VeriMoA: A Mixture-of-Agents Framework for Spec-to-HDL Generation

Heng Ping[1], Arijit Bhattacharjee[2], Peiyu Zhang[1], Shixuan Li[1], Wei Yang[1], Anzhe Cheng[1], Xiaole Zhang[1], Jesse Thomason[1], Ali Jannesari[2], Nesreen Ahmed[3], Paul Bogdan[1]

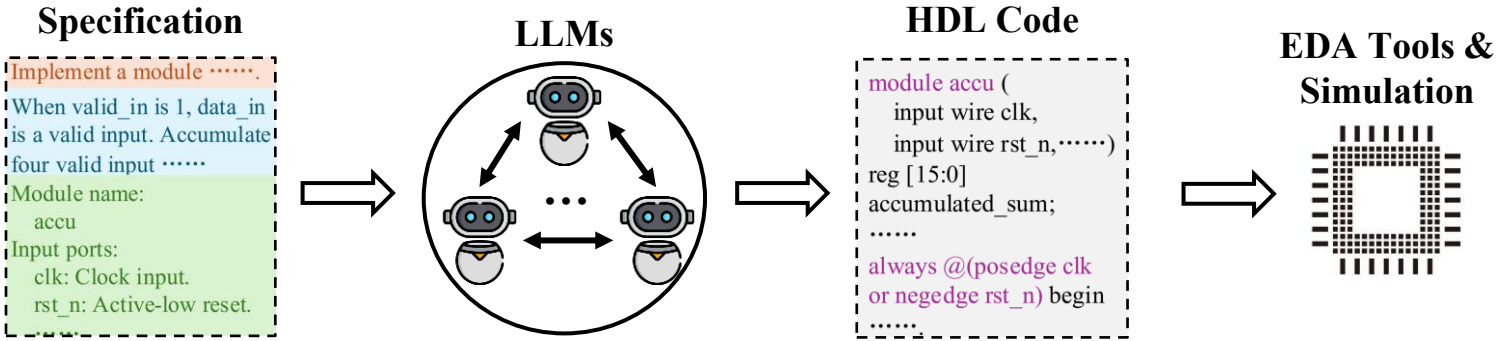
[1]University of Southern California [2]Iowa State University [3]Cisco AI Research

Proceedings of the 9 th MLSys Conference, Bellevue, WA, USA, 2026

Background & Motivation

What is the task?

- Automatically generate HDL (Hardware Design Language) code from natural-language specifications.



Why HDL is hard for LLMs?

Sparse Pretraining Data
HDL << C++/Python in LLM pretraining corpora

Concurrent/Timing Semantics
Not sequential like common programming language

Synthesis Constraints
HDL must be synthesizable, not just functionally correct

- **General-purpose LLMs lack enough domain-specific knowledge to generate correct HDL.**

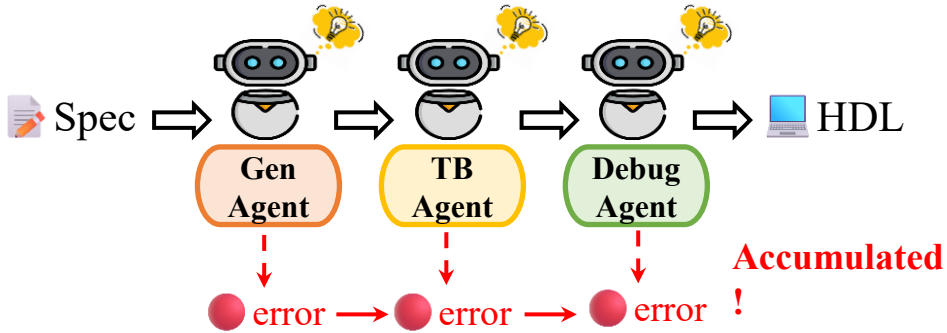
Existing Approaches & Missing Gap

Table: Comparison of Existing RTL Generation Approaches

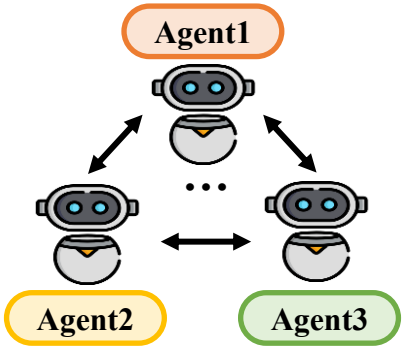
Approach	Representative Works	Limitation
🔧 **Prompt Engineering**	ParaHDL, AoT, HDLCoRe	❌ Bounded by LLM's sparse HDL knowledge
🎯 **SFT / RL**	RTLCoder, VeriRL, CodeV	❌ Costly training & data curation
🤝 **Multi-Agent Systems**	MAGE, CoopetitiveV, VeriMaAS	⚠️ Error propagation or chaotic exploration

Existing Multi-Agent Approaches

➢ Sequential Pipeline



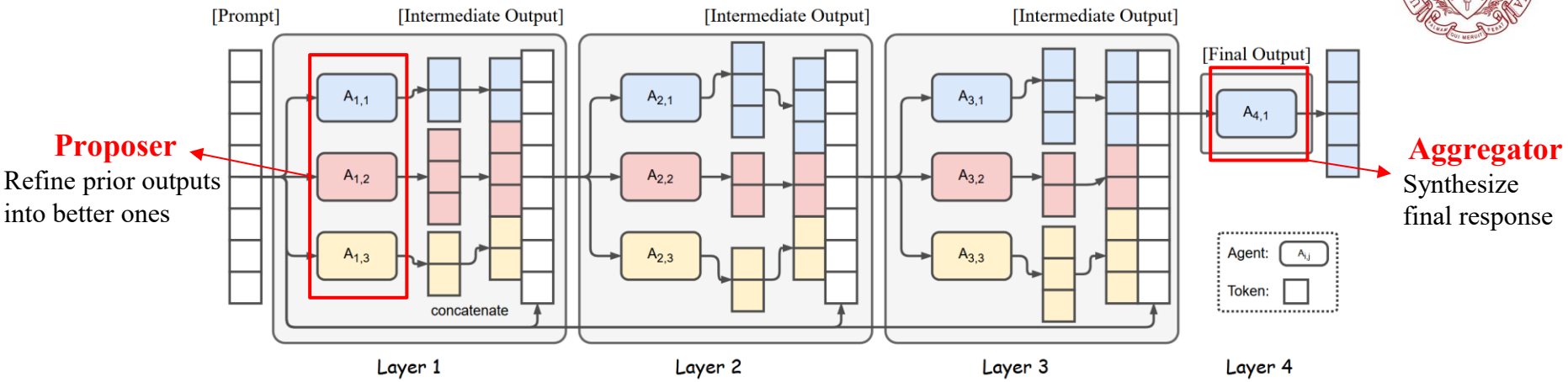
➢ Debate Pipeline



1. Stronger agent dominates
2. No convergence guarantee

Mixture-of-Agents (MoA)

MoA Structure —— A Layer-wise Paradigm



Why is MoA promising for HDL?

✅ **Layered + parallel** — structured information flow

✅ **Superior** on instruction-following & reasoning benchmarks

🌟 **Training-free** — works with any LLM backbone

What does standard MoA still lack?

❌ **Cascaded dependency**: Each layer only sees the immediately preceding layer

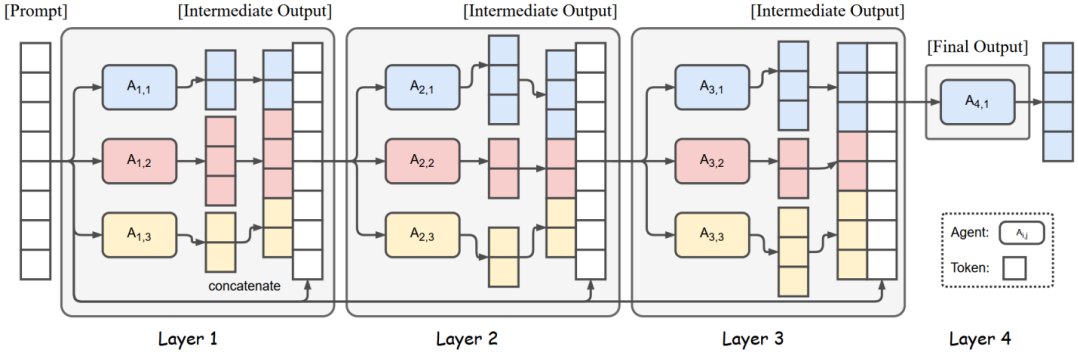
❌ **Information loss**: Valuable solutions from earlier layers are forgotten

❌ **Limited reasoning path** — all proposers reason similarly, limiting solution diversity

Key Insight — What Drives MoA Performance?

Principle of MoA

- MoA performance is highly influenced by the quality and diversity of the agents in MoA layers.[1]



$$\text{Permance} \approx \alpha \cdot \textbf{Quality} + \beta \cdot \textbf{Diversity} + \gamma$$

α > β (quality matters more than diversity, but both are essential)

How to improve MoA for HDL?

How to improve quality?

⭐ **Quality-Guided Caching**
Cache and rank all intermediate outputs across layers.

How to increase diversity?

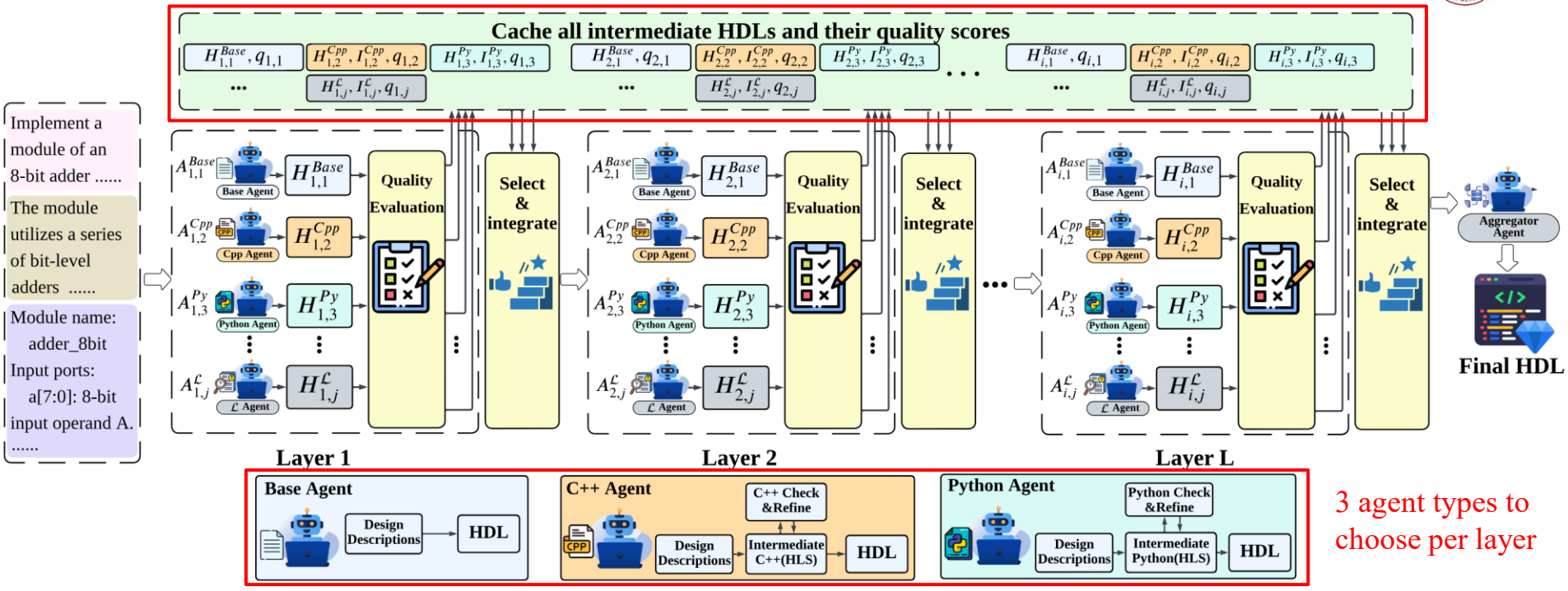
🌐 **Multi-Path Generation**
Generate via C++ and Python intermediate representations

[1] *Ref: Rethinking Mixture-of-Agents: Is Mixing Different Large Language Models Beneficial? (2025)*

Framework Overview

Our Proposed VeriMoA

Global cache indexes ALL intermediates by quality



3 agent types to choose per layer

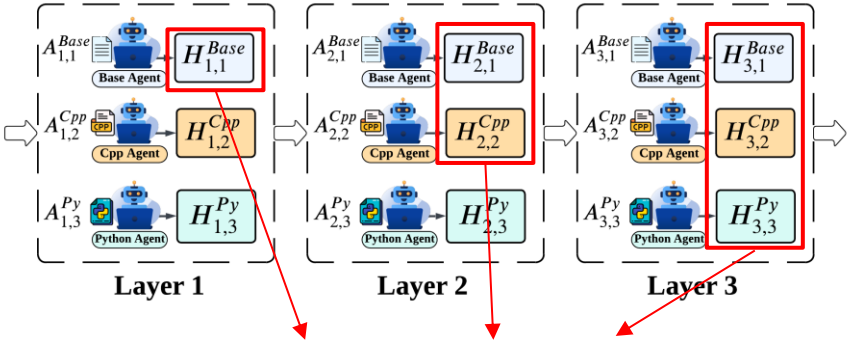
- Three agent types (Base, C++, Python) generate HDL in parallel across layers
- Intermediate outputs are scored, cached globally, and the top-N are passed to deeper layers

Innovation 1 — Quality-Guided Caching

Quality-Guided Caching Mechanism

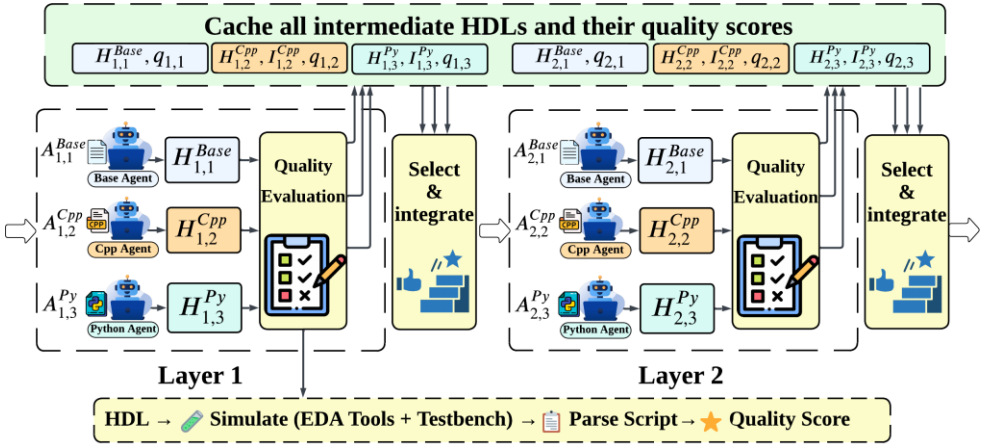
- Cache all the intermediate HDLs and evaluate the quality of each HDL
- Select top HDLs by quality from cache and integrate them into the prompt for the following agents

Standard MoA



⚠️ **Limitations**
- Errors cascade across layers
- Good solutions get lost
- Deeper agents suffer more

Quality-Guided Cached MoA



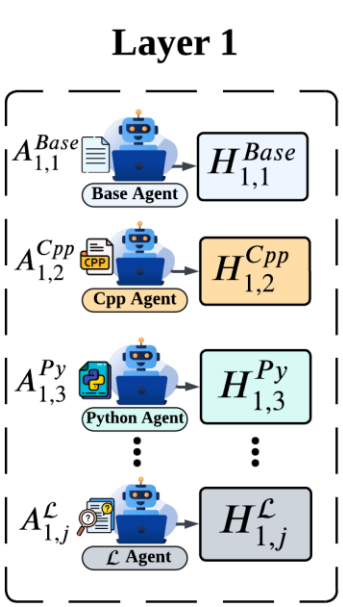
✅ **Benefits**
- Errors filtered at every layer
- Top-quality solutions preserved
- Deeper agents build on the best

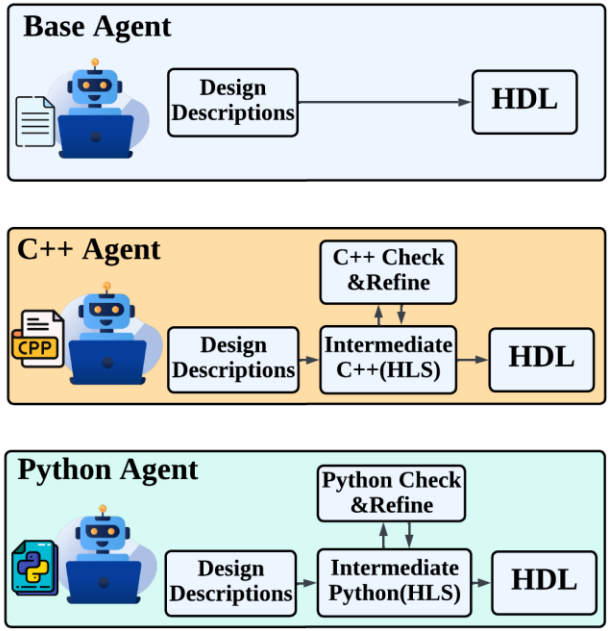
➢ **Global caching + quality scoring → monotonic quality improvement across layers.**

Innovation 2 — Multi-Path Generation

Multi-path Two-stage Generation

- A multi-path, two-stage approach uses C++/Python as intermediate representations to exploit LLM strengths in high-resource languages while promoting solution diversity





Base Agent
- Hardware-native thinking
- Captures HDL conventions

C++ Agent
- Imperative, bit-level reasoning
- Maps cleanly to HDL behavior

Python Agent
- Abstract, algorithmic reasoning
- Captures high-level intent

- **Why this works:** LLMs are far more fluent in C++/Python → leverage that strength
- **Diversity guarantee:** Heterogeneous reasoning trajectories → broader solution space exploration

Experimental Setup

Benchmarks & Baselines

Benchmarks

- **VerilogEval 2.0:** 156 problems
- **RTLLM 2.0:** 50 complex designs

Baselines

- **Prompting LLMs**
- **SFT / RL-based Methods**
- **Multi-agent LLMs**

LLM Backbones

LLM Models

- **Open-Source:**
  - Qwen2.5 (7B/14B/32B)
  - Qwen2.5-Coder (7B/14B/32B)
- **Closed-Source:**
  - GPT-4o-mini
  - GPT-4o
- **Open-Source Domain-Specific:**
  - VeriRL-CodeQwen2.5

Configuration

Para. & Testing

- L = **4 layers**
- M = **6 agents/layer** (2 Base + 2 C++ + 2 Python)
- **Simulator**: Icarus Verilog
- **Metrics**: Pass@1, Pass@3, Pass@5

Research Questions

RQ1: Main performance?
RQ2: Contribution of core components?

RQ3: Inference cost comparison?
RQ4: Parameter sensitivity?
RQ5: Robustness without manual testbenchs?

Main Result — vs Non-Training Baselines (RQ1)

Comparison with Agentic Baselines

Model	Method	VerilogEval 2.0 (%)			RTLLM 2.0 (%)		
		Pass@1	*Pass@3*	*Pass@5*	*Pass@1*	*Pass@3*	*Pass@5*
Qwen2.5-7B	Direct	22.90	32.20	40.39	18.99	30.53	36.51
	CoT	26.87 ↑3.97	35.33 ↑3.13	42.58 ↑2.19	27.49 ↑8.50	34.76 ↑4.23	39.95 ↑3.44
	HDLCoRe	30.92 ↑8.02	37.86 ↑5.66	46.49 ↑6.10	34.68 ↑15.69	40.32 ↑9.79	44.82 ↑8.31
	VeriMaAS	32.81 ↑9.91	40.25 ↑8.05	49.07 ↑8.68	40.33 ↑21.34	47.24 ↑16.71	50.47 ↑13.96
	VERIMOA	**56.44 ↑33.54**	**62.07 ↑29.87**	**65.27 ↑24.88**	**52.07 ↑33.08**	**58.10 ↑27.57**	**60.46 ↑23.95**
Qwen2.5-Coder-7B	Direct	33.91	43.90	48.06	26.16	36.94	41.11
	CoT	36.74 ↑2.83	45.88 ↑1.98	50.99 ↑2.93	31.87 ↑5.71	39.36 ↑2.42	45.36 ↑4.25
	HDLCoRe	40.67 ↑6.76	47.81 ↑3.91	52.64 ↑4.58	36.26 ↑10.10	45.61 ↑8.67	51.12 ↑10.01
	VeriMaAS	44.73 ↑10.82	51.62 ↑7.72	55.78 ↑7.72	44.71 ↑18.55	51.37 ↑14.43	56.46 ↑15.35
	VERIMOA	**60.96 ↑27.05**	**68.13 ↑24.23**	**70.69 ↑22.63**	**54.43 ↑28.27**	**62.52 ↑25.58**	**66.24 ↑25.13**
Qwen2.5-14B	Direct	39.26	49.98	53.74	31.71	40.20	45.87
	CoT	41.24 ↑1.98	51.62 ↑1.64	54.37 ↑0.63	36.49 ↑4.78	44.64 ↑4.44	48.29 ↑2.42
	HDLCoRe	45.18 ↑5.92	55.83 ↑5.85	59.42 ↑5.68	41.78 ↑10.07	49.97 ↑9.77	53.17 ↑7.30
	VeriMaAS	48.97 ↑9.71	57.84 ↑7.86	61.77 ↑8.03	48.14 ↑16.43	54.28 ↑14.08	58.76 ↑12.89
	VERIMOA	**65.27 ↑26.01**	**71.61 ↑21.63**	**74.64 ↑20.90**	**55.06 ↑23.35**	**62.76 ↑22.36**	**66.33 ↑20.46**
Qwen2.5-Coder-14B	Direct	39.74	49.72	52.21	35.61	42.34	46.43
	CoT	43.81 ↑4.07	51.27 ↑1.55	54.11 ↑1.90	40.65 ↑5.04	48.77 ↑6.43	51.72 ↑5.29
	HDLCoRe	46.82 ↑7.08	54.79 ↑5.07	57.04 ↑4.83	47.86 ↑12.25	55.19 ↑12.85	58.18 ↑11.75
	VeriMaAS	50.96 ↑11.22	58.48 ↑8.76	62.63 ↑10.42	51.20 ↑15.59	58.93 ↑16.59	61.67 ↑15.24
	VERIMOA	**66.86 ↑27.12**	**73.24 ↑23.52**	**76.38 ↑24.17**	**59.76 ↑24.15**	**64.33 ↑21.99**	**67.22 ↑20.79**
Qwen2.5-32B	Direct	46.85	56.11	58.41	43.62	48.73	50.75
	CoT	48.72 ↑1.87	57.25 ↑1.14	59.80 ↑1.39	46.99 ↑3.37	50.50 ↑1.77	52.46 ↑1.71
	HDLCoRe	51.75 ↑4.90	59.63 ↑3.52	61.79 ↑3.38	50.95 ↑7.33	54.06 ↑5.33	58.98 ↑8.23
	VeriMaAS	53.57 ↑6.72	61.82 ↑5.71	64.26 ↑5.85	54.18 ↑10.56	57.21 ↑8.48	61.54 ↑10.79
	VERIMOA	**71.85 ↑25.00**	**76.48 ↑20.37**	**78.16 ↑19.75**	**63.81 ↑20.19**	**67.60 ↑18.87**	**69.88 ↑19.13**
Qwen2.5-Coder-32B	Direct	46.93	55.73	57.12	40.40	45.42	48.24
	CoT	48.65 ↑1.72	56.31 ↑0.58	59.34 ↑2.22	45.72 ↑5.32	48.75 ↑3.33	50.29 ↑2.05
	HDLCoRe	51.28 ↑4.35	58.63 ↑2.90	61.47 ↑4.35	51.72 ↑11.32	54.73 ↑9.31	59.64 ↑11.40
	VeriMaAS	56.67 ↑9.74	63.46 ↑7.73	66.92 ↑9.80	55.82 ↑15.42	59.97 ↑14.55	62.31 ↑14.07
	VERIMOA	**73.31 ↑26.38**	**79.05 ↑23.32**	**81.27 ↑24.15**	**65.49 ↑25.09**	**71.42 ↑26.00**	**74.11 ↑25.87**
GPT-4o-mini	Direct	48.97	56.94	58.62	46.60	50.71	51.95
	CoT	52.20 ↑3.23	59.83 ↑2.89	60.93 ↑2.31	48.27 ↑1.67	53.56 ↑2.85	55.48 ↑3.53
	HDLCoRe	53.50 ↑4.53	61.46 ↑4.52	63.72 ↑5.10	51.19 ↑4.59	56.72 ↑6.01	59.01 ↑7.06
	VeriMaAS	57.24 ↑8.27	64.77 ↑7.83	66.85 ↑8.23	57.25 ↑10.65	61.46 ↑10.75	63.17 ↑11.22
	VERIMOA	**72.43 ↑23.46**	**76.94 ↑20.00**	**79.46 ↑20.84**	**64.23 ↑17.63**	**67.45 ↑16.74**	**68.67 ↑16.72**
GPT-4o	Direct	64.74	69.89	71.66	52.48	56.18	57.62
	CoT	66.38 ↑1.64	71.18 ↑1.29	74.22 ↑2.56	54.89 ↑2.41	59.22 ↑3.04	61.77 ↑4.15
	HDLCoRe	69.60 ↑4.86	73.53 ↑3.64	75.86 ↑4.20	56.27 ↑3.79	61.47 ↑5.29	63.23 ↑5.61
	VeriMaAS	71.34 ↑6.60	76.12 ↑6.23	79.21 ↑7.55	61.70 ↑9.22	67.25 ↑11.07	68.84 ↑11.22
	VERIMOA	**84.97 ↑20.23**	**89.65 ↑19.76**	**91.03 ↑19.37**	**69.17 ↑16.69**	**73.68 ↑17.50**	**75.62 ↑18.00**

Key Findings

① **+15 to +33 Pass@1 points** improvement across all backbones

② **Smaller models beat larger baselines**: VeriMoA-Qwen7B (56.4%) > VeriMaAS-Qwen32B (53.6%)

③ **Bigger boost for smaller models:** Qwen-7B gains +33 pts vs GPT-4o gains +20 pts

Architectural design beats scale — every backbone improves, smaller ones improve more.

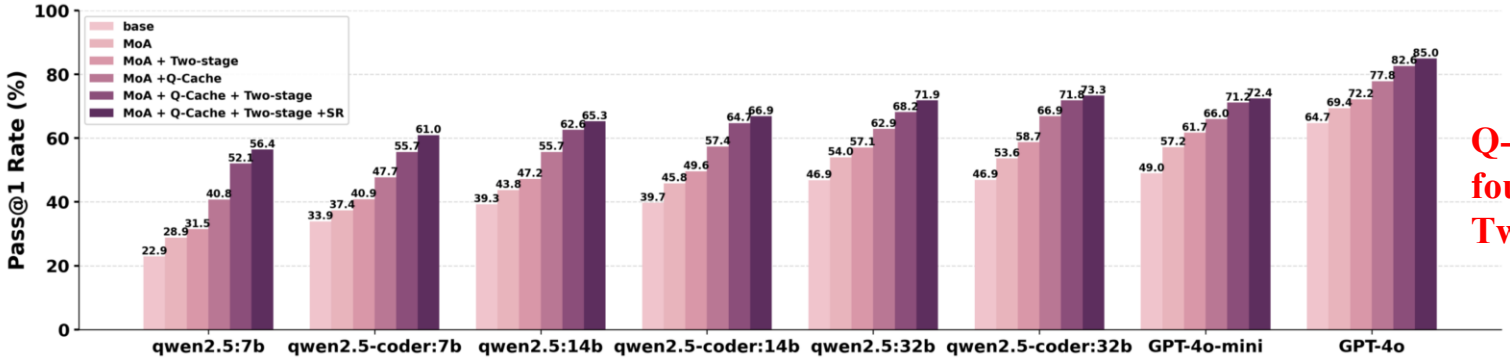
Main Result — vs SFT/RL Models + Ablation (RQ1 & RQ2)

Comparison with SFT/RL Baselines

	Model	Size	VerilogEval 2.0 (%)			RTLLM 2.0 (%)		
			Pass@1	*Pass@3*	*Pass@5*	*Pass@1*	*Pass@3*	*Pass@5*
Verilog-Specific (finetune)	RTLCoder-Mistral	7B	35.62	36.63	37.71	38.68	40.86	41.84
	RTLCoder-DeepSeek-Coder	6.7B	39.67	43.66	45.99	40.75	46.94	49.83
	OriGen-DeepSeek-Coder-7B-v1.5	7B	51.19	54.88	56.78	41.11	52.09	58.48
	HaVen-CodeQwen1.5	7B	55.40	59.73	62.76	51.04	57.38	60.89
	VeriRL-DeepSeek-Coder	6.7B	64.57	68.96	71.78	58.64	63.92	66.26
	VeriRL-CodeQwen2.5	7B	66.28	71.35	73.43	61.53	65.27	68.94
Ours	Qwen2.5-Coder-7B + VERIMOA	7B	60.96	68.13	70.69	54.43	62.52	66.24
	Qwen2.5-Coder-14B + VERIMOA	14B	66.86	73.24	76.38	59.76	64.33	66.22
	Qwen2.5-Coder-32B + VERIMOA	32B	73.31	79.05	81.27	65.49	71.42	74.11
	VeriRL-CodeQwen2.5 + VERIMOA	7B	82.47	87.91	90.68	74.45	78.63	82.02

➢ **Training-free framework matches or exceeds fine-tuned models AND is complementary to them.**

Contribution of Core Mechanisms



Q-Cache is the foundation unlocking Two-stage's potential.

Inference Cost Analysis (RQ3)

Inference Cost Comparison

Dataset	Model	Baseline		VeriMaAS		VERIMOA-Lite		VERIMOA	
		Pass@1	*Tokens*	*Pass@1*	*Tokens*	*Pass@1*	*Tokens*	*Pass@1*	*Tokens*
VerilogEval 2.0	Qwen2.5-Coder-32B	46.93%	0.63k	56.67%	3.67k (5.83×)	67.93%	3.73k (5.92×)	73.31%	6.95k (11.03×)
	GPT-4o-mini	48.97%	0.52k	57.24%	3.18k (6.12×)	68.41%	2.94k (5.65×)	72.43%	5.51k (10.60×)
RTLLM 2.0	Qwen2.5-Coder-32B	40.40%	0.76k	55.82%	4.59k (6.04×)	62.82%	4.05k (5.33×)	65.49%	7.28k (9.58×)
	GPT-4o-mini	46.60%	0.67k	57.25%	4.32k (6.45×)	61.29%	3.79k (5.66×)	64.23%	6.47k (9.66×)

⚡ **Cost:** ~10× baseline tokens → **Gain:** +20–25 Pass@1 points

🪶 **VeriMoA-Lite:** same cost as VeriMaAS, +11 points higher Pass@1

Accuracy Vs. Tokens——Scale up

➢ varying agents per layer using GPT-4o-mini

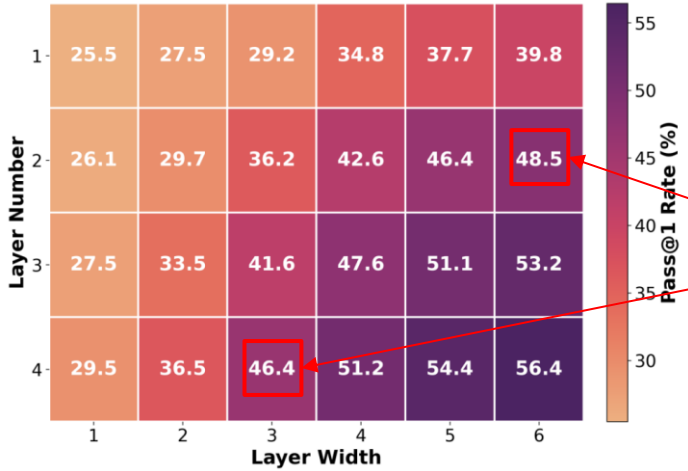
Agents	VerilogEval 2.0		RTLLM 2.0	
	Pass@1	*Tokens*	*Pass@1*	*Tokens*
2	64.52%	1.86k (3.57×)	57.89%	2.28k (3.40×)
3	68.41%	2.94k (5.65×)	61.29%	3.79k (5.66×)
4	70.28%	3.88k (7.46×)	62.76%	4.71k (7.03×)
5	71.49%	4.70k (9.04×)	63.58%	5.60k (8.36×)
6	72.43%	5.51k (10.60×)	64.23%	6.47k (9.66×)

📈 **Cost scales linearly, gains saturate:** tokens grow linearly, Pass@1 gains shrink from +3.9 to +0.9 pts

💡 **Diminishing returns** — fewer agents already capture most of the gain

Parameter Sensitivity & Robustness (RQ4 & RQ5)

Parameter Sensitivity——*Layer Depth × Layer Width (VerilogEval 2.0 using Qwen2.5:7b)*



📊 **Both depth & width matter —** 1-layer or 1-agent configs cap at <40% Pass@1

🎯 **Width > Depth at equal budget —** 2L×6W (48.5%) beats 4L×3W (46.4%)

Manual Testbench vs LLM-Generated Testbench (GPT-4o-mini)

Testbench	VerilogEval 2.0 (%)			RTLLM 2.0 (%)		
	P@1	*P@3*	*P@5*	*P@1*	*P@3*	*P@5*
Golden	72.43	76.94	79.46	64.23	67.45	68.67
LLM-Gen.	67.84	73.57	76.63	60.51	63.87	65.74
Δ	*-4.59*	*-3.37*	*-2.83*	*-3.72*	*-3.58*	*-2.93*

🛡 **Minor degradation: only** −2.8 to −4.6 Pass@1 pts without manual TB

✅ **Still beats VeriMaAS (**which uses manual TB**)**

Conclusion & Future Plan

Conclusion

🎯 **Quality-Guided Caching**: monotonic knowledge accumulation across layers

🌐 **Multi-Path Generation**: leverage LLMs' C++/Python fluency + structured diversity

📈 **15–30% Pass@1 gains** across diverse LLMs; small models match large; complementary to fine-tuning

Future Plan

🏭 **Hierarchical Generation**: Planner agent decomposes large designs (RISC-V cores, NoC routers) into modules for VeriMoA

🔄 **Cross-Domain Transfer**: Apply quality-guided multi-path MoA to other code generation tasks (CUDA, assembly, HLS)

Thanks for Your Listening !

Q&A