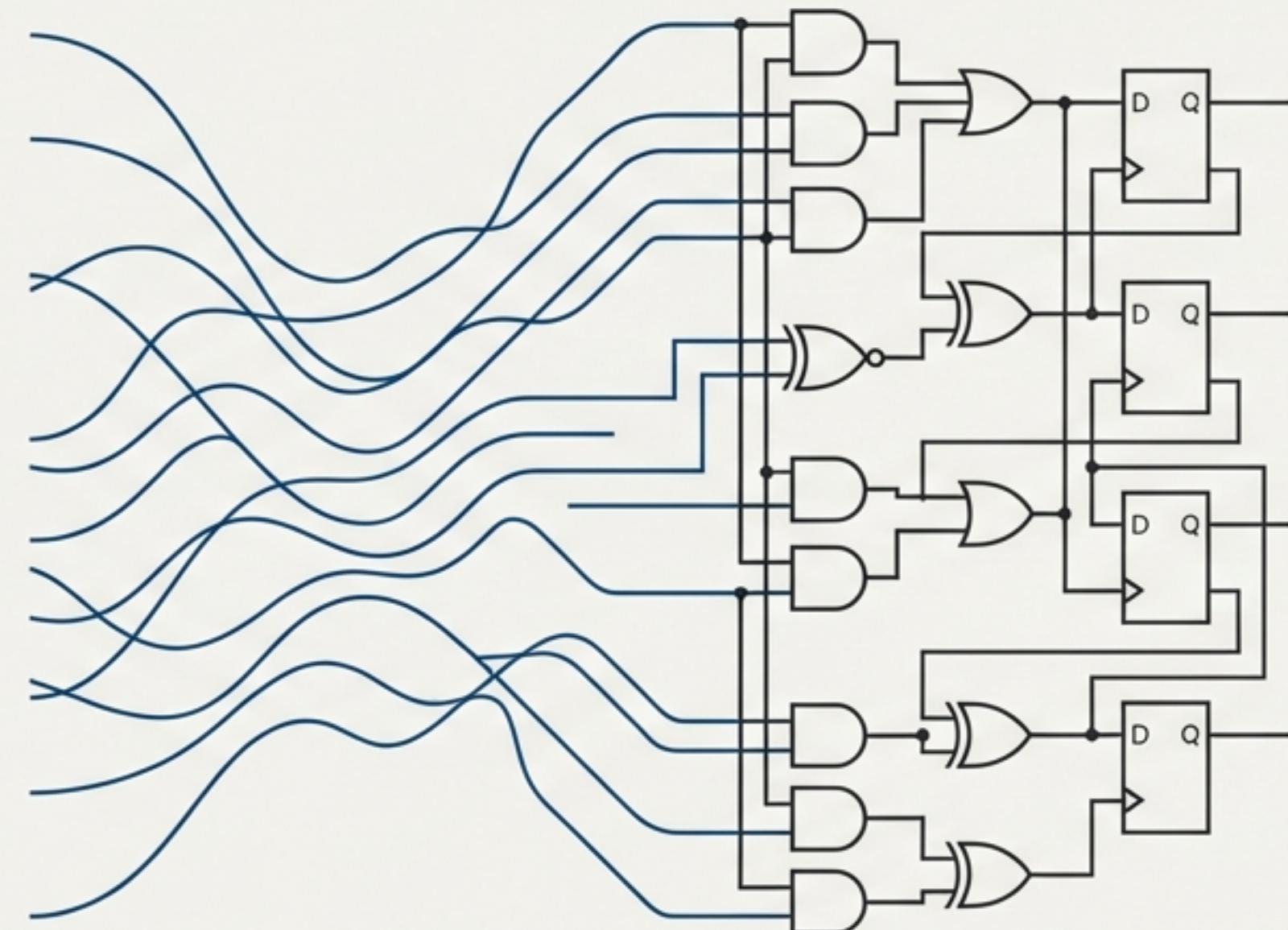
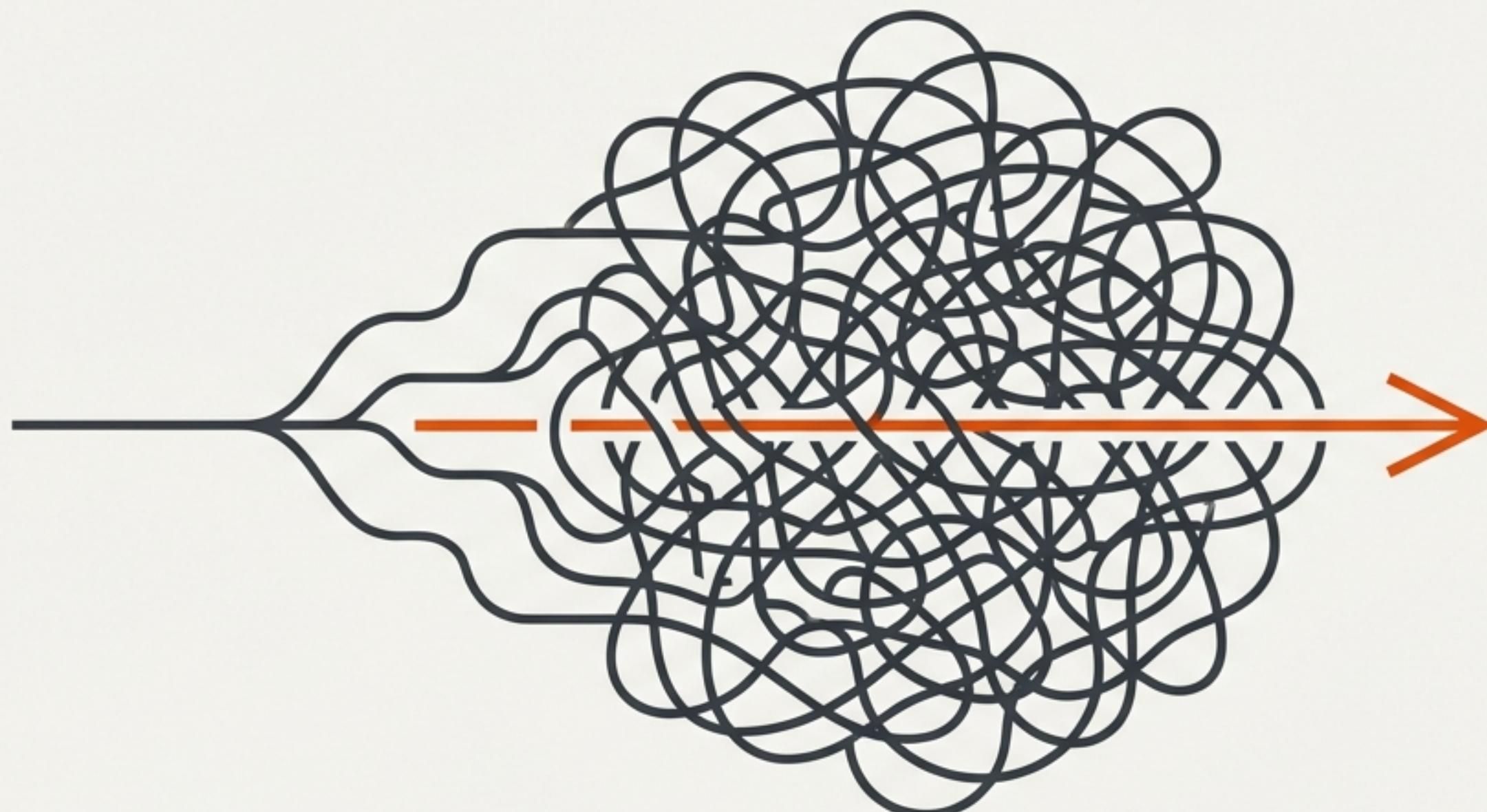


# ChipChat: Generating Verilog with Generative AI

- Objective: Evaluate the feasibility of generating synthesizable Register-Transfer Level (RTL) code from natural language descriptions.
- Core Task: Translate high-level functional requirements for digital circuits directly into Verilog source code using a Large Language Model (LLM).
- Experimental Scope: An end-to-end workflow from crafting a descriptive prompt to performing simulation-based verification of the generated hardware module.
- Key Technologies Employed: Python for scripting, the OpenAI API for LLM access, and Icarus Verilog for compiling and simulating the output.



# The Challenge: Accelerating the RTL Design Cycle



- **The Problem:** Traditional RTL design is a manual, expertise-intensive process that can be a significant bottleneck in chip development.
- **Complexity Barrier:** As system-on-chip (SoC) designs grow, the time spent writing, debugging, and verifying Verilog or VHDL code increases exponentially.
- **The Opportunity for Automation:** Can we abstract the intricate syntax of Hardware Description Languages (HDLs) behind a more intuitive natural language interface?
- **Central Research Question:** Is today's Generative AI capable of acting as a reliable co-pilot for hardware engineers, speeding up the creation of initial design drafts?

# The ChipChat Lab Workflow



## 1. Prompt Engineering

A precise natural language prompt is crafted, defining the hardware module's intended function, inputs, and outputs.

## 2. AI-Powered Generation

The prompt is submitted to an LLM (e.g., gpt-4o-mini) via an API call, which returns the corresponding Verilog source code.

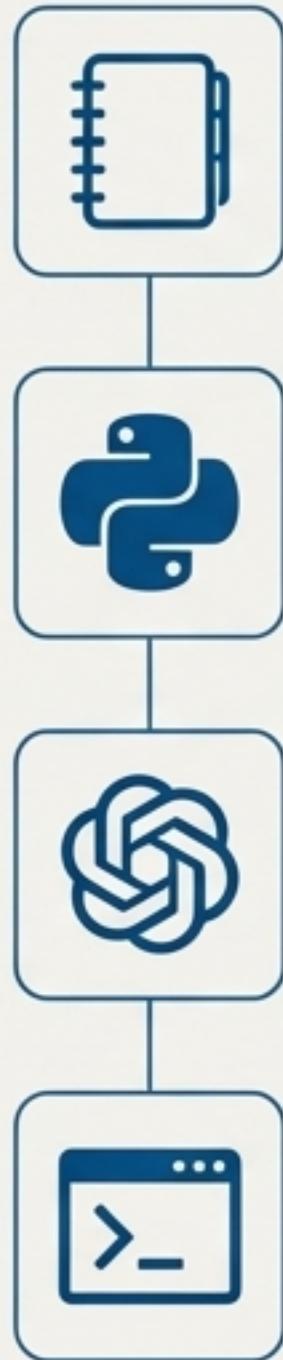
## 3. Simulation & Verification

The AI-generated Verilog is compiled with a pre-written testbench using the Icarus Verilog simulator to check for correctness.

## 4. Analysis & Iteration

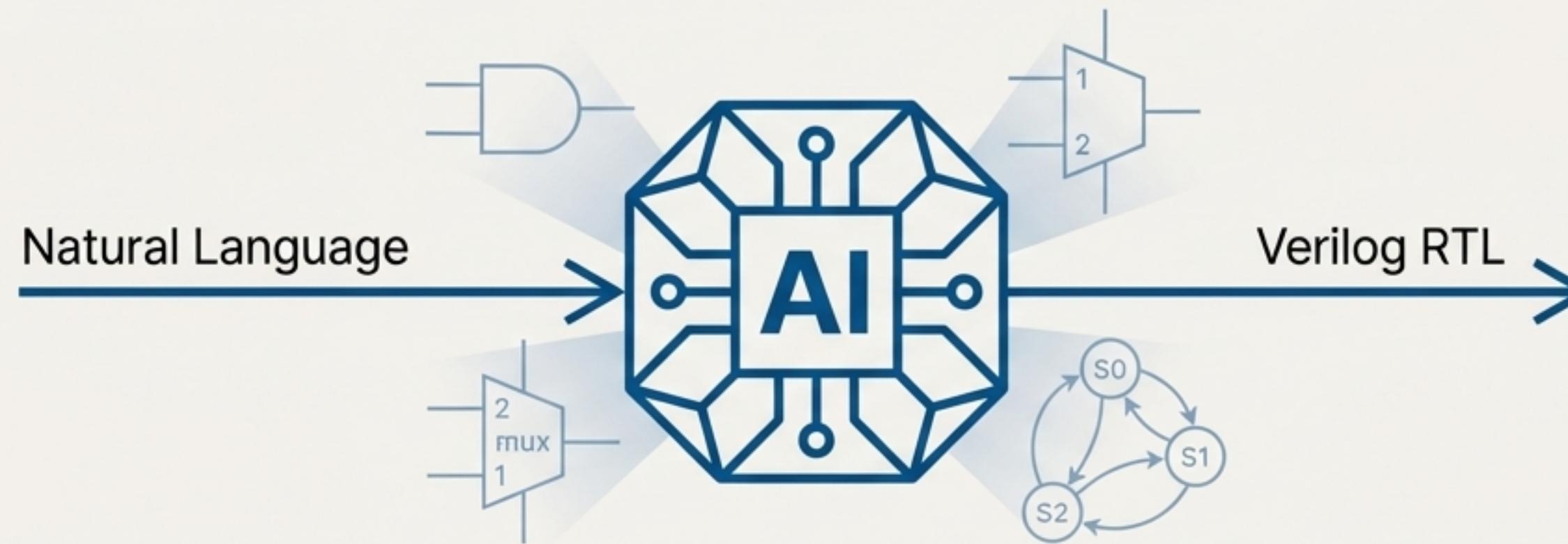
The simulation output is analyzed. If failures occur, the process is repeated by refining the prompt or manually debugging the code.

# Lab Methodology and Toolchain



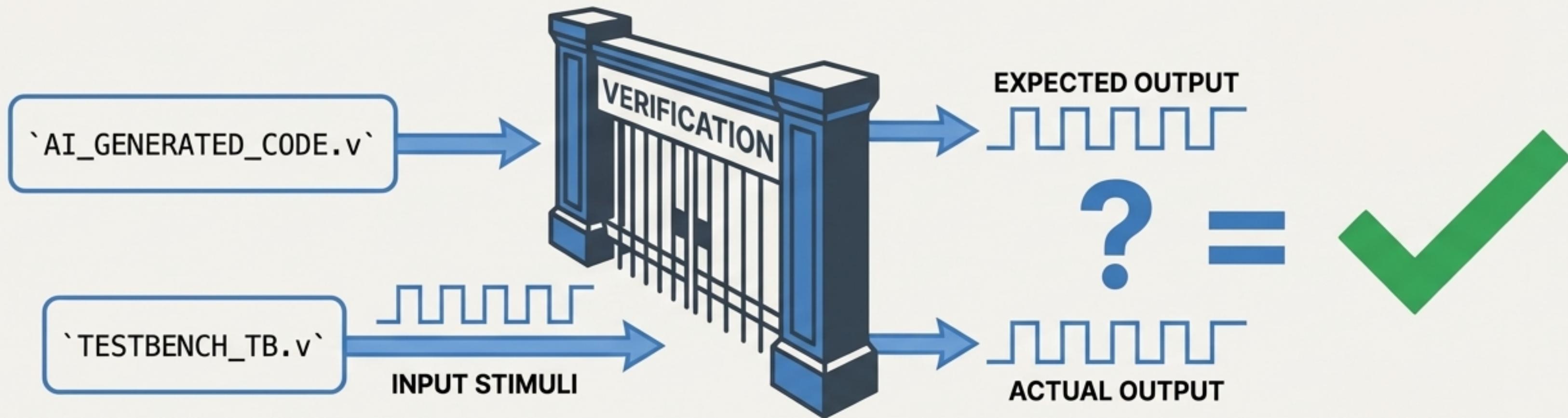
- **Execution Environment:** All experiments are conducted within a **Google Colab** notebook to ensure a consistent and reproducible setup.
- **GenAI Interface:** The official `openai` **Python library** serves as the bridge to the LLM API endpoints.
- **Verification Suite:** **Icarus Verilog** (`iverilog`) and its runtime engine (`vvp`) are used for command-line compilation and simulation of the generated Verilog against its testbench.
- **Automation Scripting:** The entire process is managed by **Python scripts** that handle API requests, parse LLM responses, manage file I/O, and execute simulation commands.

# The Specific Role of Generative AI



- **Core Function:** Act as a translator, converting abstract functional descriptions from natural language into syntactically valid Verilog RTL.
- **Primary Use Case:** Rapidly generate a 'first draft' of a hardware module, which can then be handed off to an engineer for verification and refinement.
- **Design Space Exploration:** The lab tests the LLM's ability across a range of digital circuits, from a simple `binary\_to\_bcd\_converter` to more complex Finite-State Machines (FSMs).
- **Key Variable:** The quality and correctness of the output is shown to be highly dependent on the specificity and clarity of the input prompt.

# A Strategy for Verification and Validation



- **Guiding Principle:** All AI-generated code is considered untrusted and must pass rigorous, independent verification before being accepted as correct.
- **Simulation-Based Testing:** Each generated module is compiled and run with a corresponding, manually-written Verilog testbench (e.g., `module\_name\_tb.v`).
- **Testbench Responsibility:** The testbenches provide a deterministic set of input stimuli and check for the exact expected output behavior over a series of clock cycles.
- **Success Criteria:** A design is only considered successful if it compiles without error and the testbench simulation completes with zero reported failures.

# Key Results and Direct Observations

## Perfect Generation



`binary\_to\_bcd\_converter`

## Logical Failure



"Cycle 8, Expected: 1, Got: 0"

## Syntax & Elaboration Errors



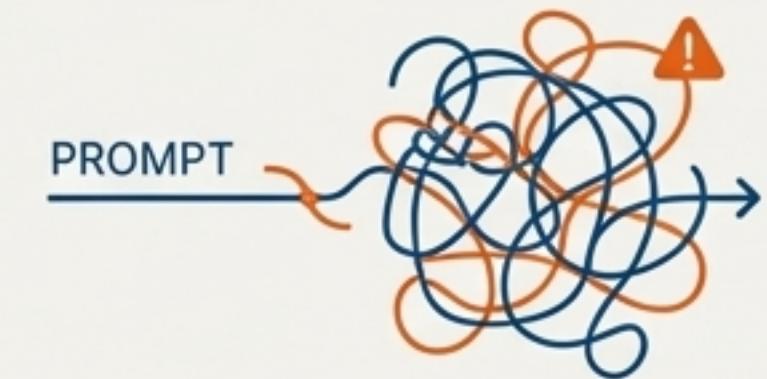
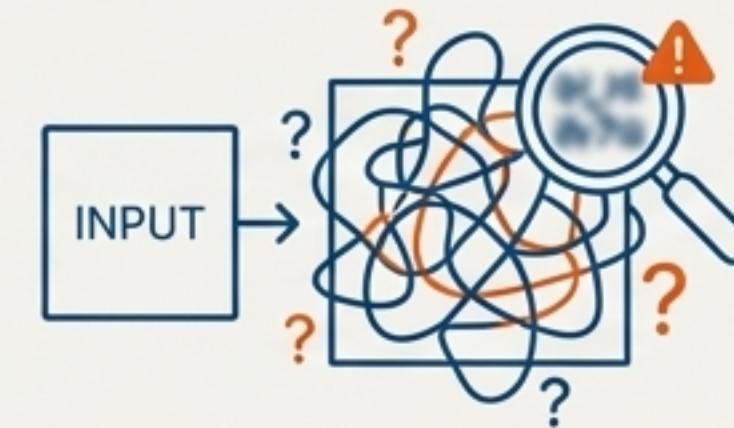
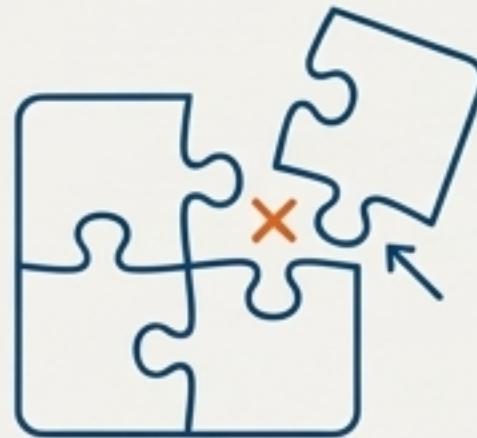
"Unknown module type"

## Iterative Success



FSM '00011000' after prompt refinement

# Limitations and Challenges Encountered



- Inconsistent Reliability: The quality of the generated Verilog varied significantly, from production-ready to completely non-functional, with no guarantee of success.
- Semantic Gaps: The LLM can produce code that is syntactically correct but logically flawed, failing to fully capture the design's intended behavior.
- The Debugging Problem: When the AI's logic is incorrect or convoluted, debugging the generated code can be as time-consuming as writing it from scratch.
- Prompt Brittleness: The lab demonstrated that minor ambiguities or omissions in the natural language prompt can cascade into major functional errors in the Verilog output.

# Practical Lessons Learned from the Lab



- **A Human-in-the-Loop is Essential:** GenAI serves best as an augmentation tool, not an autonomous replacement for engineering expertise. Verification and debug remain critical human tasks.



- **The Power of Iterative Refinement:** The most complex and successful designs were achieved not on the first try, but through a cycle of defining, prompting, verifying, and refining.



- **Specificity Is the Key to Success:** Vague prompts lead to flawed results. Highly detailed prompts that explicitly define states, transitions, and specifications produce far better outcomes.



- **Verification Is the Ultimate Safety Net:** A robust, independent verification framework is non-negotiable when working with AI-generated hardware designs.

# Conclusion: A Promising but Immature Accelerator

- **Proof of Concept Validated:** The ChipChat lab successfully demonstrates that LLMs can automate the generation of functional Verilog code from natural language prompts.
- **Potential as a Design Accelerator:** This workflow shows clear potential to significantly reduce the time spent on writing initial RTL code, allowing engineers to focus more on architecture and verification.
- **Current State:** The technology in its current form lacks the consistency and reliability required for fully automated workflows, mandating a strong verification overlay.
- **Path Forward:** Future progress will likely depend on fine-tuning models on large, curated datasets of verified hardware designs and integrating these tools more deeply into standard EDA flows.

