

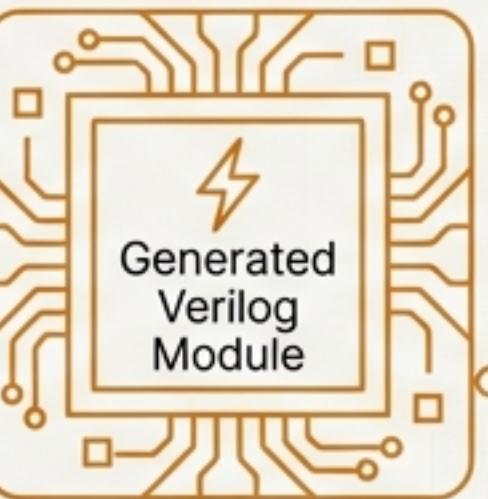


Natural
Language
Prompt

AutoChip: A GenAI-Powered Framework for Automated Verilog Generation

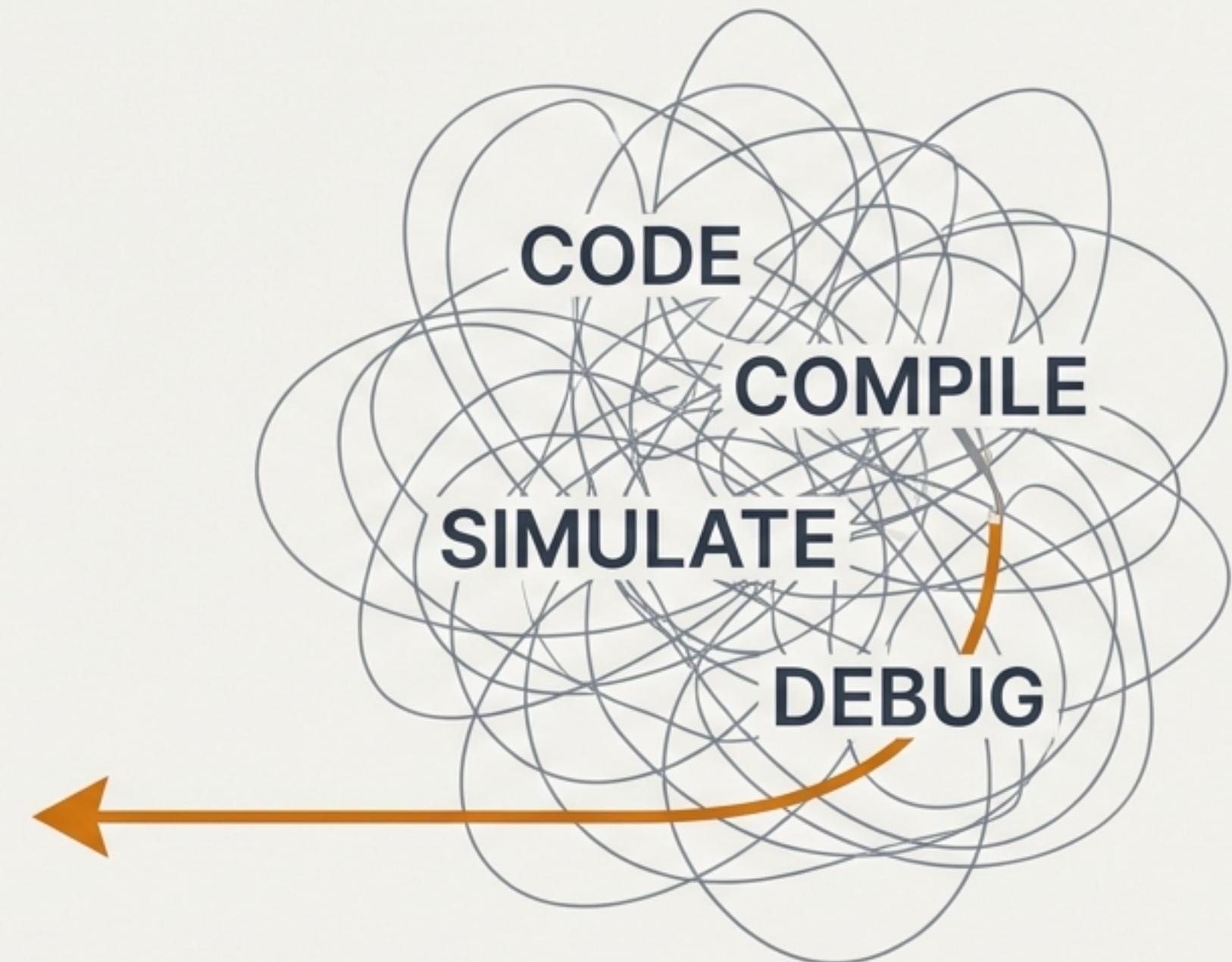
Lab Objective: To implement and evaluate an iterative, testbench-driven workflow for generating and debugging Verilog code using Large Language Models (LLMs).

- Explore the automation of hardware design from a natural language prompt and a Verilog module template.
- Utilize a feedback loop of compilation and simulation with Icarus Verilog to iteratively refine AI-generated code.
- Assess the capability of GenAI on various design complexities, from simple combinational logic to complex finite-state machines (FSMs).
- Systematically evaluate code candidates using a quantitative ranking mechanism based on verification results.



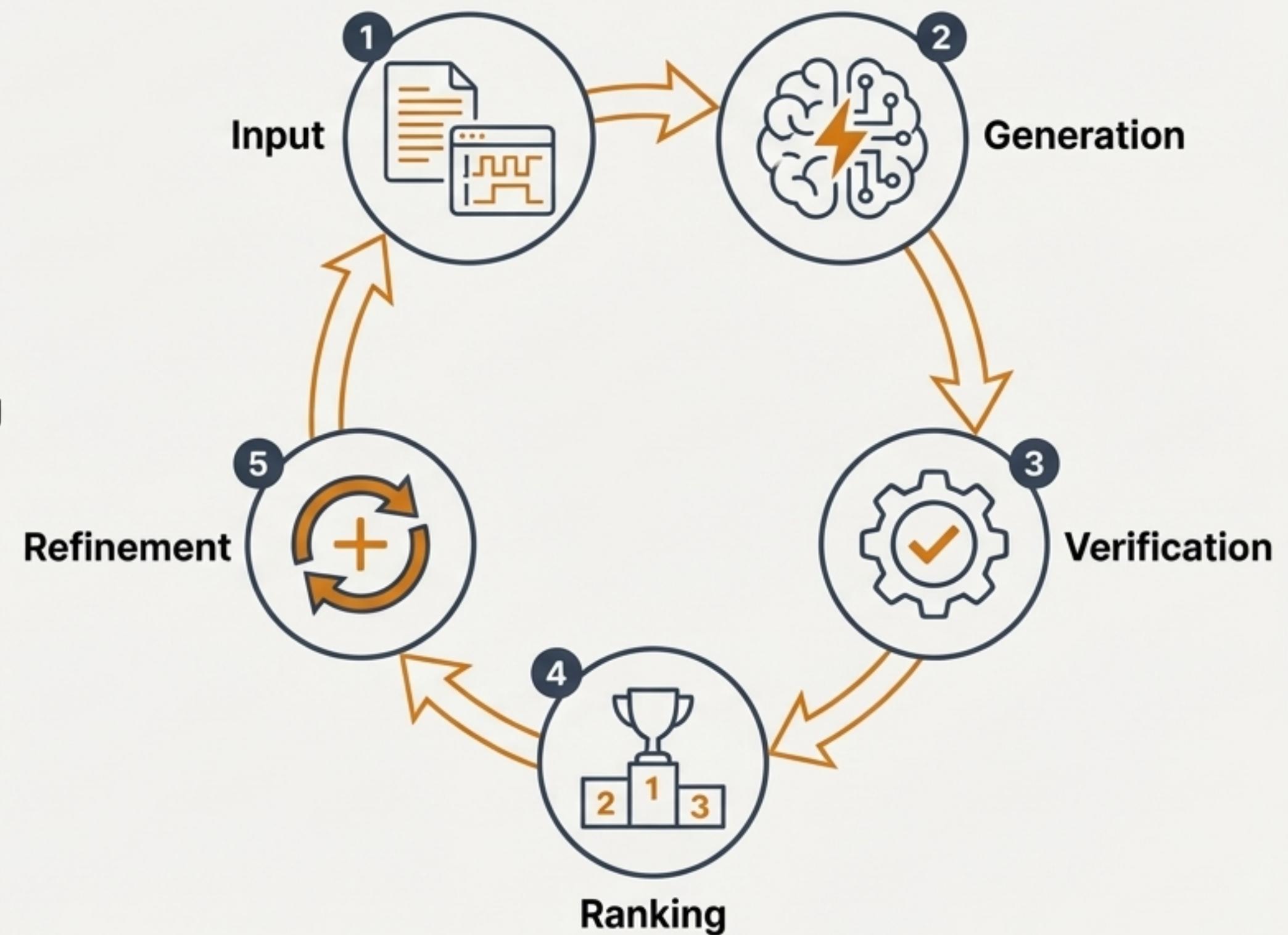
The Challenge in Modern Hardware Design

- The traditional design cycle is a manual, time-intensive loop of coding, compiling, simulating, and debugging.
- Writing Verilog and other Hardware Description Languages (HDLs) is intricate, requiring precise syntax and logical correctness.
- Debugging hardware logic is often non-intuitive and demands significant engineering expertise and effort to resolve.
- The central challenge: Can Generative AI be harnessed to automate the code generation and correction process, thereby accelerating the hardware design lifecycle?



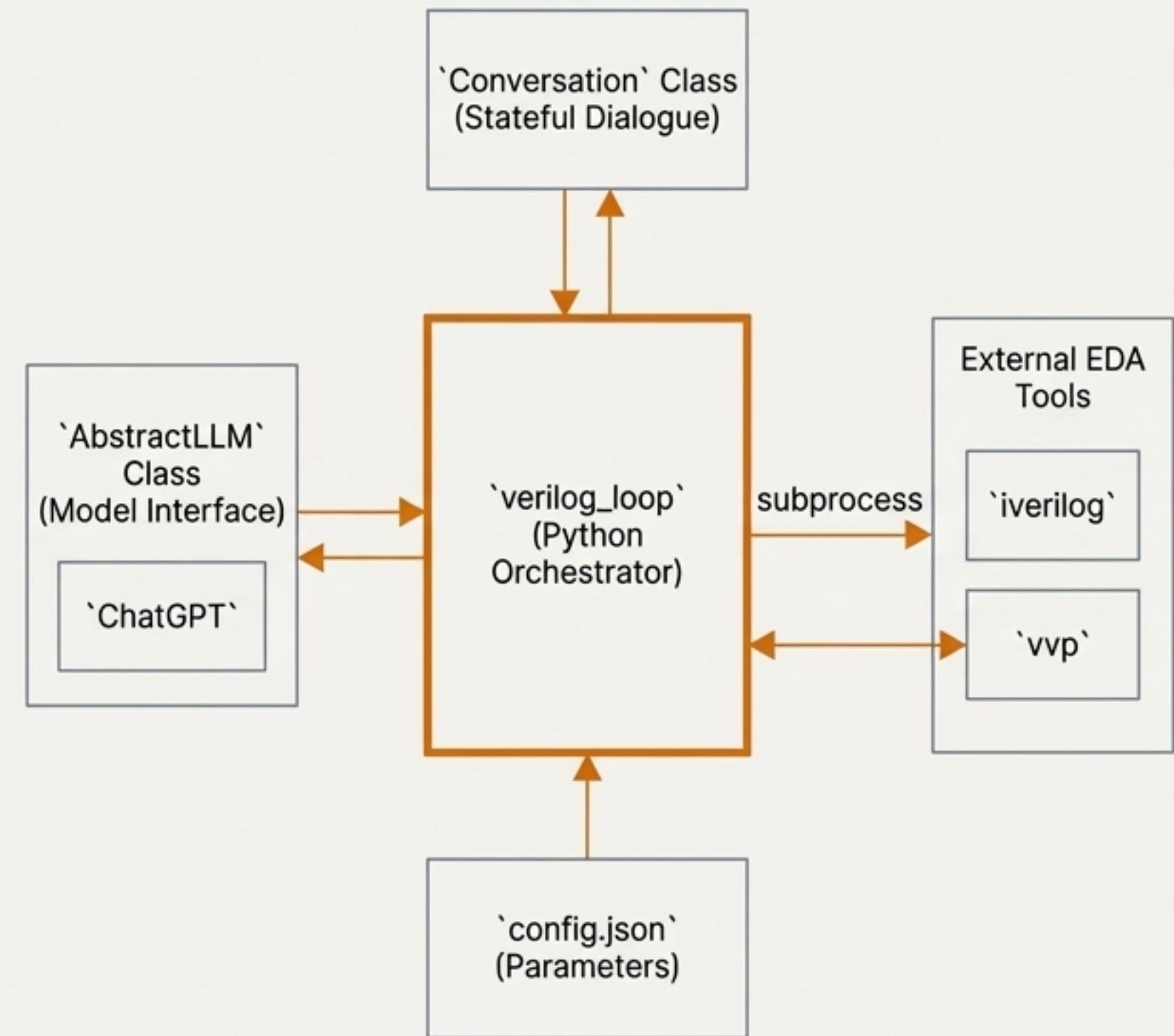
The AutoChip Workflow: An Iterative Feedback Loop

- **Input:** The process begins with a design specification (the initial prompt) and a verification testbench ('.sv' file).
- **Generation:** A Large Language Model generates multiple Verilog code candidates based on the current state of the conversation.
- **Verification:** Each candidate is automatically compiled and simulated using the external EDA: Icarus Verilog (iverilog').
- **Ranking:** Candidates are scored based on parsing success, compilation status (errors/warnings), and simulation output mismatches.
- **Refinement:** The highest-ranked response and its corresponding tool feedback are used to construct a new prompt for the LLM in the next iteration.



Under the Hood: The AutoChip System Architecture

- The system is orchestrated by a central Python script (``verilog_loop``) that manages the entire iterative process.
- A `Conversation` class maintains the stateful dialogue with the GenAI model, preserving the history of prompts and feedback.
- An extensible `AbstractLLM` class provides a common interface, allowing for easy integration with various model families (e.g., `ChatGPT`).
- Python's `subprocess` module is used to call external EDA tools (`iverilog`, `vvp`) and capture their stdout/stderr streams for analysis.
- Configuration is managed via a `config.json` file, controlling parameters such as the model ID, number of iterations, and number of candidates.



GenAI as a Verilog Co-Pilot

- The LLM's primary function is to act as a Verilog "autocomplete and correction engine."
- It performs initial code generation from a high-level functional description provided in the initial prompt.
- Critically, it also functions as an automated debugger, attempting to fix code based on structured feedback from the Verilog compiler and simulator.

System Prompt: "You are an autocomplete engine for Verilog code... you will suggest corrections... Format your response as Verilog code containing the end to end corrected module..."

Prompt

Generate a 4-bit counter with reset based on this functional description...



Generated Verilog Code

```
`module counter (
    input clk, rst,
    output reg [3:0] count);

    always @(posedge clk or posedge rst)
    begin
        if (rst) count <= 4'b0000;
        else count <= count + 1;
    end
endmodule
```

Compiler Error Feedback

⚠ ERROR:
FILE: counter.v
LINE: 4 - 'count' is
not declared as a reg
type. Use 'output reg'
or similar.



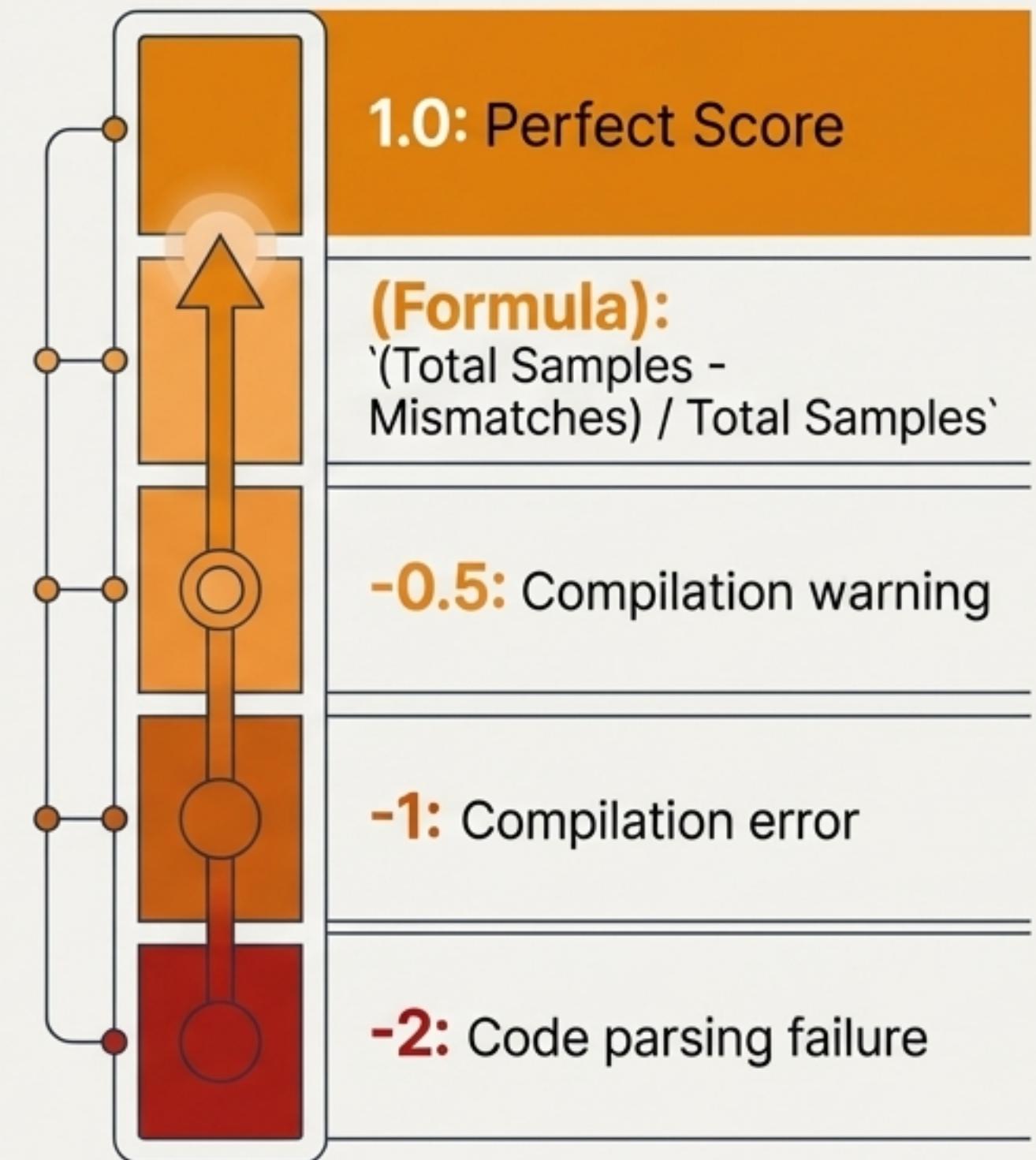
Corrected Verilog Code

```
`module counter (input clk, rst,
    output reg [3:0] count);

    always @(posedge clk or posedge rst)
    begin
        if (rst) count <= 4'b0000;
        else count <= count + 1;
    end
endmodule
```

Automated Verification and Quantitative Ranking

- Verification is driven entirely by a user-provided testbench, establishing it as the definitive "source of truth."
- A multi-stage ranking function (calculate_rank) quantitatively evaluates each AI-generated response:
 - **Rank -2:** Code parsing failure (no valid Verilog module found in the response).
 - **Rank -1:** Compilation error returned by `iverilog`.
 - **Rank -0.5:** Compilation warning returned by `iverilog`.
- For code that compiles and runs, the rank is calculated based on simulation results: $\text{Rank} = (\text{Total Samples} - \text{Mismatches}) / \text{Total Samples}$.
- A perfect score of **1.0** indicates that the testbench passed completely with zero mismatches, signaling success.



Performance Across Different Design Complexities



Success



7420 NAND Gate

Rank 1.0



Rule 90.

Rank 1.0

****Success on Combinational & Simple Sequential Logic:** The system consistently achieved a perfect rank of 1.0 for simpler designs like the 7420 NAND gate chip (0 mismatches) and the Rule 90 cellular automaton (0 mismatches), often on the first attempt.



Improvement



****Iterative Improvement on FSMs:** For the more complex Arbiter FSM, the system demonstrated clear iterative improvement, reducing simulation mismatches from 60 to 56 across iterations, though it did not achieve a perfect score.

****Candidate Exploration is Key:** Generating multiple candidates (e.g., 5 per iteration) proved effective, increasing the probability of finding a viable or improved solution in each loop.



Struggle

Sequence Detectors

Non-overlapping

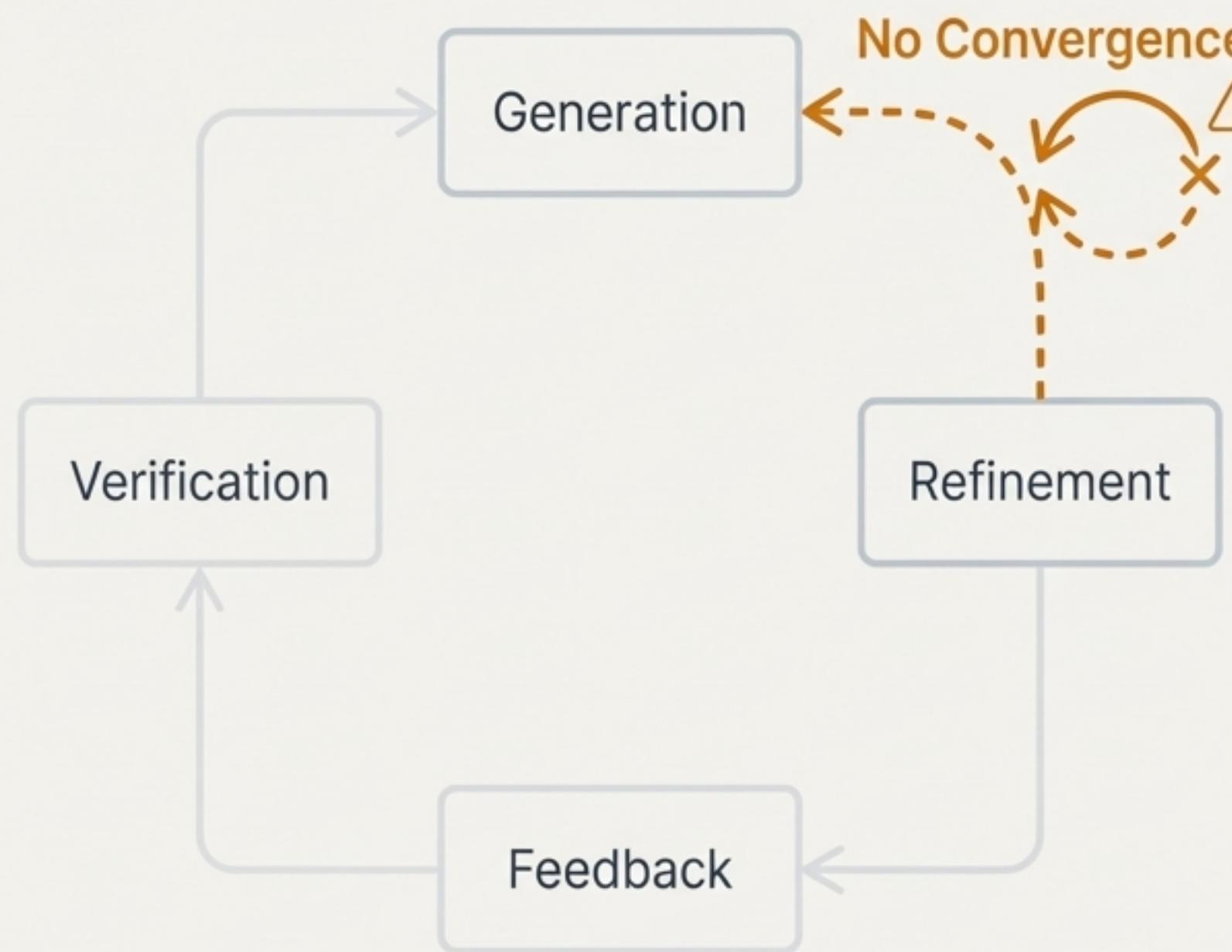
Rank ~0.657

Overlapping

Rank ~0.911

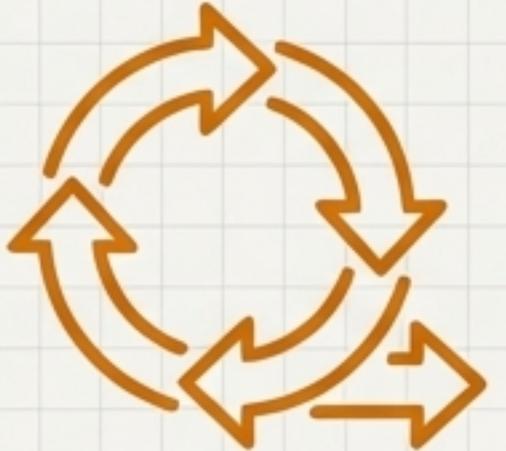
****Struggle with Complex FSMs:** The system failed to achieve a perfect score for the non-overlapping and overlapping sequence detector FSMs, with best ranks of ~0.657 and ~0.911 respectively.

Current Limitations and Remaining Challenges

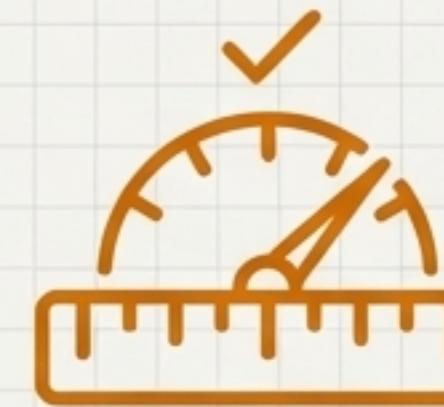


- The system struggles to converge to a correct solution (rank 1.0) for **complex sequential logic**, such as the non-overlapping and overlapping sequence detectors.
- The final outcome is highly sensitive to the quality and coverage of the provided **testbench**. An incomplete testbench can validate incorrect code.
- Feedback, while structured, is limited to raw tool output and may lack the high-level design context needed to guide the LLM toward an optimal fix.
- There is no guarantee of convergence; the system can become stuck in a loop of incorrect solutions or exhaust the maximum number of iterations.

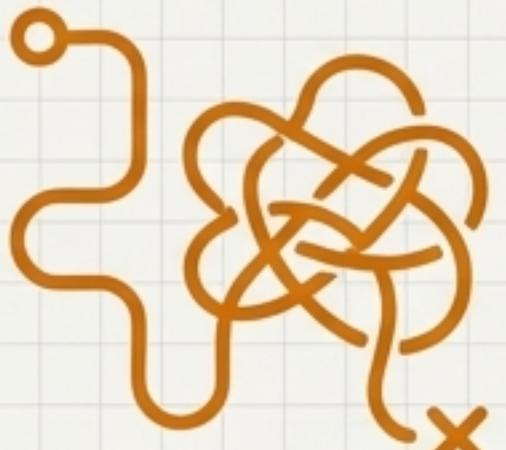
Key Takeaways and Lessons Learned



A testbench-driven, iterative framework is a robust and effective method for applying GenAI to HDL generation and debugging.



The quantitative ranking of candidates is essential for systematically guiding the automated debugging process toward a correct solution.

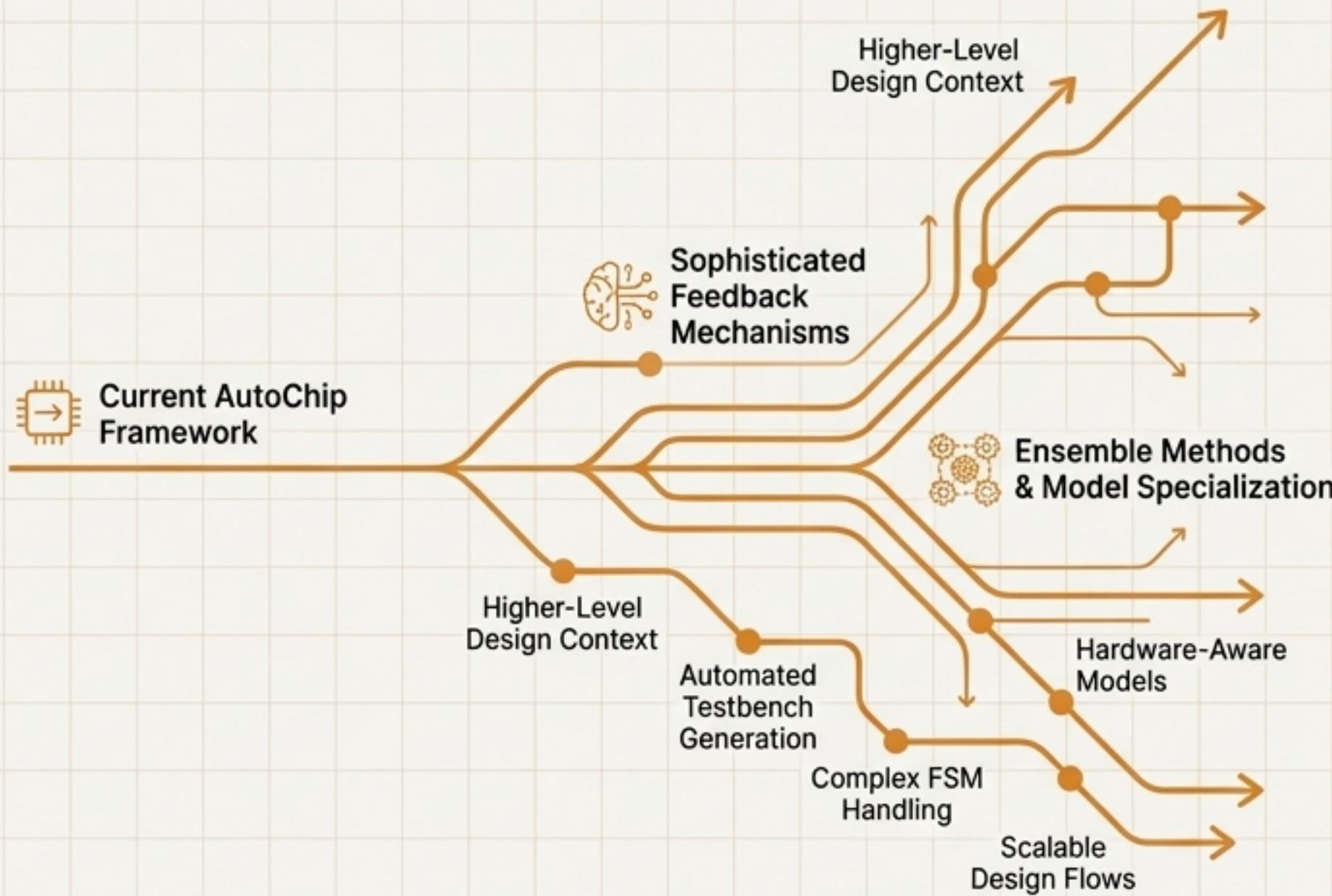


While powerful for boilerplate code and simpler logic, GenAI still faces significant challenges with the nuanced state transitions and logic of complex FSMs.



The quality of the initial prompt and, most critically, the comprehensiveness of the testbench are the most important factors for success in this automated workflow.

Conclusion: The Future of AI in Hardware Design



- AutoChip validates the potential of using LLMs as automated coding and debugging assistants within a structured, test-driven hardware design flow.
- The framework effectively accelerates the design process for low-to-medium complexity Verilog modules, successfully handling combinational and simpler sequential logic.
- Future work should focus on creating more sophisticated feedback mechanisms that provide higher-level design context beyond raw tool logs.
- Exploring ensemble methods, where different models are used for generation versus refinement, could further improve performance and overcome current limitations.