ReChisel: Effective Automatic Chisel Code Generation by LLM with Reflection

Juxin Niu[1], Xiangfeng Liu[1,2], Dan Niu[3], Xi Wang[3,4], Zhe Jiang[3,4], Nan Guan[1]

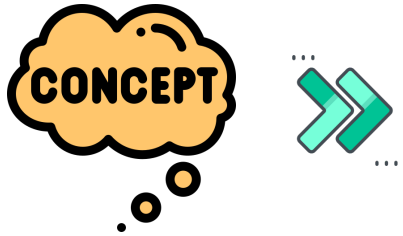
[1] City University of Hong Kong, Hong Kong SAR
[2] Northeastern University, China
[3] Southeast University, China
[4] National Center of Technology Innovation for EDA, China

juxin.niu@my.cityu.edu.hk

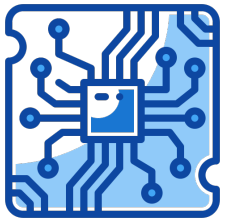
62 THE CHIPS TO SYSTEMS CONFERENCE

SPONSORED BY CEDA IEEE Council on Electronic Design Automation SIGDA acm

HARDWARE DESCRIPTION LANGUAGE (HDL)



Hardware Description Language
(HDL)



- ✓ Behavior
- ✓ Structure
- ✓ Function

Hardware Description Language (HDL)

Hardware Description Language
(HDL)

- ✓ Behavior
- ✓ Structure
- ✓ Function

Verilog vs **Chisel**

- Traditional HDL
- Widely-used

- Modern HDL
- Future of agile design

CHISEL HDL — Next Generation HDL

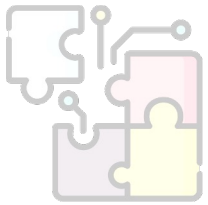
- ***Why?***

High-Level Abstraction

Use advanced language constructs to simplify complex hardware modeling.

CHISEL HDL — Next Generation HDL

- ***Why?***



High-Level Abstraction

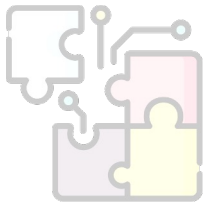
Use advanced language constructs to simplify complex hardware modeling.

Concise & Maintainable Code

Offer robust APIs to minimize boilerplate for clarity and readability.

CHISEL HDL — Next Generation HDL

- ***Why?***



High-Level Abstraction

Use advanced language constructs to simplify complex hardware modeling.

Concise & Maintainable Code

Offer robust APIs to minimize boilerplate for clarity and readability.

Scalable & Reusable Design

Support objective-oriented features to efficiently scale for larger projects.

CHISEL HDL — Next Generation HDL

- *Chisel vs. Verilog*

1 Registers with Enable and Reset

WITH CHISEL:

```
io.q := RegEnable(io.d, 0.U, io.en)
```

WITH VERILOG:

```
always @(posedge clk) begin
  if (reset)
    q <= 8'b0;
  else if (en)
    q <= d;
  // else q retains its value
end
```

2 Integer to One-Hot

WITH CHISEL:

```
io.out := UIntToOH(io.in)
```

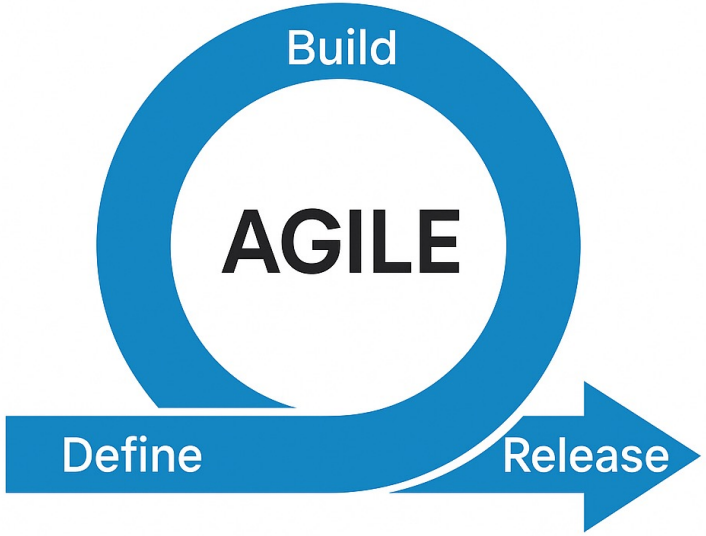
WITH VERILOG:

```
always @(*) begin
  out = 16'b0;
  case (in)
    4'd0 : out = 16'h0001;
    4'd1 : out = 16'h0002;
    // …
    4'd14: out = 16'h4000;
    4'd15: out = 16'h8000;
  endcase
end
```

CHISEL HDL — Future of Agile Design

Concise &
Maintainable Code

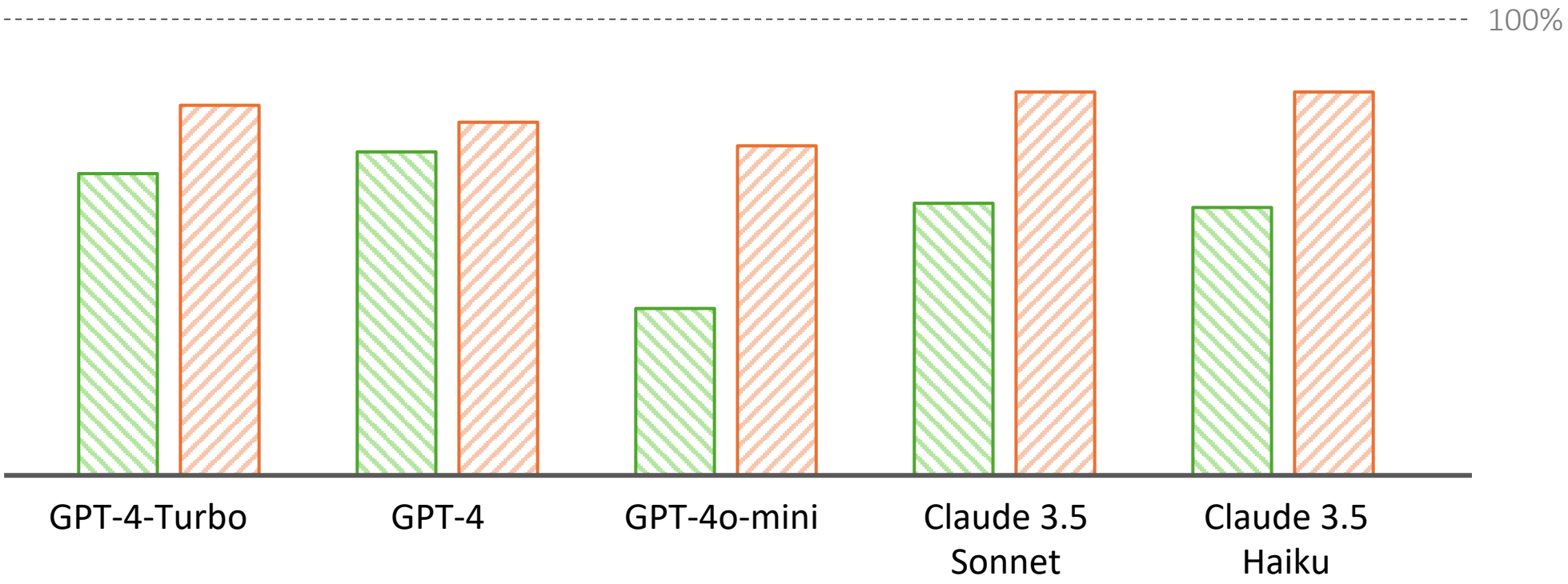
High-Level
Abstraction



Scalable & Reusable
Design

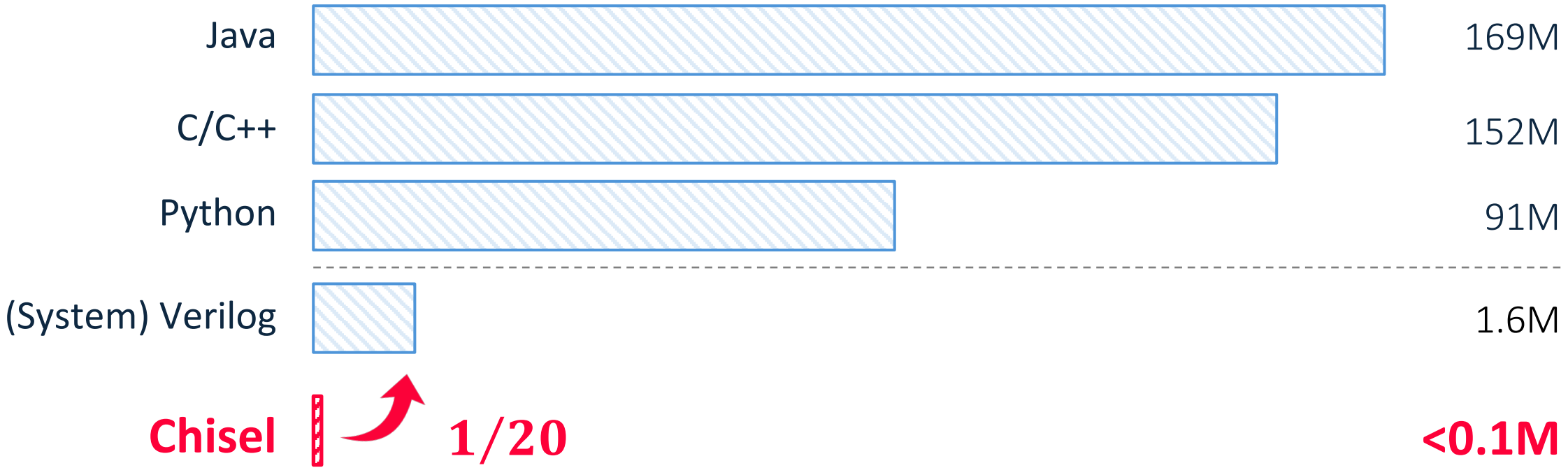
CHISEL HDL — Limited with LLMs

- ***Inadequate Baseline Capability***



CHISEL HDL — Limited with LLMs

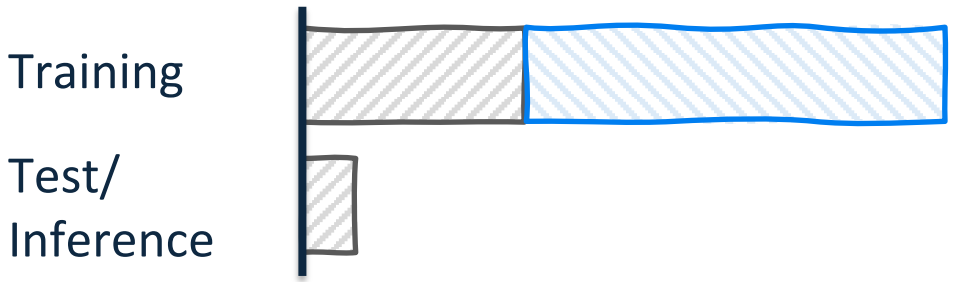
- ***Why? — Scarce Training Data***



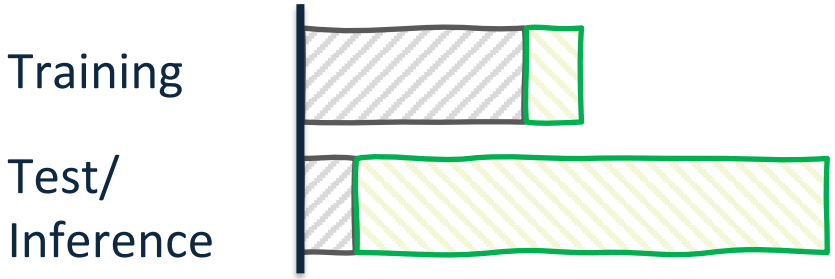
APPROACH

- ***From Training-Time to Test-Time***

Training-Time Strategy



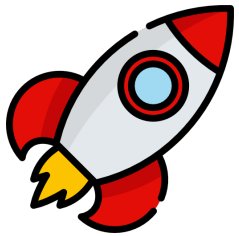
Test-Time Strategy



Test-Time: Exploring how good LLMs can be after training.

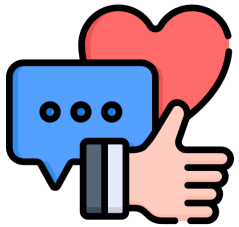
Approach

- *Advantages of Test-Time Strategy*



Resource Efficiency

Great Potential

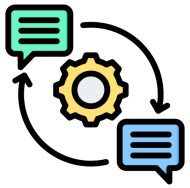


Quick Feedback

Extended Participants

APPROACH

- ***Insights to Test-Time Strategy***

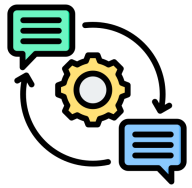


LLM Self-Reflection

Enable dynamic reasoning to detect and correct logical flaws autonomously.

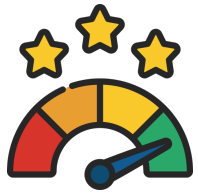
Approach

- ***Insights to Test-Time Strategy***



LLM Self-Reflection

Enable dynamic reasoning to detect and correct logical flaws autonomously.



Compiler Feedback

Offer precise syntax error reports with explanations and suggested fixes.

ILLUSTRATION

```
class Slicing extends Module {
  val io = IO(new Bundle{
    val in       = Input (UInt(16.W))
    val startIdx = Input (UInt( 4.W))
    val out      = Output(UInt( 4.W))
  })

  io.out := io.in(io.startIdx + 3.U, io.startIdx)
}
```

ILLUSTRATION

- ***Compiler Feedback***

```
class Slicing extends Module {
  val io = IO(new Bundle{
    val in       = Input (UInt(16.W))
    val startIdx = Input (UInt( 4.W))
    val out      = Output(UInt( 4.W))
  })

  io.out := io.in(io.startIdx + 3.U, io.startIdx)
}
```



```
[error] path\to\Main.scala:

overloaded method apply with alternatives:
    (x: BigInt, y: BigInt) chisel3.UInt <and>
    (x: Int, y: Int) chisel3.UInt.
cannot be applied to (chisel3.UInt, chisel.3UInt)

io.out := io.in(io.startIndx + 3.U, io.startIdx)
          ^

(Compile / CompileIncreamental) Compilation failed
```

ILLUSTRATION

- ***Self-Reflection***

```
[error] path\to\Main.scala:

overloaded method apply with alternatives:
    (x: BigInt, y: BigInt) chisel3.UInt <and>
    (x: Int, y: Int) chisel3.UInt.
cannot be applied to (chisel3.UInt, chisel.3UInt)

io.out := io.in(io.startIndx + 3.U, io.startIdx)
          ^

(Compile / CompileIncreamental) Compilation failed
```

Revision Plan by LLMs:

Root Cause: The error arises because you are trying to use chisel.UInt types as indices to slice an array (or bit vector). Chisel requires these indices to be integers (Int) because they must be constant at compiler time.

Solution: To address, right-shift the 16-bit input by the chisel.UInt index to align the 4-bit window at bit-position 3 down to 0, then use a constant slice.

ILLUSTRATION

- ***Self-Reflection***

```
[error] path\to\Main.scala:

overloaded method apply with alternatives:
    (x: BigInt, y: BigInt) chisel3.UInt <and>
    (x: Int, y: Int) chisel3.UInt.
cannot be applied to (chisel3.UInt, chisel.3UInt)

io.out := io.in(io.startIndx + 3.U, io.startIdx)
          ^

(Compile / CompileIncreamental) Compilation failed
```

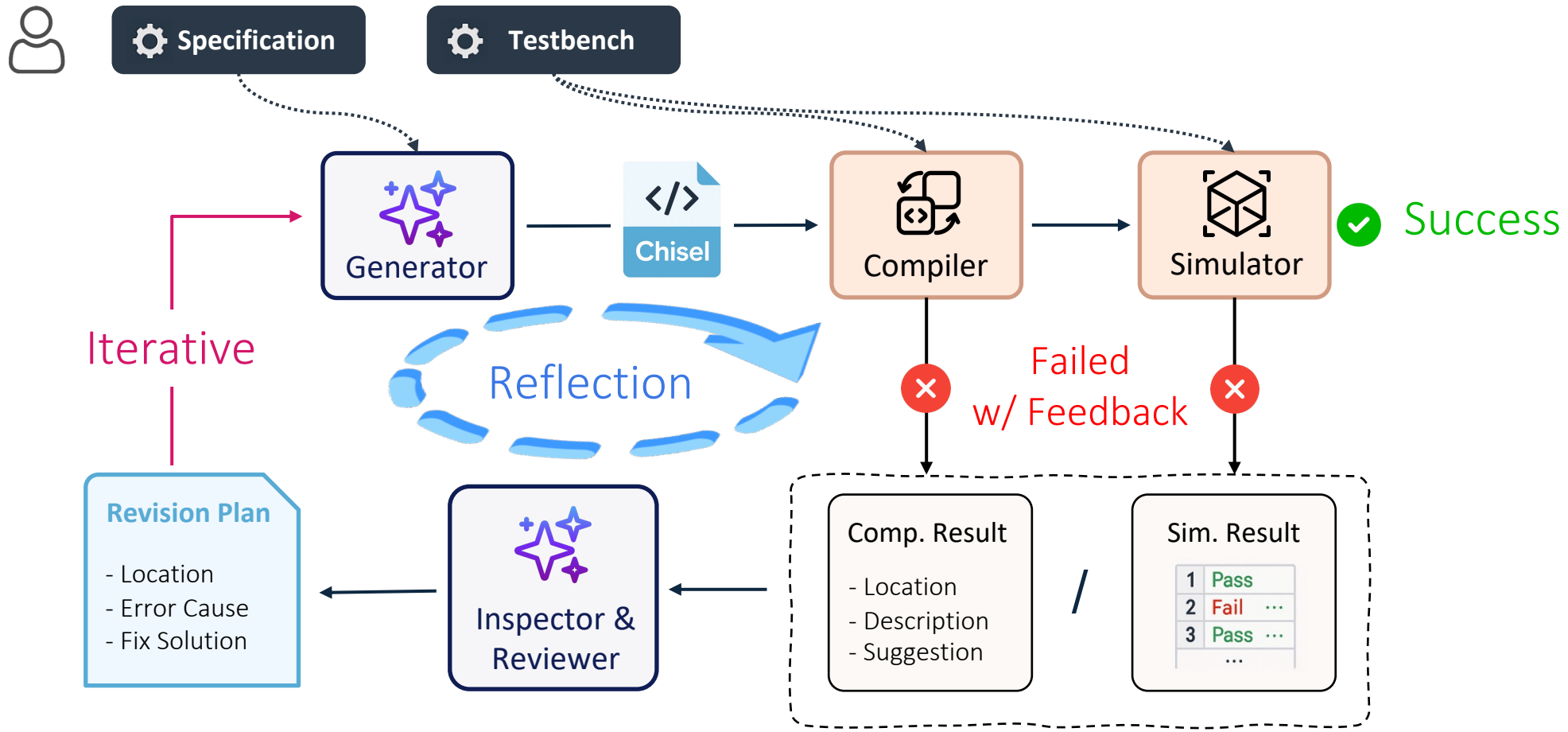
Revision Plan by LLMs:

Root Cause: The error arises because you are trying to use chisel.UInt types as indices to slice an array (or bit vector). Chisel requires these indices to be integers (Int) because they must be constant at compiler time.

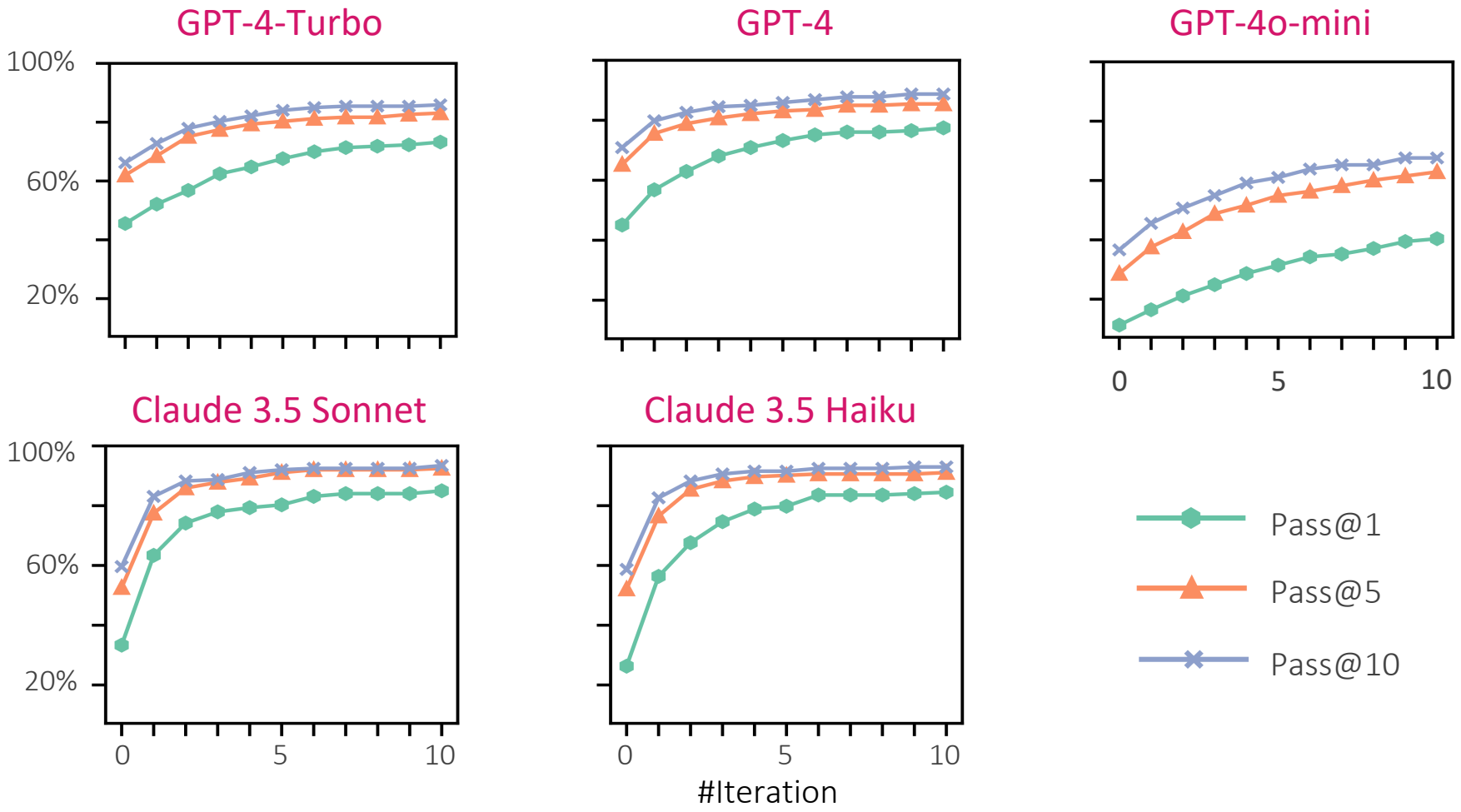
Solution: To address, right-shift the 16-bit input by the chisel.UInt index to align the 4-bit window at bit-position 3 down to 0, then use a constant slice.

```
io.out := (io.in >> io.startIdx)(3,0)
```

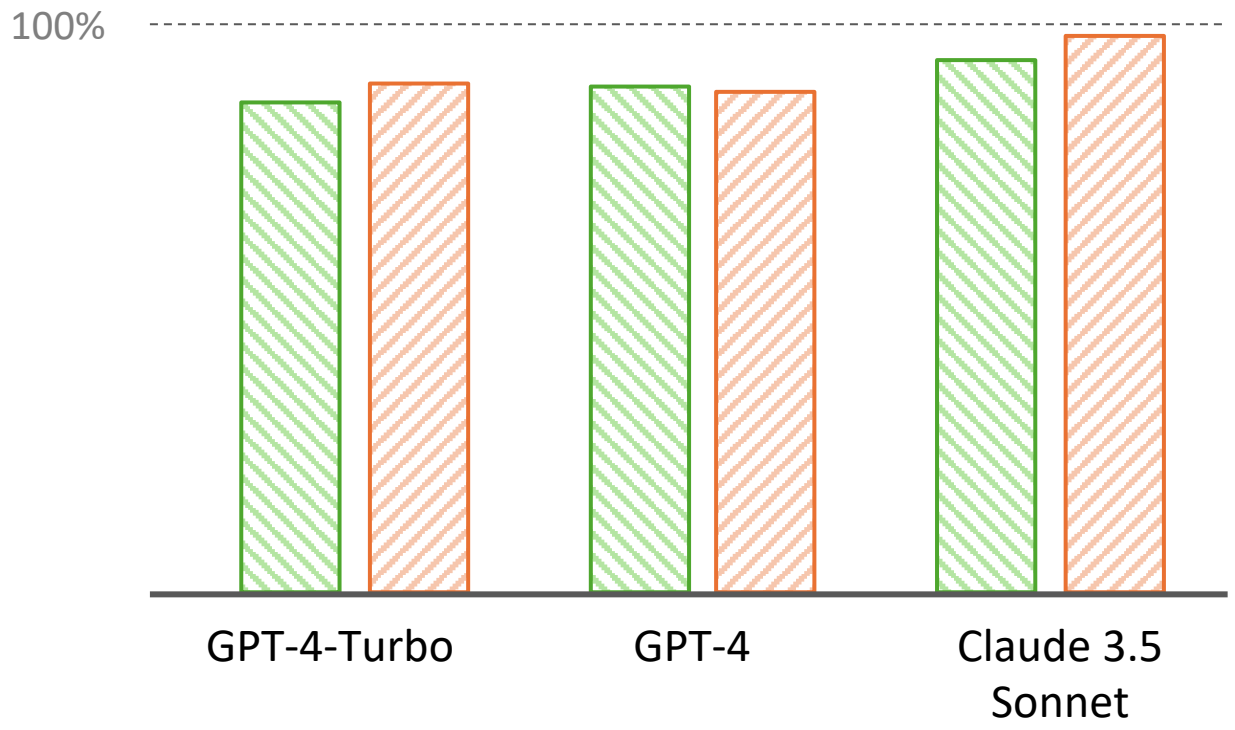
ReChisel Workflow



ReChisel Evaluation — Success Rate



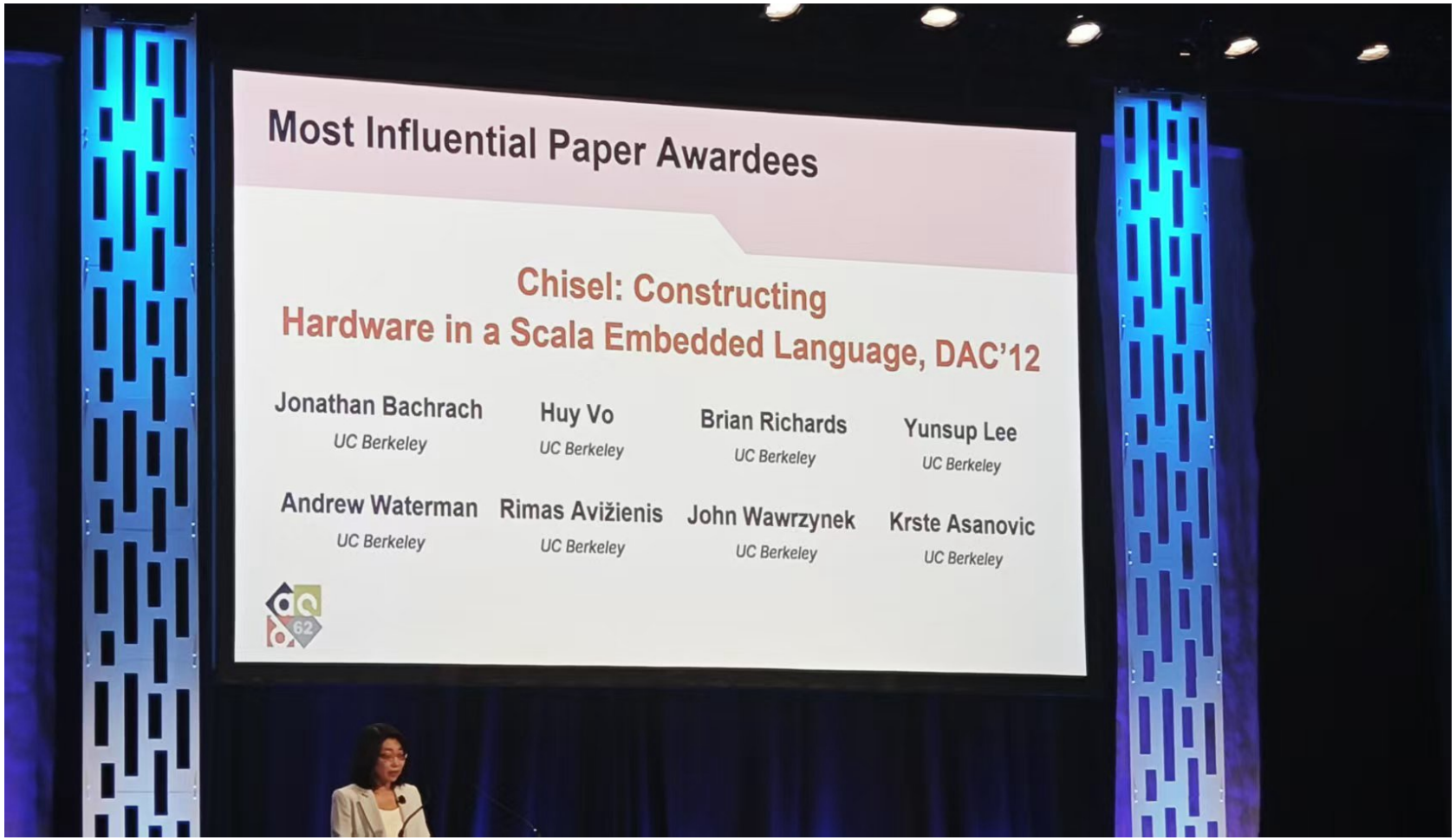
ReChisel Evaluation — vs. Verilog



ReChisel (Ours)

AutoChip
SOTA of Automatic Verilog Code Generation

CHISEL IN DAC 62



AI

Security

Systems

Thanks

EDA

Design

62 THE CHIPS TO SYSTEMS CONFERENCE

SPONSORED BY CEDA® IEEE Council on Electronic Design Automation

sigda acm