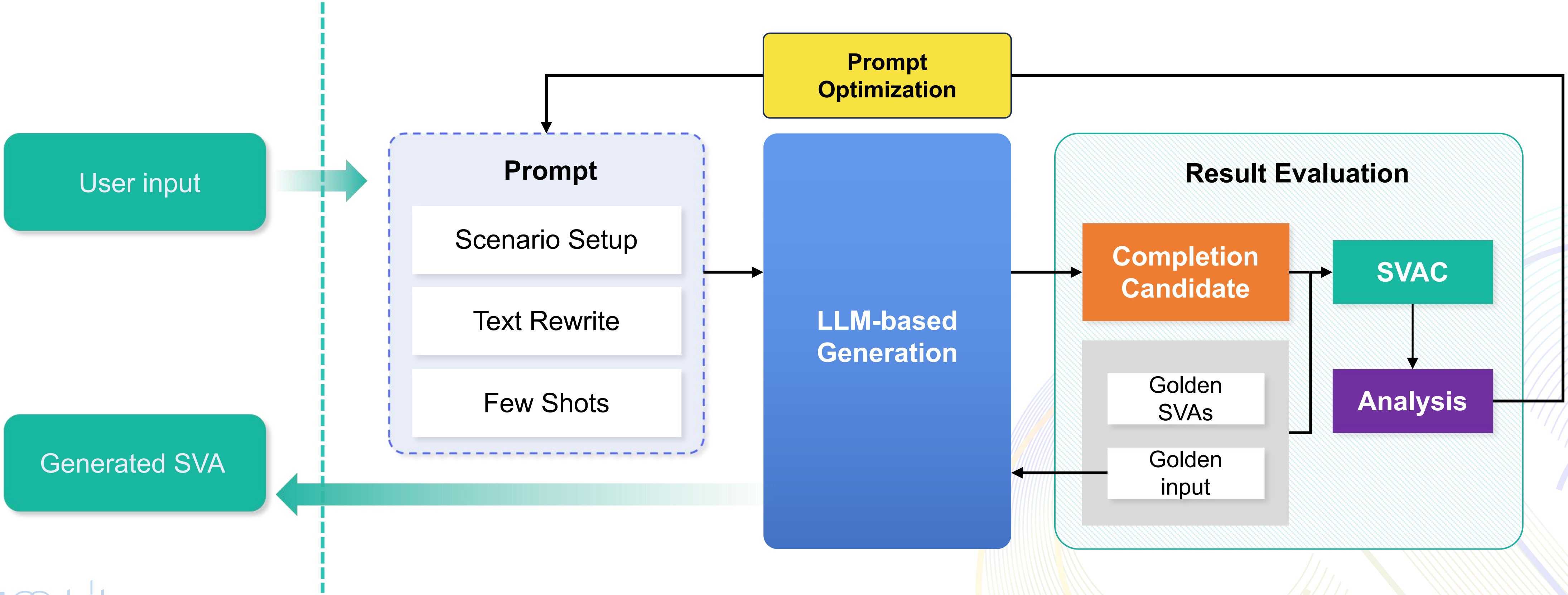


C. Post-training compute w/ SVAC:

- Reconstruction check;
- Consistency check;
- Check with golden model;

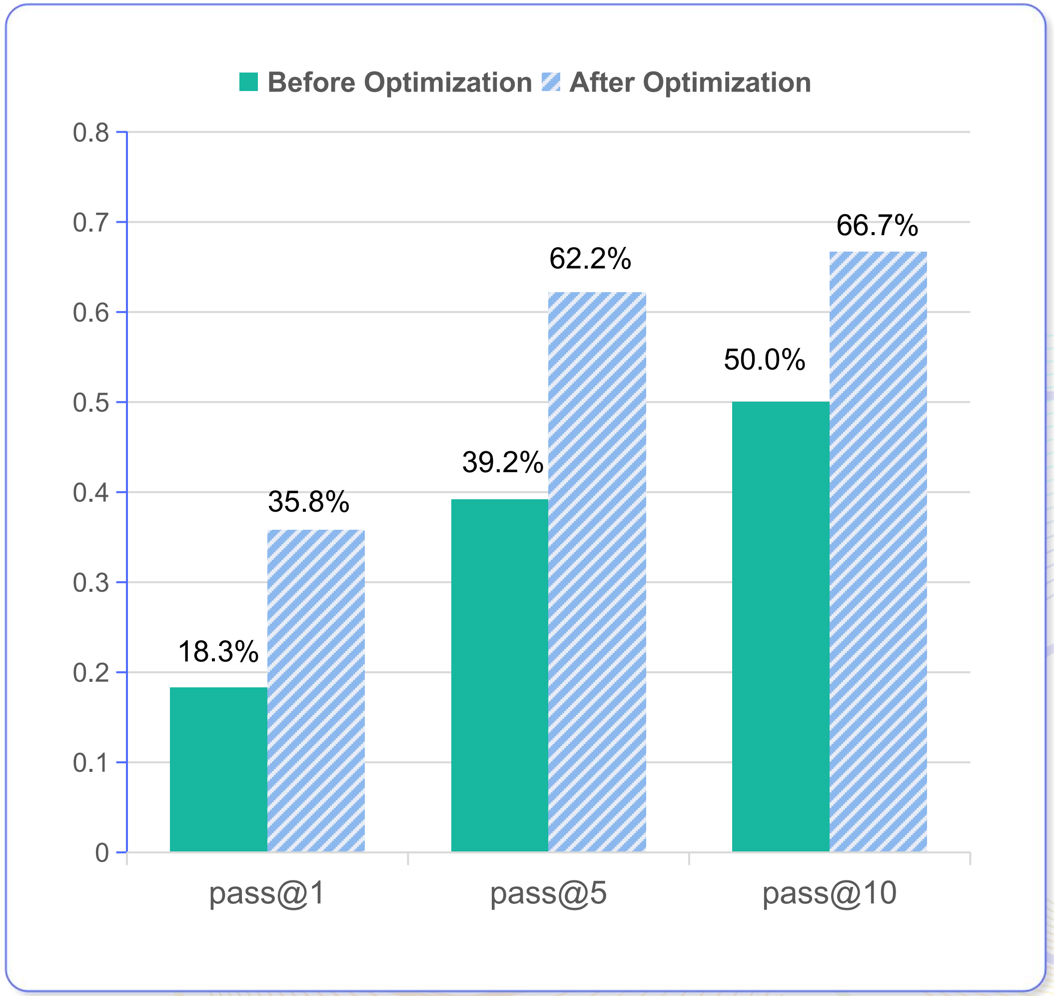


Method – Iterative Optimization



➢ Experiments

<table>
<tr><td rowspan="2">Beginner</td><td>Create a SystemVerilog assertion named "p_signal_follow" that with 3 signals (i.e., clk, src, dst) and includes an additional temporary variable 'tmp'. The assertion should ensure that when the 'src' signal changes at the positive edge of 'clk', the 'dst' signal should adopt the value of 'src' at the time of its change, within a timeframe ranging from 0 to 10 clock cycles after this change.</td></tr>
<tr><td>property p_signal_follow;
   bit tmp;
   @(posedge clk) ($changed(src),tmp=src) |-> ##[0:10] dst==tmp;
endproperty
assert property (p_signal_follow);</td></tr>
<tr><td rowspan="2">Experienced</td><td>Construct a SystemVerilog assertion named "p_rose_throughout" that involves three signals (i.e., clk, a, b, c). This assertion checks that whenever signal 'a' rises (transitions from low to high) at the positive edge of 'clk', the signal 'b' must be true continuously during the entire period when signal 'c' is false, which lasts for exactly one clock cycle following the rise of 'a'.</td></tr>
<tr><td>property p_rose_throughout;
   @(posedge clk) $rose(a) |-> b throughout !c [->1] ;
endproperty
assert property (p_rose_throughout);</td></tr>
<tr><td rowspan="2">Expert</td><td>Design a SystemVerilog assertion named "p_within_rst" that involves 6 signals (i.e., clk, rst_n, a, b, c, d). This assertion ensures that when signal 'a' falls (transitions from high to low) at the positive edge of 'clk', a specific sequence involving signals 'c' and 'd' must occur within a defined window of time marked by signal 'b' falling and then rising, provided that the reset signal 'rst_n' is high (active). Additionally, this behavior is only checked when the reset signal 'rst_n' is high, ensuring that the assertion is disabled (does not evaluate or enforce the rule) when 'rst_n' is low (reset condition).</td></tr>
<tr><td>property p_within_rst;
   @(posedge clk) disable iff(!rst_n) $fell(a) |-> ((c [->2] ##1 d) within
($fell(b) ##[5:10] $rose(b)));
endproperty
assert property (p_within_rst);</td></tr>
</table>



Conclusions

- Automated SVA generation with LLMs and evaluated by formal checking.
- Improved the baseline by 58% for Pass@5 and 33% for Pass@10 through testing SLEG with Sanechips Tech. in real IC projects.
- Future: Continue partnership with Sanenchips to expand LLM+FV design paradigm for other generation tasks (e.g. SV/RTL/SystemC) and application scenarios (e.g., data bootstrapping, enhanced prediction).

2025 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION CHINA

accellera SYSTEMS INITIATIVE

Thank you

关注官方微信
了解芯华章最新动向

16 Apr 2025 上海 SHANGHAI

➢ References

1. Witharana, Hasini, Yangdi Lyu, Subodha Charles, and Prabhat Mishra. “A Survey on Assertion-Based Hardware Verification.” ACM Computing Surveys 54, no. 11s (January 31, 2022): 1–33. https://doi.org/10.1145/3510578.

2. Ren, Shuo, Daya Guo, Shuai Lu, Long Zhou, Shujie Liu, Duyu Tang, Neel Sundaresan, Ming Zhou, Ambrosio Blanco, and Shuai Ma. “CodeBLEU: A Method for Automatic Evaluation of Code Synthesis.” arXiv, September 27, 2020. http://arxiv.org/abs/2009.10297.

3. Chen, Mark, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, et al. “Evaluating Large Language Models Trained on Code.” arXiv, July 14, 2021. http://arxiv.org/abs/2107.03374.

4. Yang, Sichao, and Ye Yang. “FormalEval: A Method for Automatic Evaluation of Code Generation via Large Language Models.” In 2024 2nd International Symposium of Electronics Design Automation (ISEDA), 660–65. IEEE, 2024.

5. Lu, Yao, Shang Liu, Qijun Zhang, and Zhiyao Xie. “RTLLM: An Open-Source Benchmark for Design RTL Generation with Large Language Model.” arXiv, November 11, 2023. http://arxiv.org/abs/2308.05345.

6. Mneimneh, M.N., and K.A. Sakallah. “Principles of Sequential-Equivalence Verification.” IEEE Design and Test of Computers 22, no. 3 (May 2005): 248–57. https://doi.org/10.1109/MDT.2005.68.

7. Joost-Pieter Katoen. “Introduction to Model Checking.” RWTH Aachen University. https://moves.rwth-aachen.de/teaching/ss-16/ss16introduction-to-model-checking.

8. Liu, Shang, Wenji Fang, Yao Lu, Qijun Zhang, Hongce Zhang, and Zhiyao Xie. “RTLCoder: Outperforming GPT-3.5 in Design RTL Generation with Our Open-Source Dataset and Lightweight Solution.” arXiv, January 17, 2024. http://arxiv.org/abs/2312.08617.