From Generation to Verified Synthesis

Bridging Industrial Reality via C-Guided Agents

Min Li

January 27, 2026

School of Integrated Circuit, Southeast University

Contributor：Kezhi Li （The Chinese University of Hong Kong)

Table of content

1 Intro & Motivation

2 Methodology: The Core

3 Benchmark & Experiments

4 Conclusion

Background: The Rise of LLM-Assisted Design

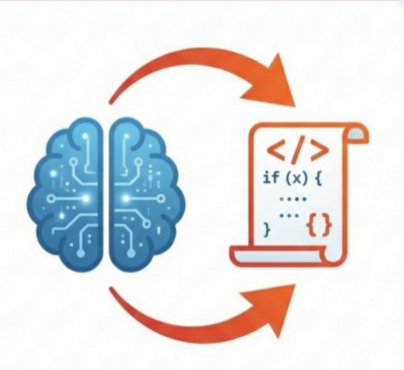
The Challenge

AI Chip Complexity ⬆️
(GPUs, NPUs)

Pain Point:
Engineering bottleneck & prolonged design cycles.

The Opportunity

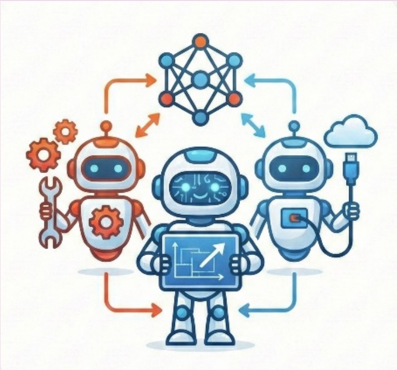
LLMs show potential in code generation.



Natural Language Understanding
⬇️
Hardware Synthesis

Current Approach

Methods: *From* <u>Supervised Fine-tuning</u>

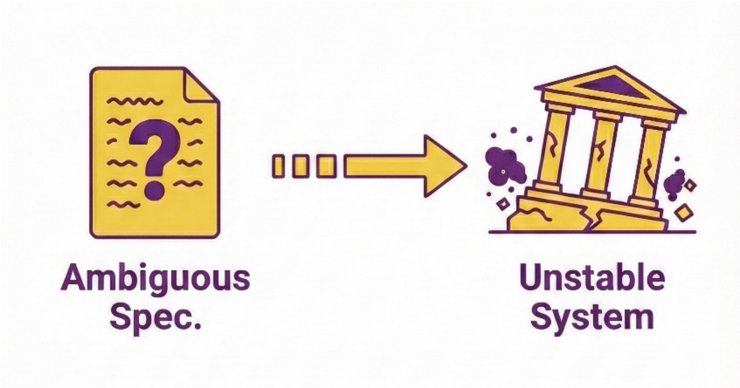


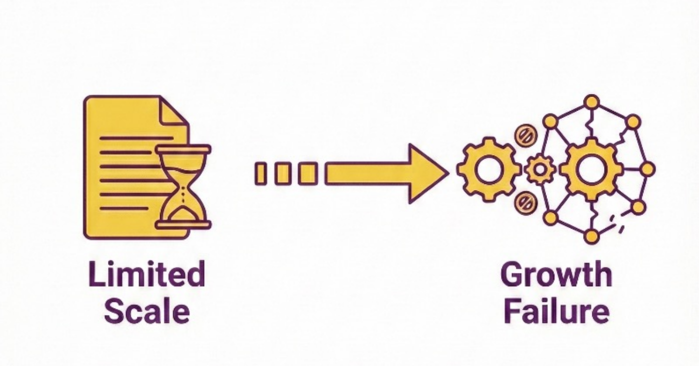
To <u>Multi-Agent System</u>.
While promising, still in the prototype phase.

Current Limitation: The Gap to Industrial Reality

1. Ambiguous Specification

- Natural language lacks precision for complex logic.
- *Result*: Unstable architectural decisions.



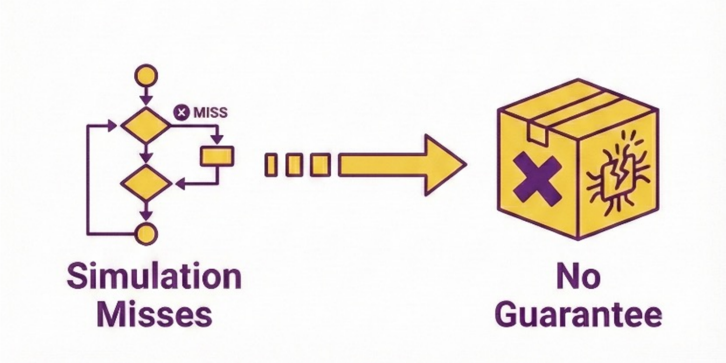


2. Limited Scalability

- Natural language lacks precision for complex logic.
- *Result*: Unstable architectural decisions.

3. No Formal Guarantee

- Simulation misses corner cases.
- *Result*: Cannot ensure functional correctness.



Current Limitation: The Gap to Industrial Reality

	Academia (LLM-only)	Industry Requirements
Input	Natural Language	C Reference Model
Scale	< 100 Lines	> 1000 Lines
Verification	Simulation Only ❌	Formal Equivalence ✅

Direct synthesis from natural language cannot meet industrial standards for correctness and scale.

Table of content

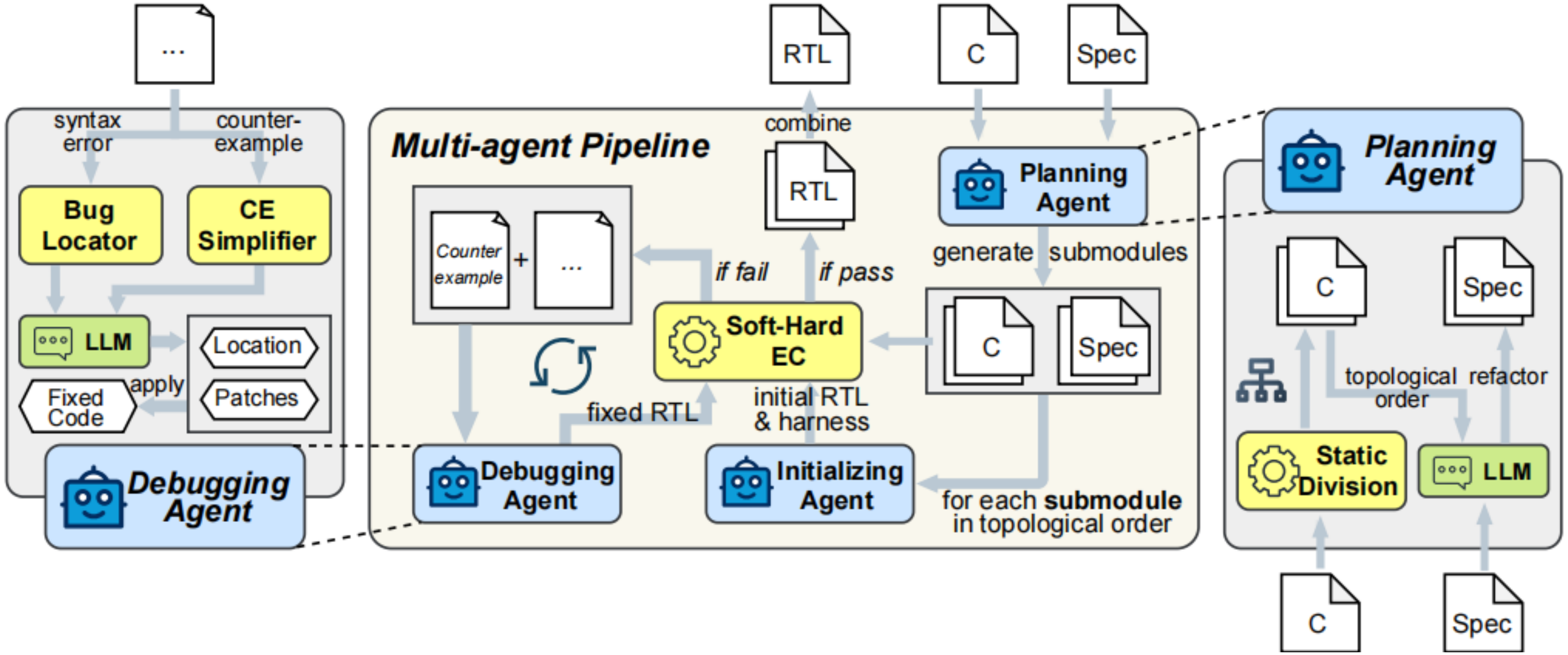
1 Intro & Motivation

2 Methodology: The Core

3 Benchmark & Experiments

4 Conclusion

Methodology Overview



C-Guided Planning → **RTL Generation** → **Verifiable Debugging Loop**

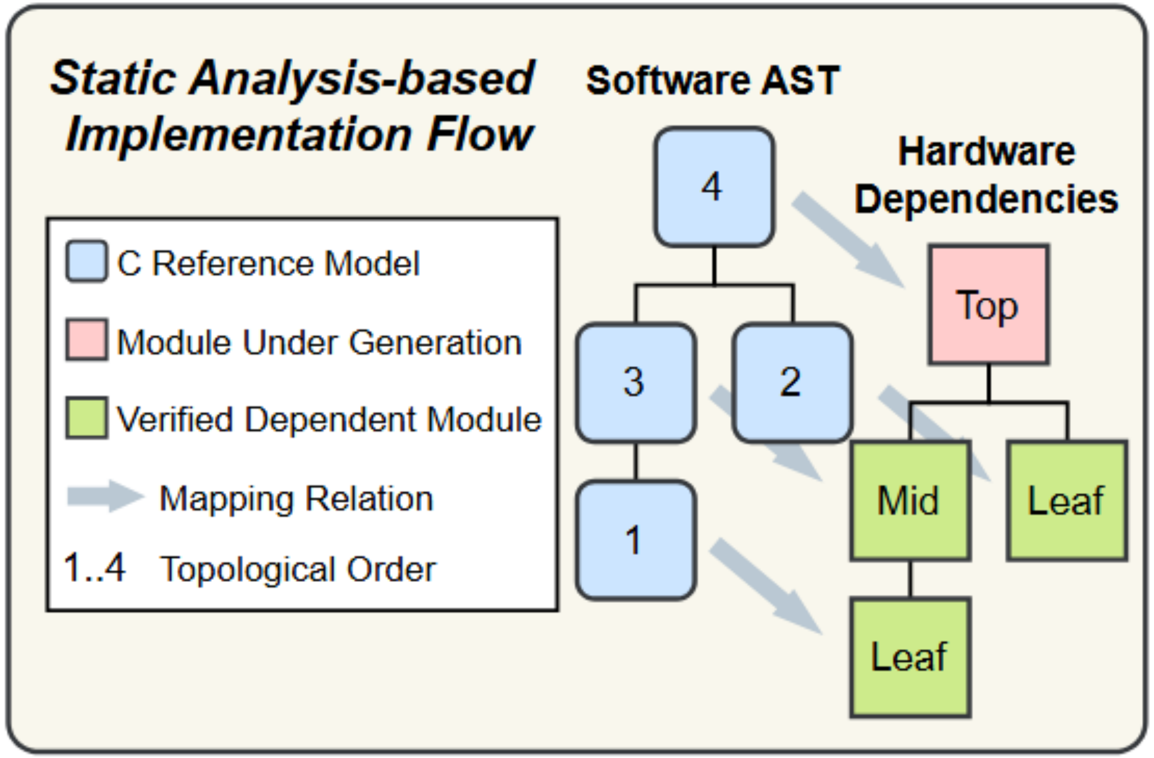
Integrating Static Analysis, Equivalence Checking, and LLM Agents in a unified pipeline.

The Planning Agent: Static Decomposition

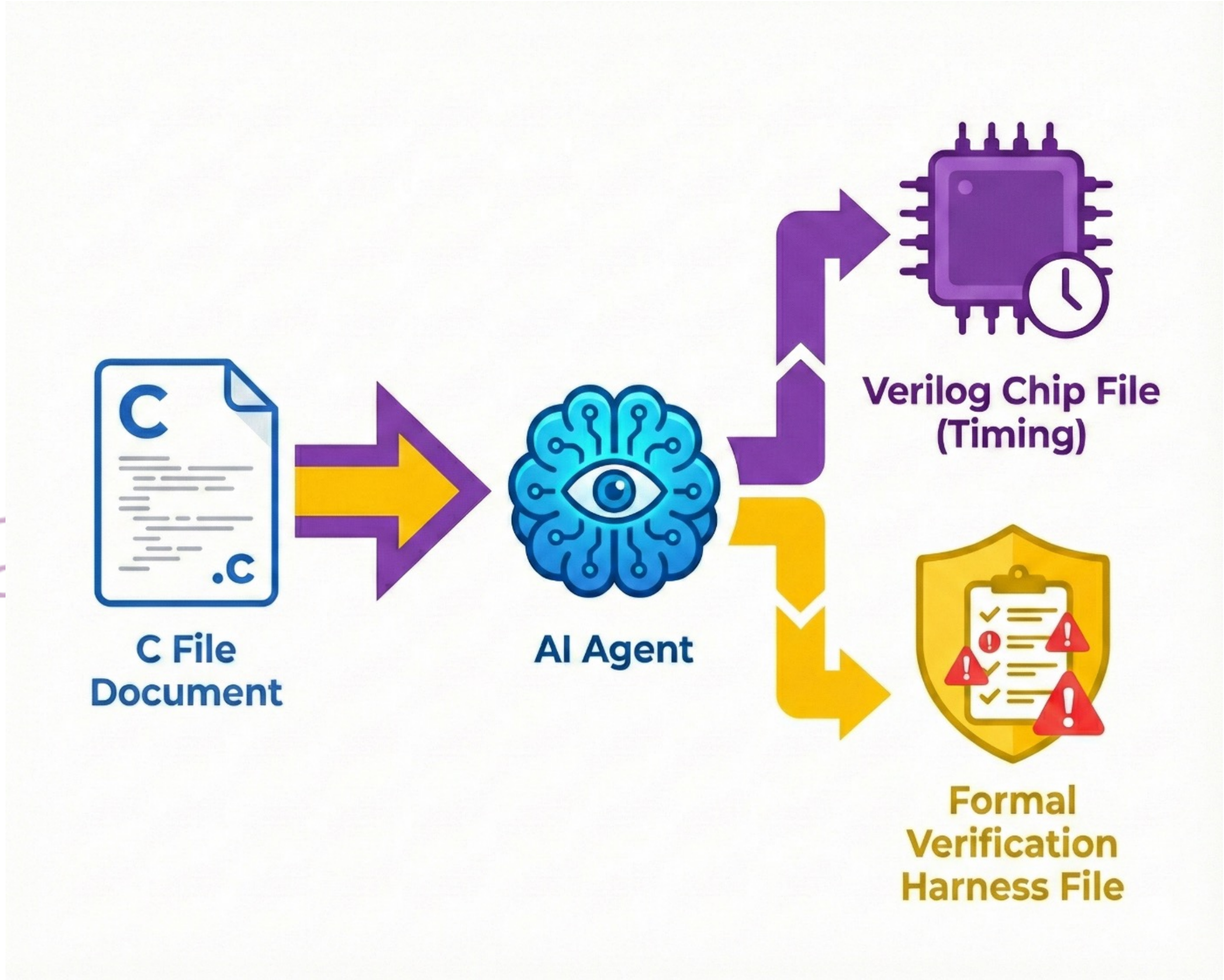
Driven by Static Analysis
Uses C Compiler (Clang) AST to analyze function dependencies.

Modular Partitioning
Breaks monolithic designs into manageable sub-tasks.

Bottom-Up Strategy
Verifies leaf modules first to simplify top-level integration.



The Initializing Agent: Setup & Synthesis



Dual Generation
Produces both the Initial RTL and the Verification Harness simultaneously.

Timeframe Alignment
Automatically determines pipeline depth (latency) for sequential logic.

Strict Anchoring
Harness asserts strict equivalence between C outputs and RTL outputs

The Debugging Agent: Formal-Guided Convergence

Beyond Pass/Fail
Uses formal tools (hw-cbmc) to generate precise feedback.

Bug Locator
Identifies the exact line of syntax errors or logic mismatches.

CE Simplifier
Extracts minimal Counter-Examples (input values that break the design) to guide the fix.

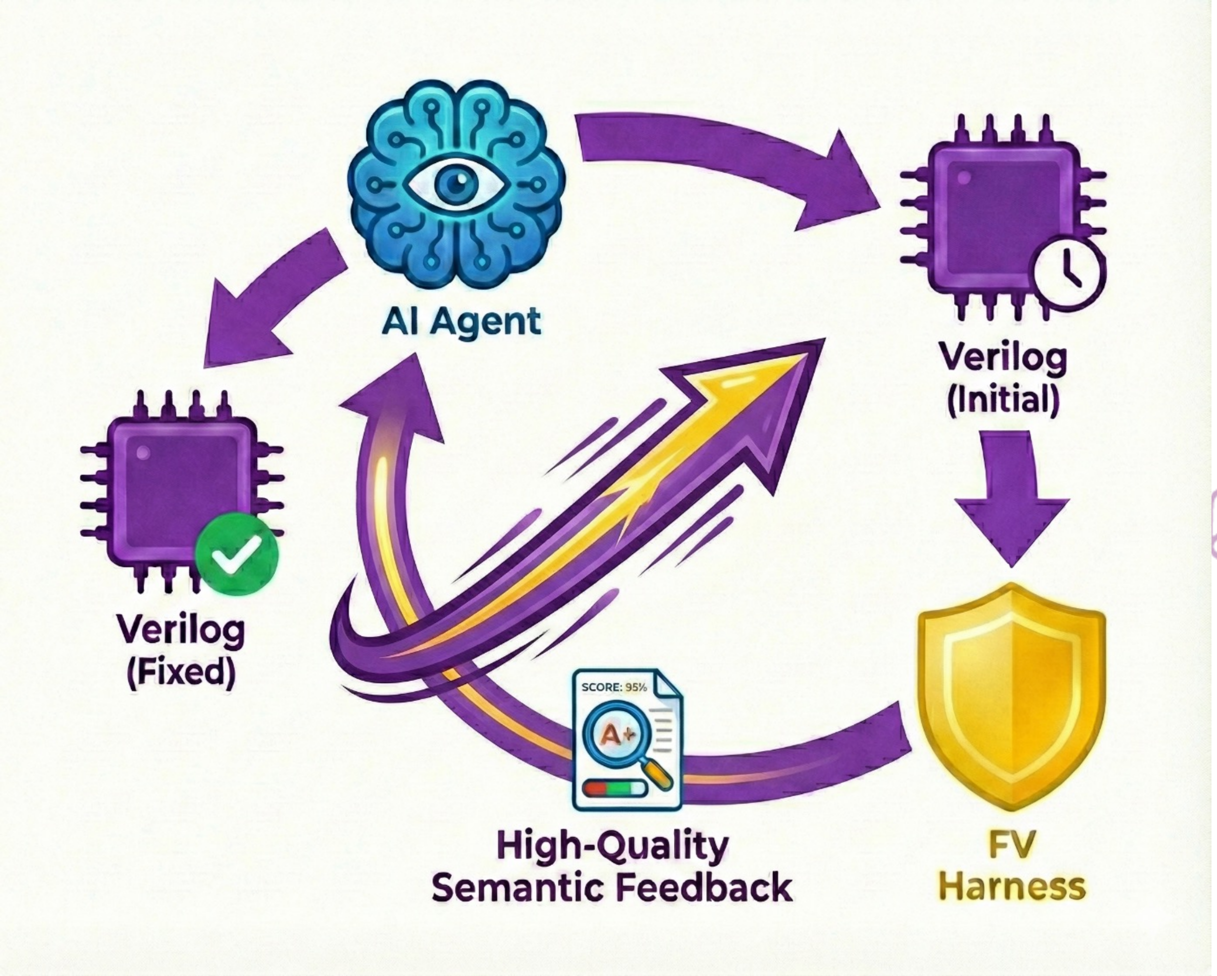


Table of content

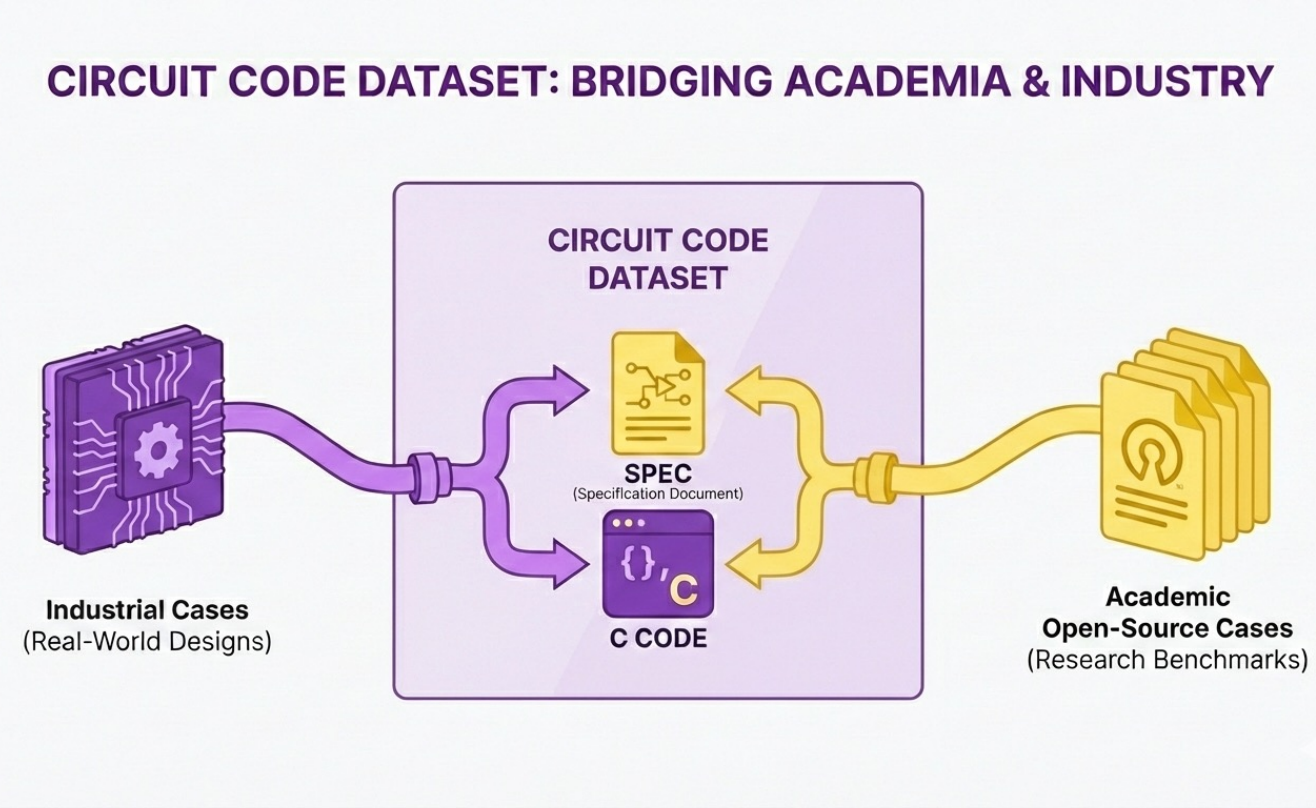
1 Intro & Motivation

2 Methodology: The Core

3 Benchmark & Experiments

4 Conclusion

Benchmark Curation



The Gap
Existing datasets lack industrial specifications and executable C reference models.

Our Contribution
A novel suite targeting <u>datapath-intensive</u> designs.

Completeness
Each case includes a detailed **NL Spec** and a **Golden C Reference Model**.

Main Results: Robustness Across Complexity

Selected Results

Design	Module Name	Type	Initial Success Rate (pass@1)	# of Iterations		Final Success Rate	Module / Total RTL Length (Average # of Lines)
				I_{avg}	I_{std}		
	f16_add	top	65%	1.00	1.84	100%	46.15 / 1129.95
	hifloat8_add (sequential)	top	20%	16.00	16.25	70%	420.64 / 981.99
	hifloat8_mul (sequential)	top	60%	9.65	15.67	80%	220.10 / 757.35

Broad Coverage: Verified across **15+ modules**, ranging from standard IEEE754 to custom HiFloat8.

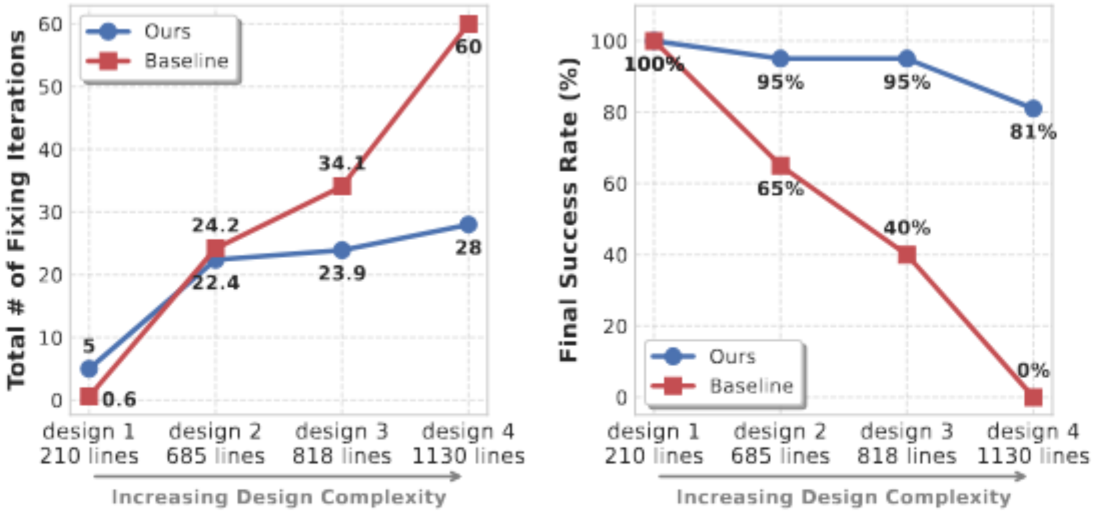
Scale Handling: Successfully generated designs exceeding **1000 lines** of RTL code.

The "Agent" Effect:

- Initial LLM generation (ISR) can be as low as **20%** for complex logic.
- It achieves **>70% Final Success Rate (FSR)** on most modules through iterative fixing.

Ablation Studies: Why It Works

Necessity of Reference-Guided Planning



Baseline (No Static Analysis): Fails completely on large designs (FSR ➡️ 0%).
Ours: Maintains stability via modular decomposition.

Efficiency of Formal Debugging Tools

Methods	Average # of Iterations	FSR
w/o bug locator	14.15	80%
w/o CE simplifier	9.55	90%
w/o CE-guided debugging	21.30	55%
FormalRTL	**6.40**	**95%**

w/o Bug Locator: Success Rate ⬇️ (Hard to locate errors in the code).
w/o CE Simplifier: Need more efforts and tokens on the CE understanding.
w/o CE: Fails greatly (Blind guessing).

The Gap to Manual Design

	hifloat8_mul		f16_mul	
	Area (μm^2)	Delay (ns)	Area (μm^2)	Delay (ns)
FormalRTL	1015	3.29	1629	3.54
Engineer	743	1.78	1343	2.50

Observation:
LLM-generated RTL trails expert human designs in Area and Delay.

The "Why":
Our current priority is guaranteeing functional equivalence (passing formal checks) rather than aggressive logic optimization.

The Value:
Verified Baseline: It provides a correct, executable starting point.
Agile Iteration: It is easier for engineers to optimize correct code than to fix broken logic.

Table of content

1 Intro & Motivation

2 Methodology: The Core

3 Benchmark & Experiments

4 Conclusion

Conclusion

Summary

Reference-Driven Paradigm
Shifted from ambiguous natural language to C-Guided Synthesis.

End-to-End Verification
The first pipeline to integrate Formal Equivalence Checking into the generation loop.

Industrial Validity
Curated a benchmark suite (including HiFloat8) to prove robustness on complex datapaths.

Future Work

Specialized Debugging Models
Training smaller, task-specific models to reduce reliance on large commercial LLMs.

Open-Source Tooling
Developing more robust open-source Equivalence Checking tools for modern RTL.

Full-System Scale
Extending the decomposition strategy to handle full industrial-scale system designs.

Thank You