2025 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION CHINA

accellera SYSTEMS INITIATIVE

云豹智能 Micro

A Multi-Agent Generative AI Framework for IC Module-Level Verification Automation

Wenbo Liu, wenbo.liu@jaguarmicro.com

Forbes Hou, forbes.hou@jaguarmicro.com

Jon Zhang, jon.zhang@jaguarmicro.com

Hong Liu, hong.liu@jaguarmicro.com

Allen Lei, allen.lei@jaguarmicro.com

IC Design Department, Jaguar Microsystems

16 Apr 2025 上海 SHANGHAI

公司介绍

云豹智能——中国数据处理器芯片(DPU) 头部企业

460+员工，国内DPU头部企业及行业独角兽

- 由腾讯、深圳市引导基金、杭州头部资本、中芯聚源、深创投和其他顶级投资者投资

国产高性能DPU/SuperNIC芯片企业

- 高达400Gbps吞吐量，支持25G/50G/100G/200G 端口
- 国内性能最高的DPU网卡芯片，2024年Q4已量产

市场规模

- 国内需求量2026年达5百万片

主要投资人

腾讯 Tencent

深圳市引导基金

杭州头部资本

宝安区引导基金

SCGC 深创投

中芯聚源 China Fortune-Tech Capital

阳光保险集团 Sunshine Insurance Group

中信建投资本 CHINA CAPITAL

国信资本 GUOSEN CAPITAL

Agenda

1. Limitations and Opptunites of Traditional Verification Methods
2. Multi-Agent Verification Framework
3. Comparative Demonstration
4. Evaluation
5. Discussion
6. Q&A

1. Limitations of Traditional Verification Methods

1. Limitations and Opportunities of Traditional Verification Methods

Phase	Traditional Approach	Intelligent Approach
Specification Analysis	Manual Analysis	Auto-generation + Manual Calibration
Verification Planning	Manual Analysis	Auto-generation + Manual Calibration
Environment Setup	Framework Auto-generation + Manual Coding	Auto-generation + Manual Calibration

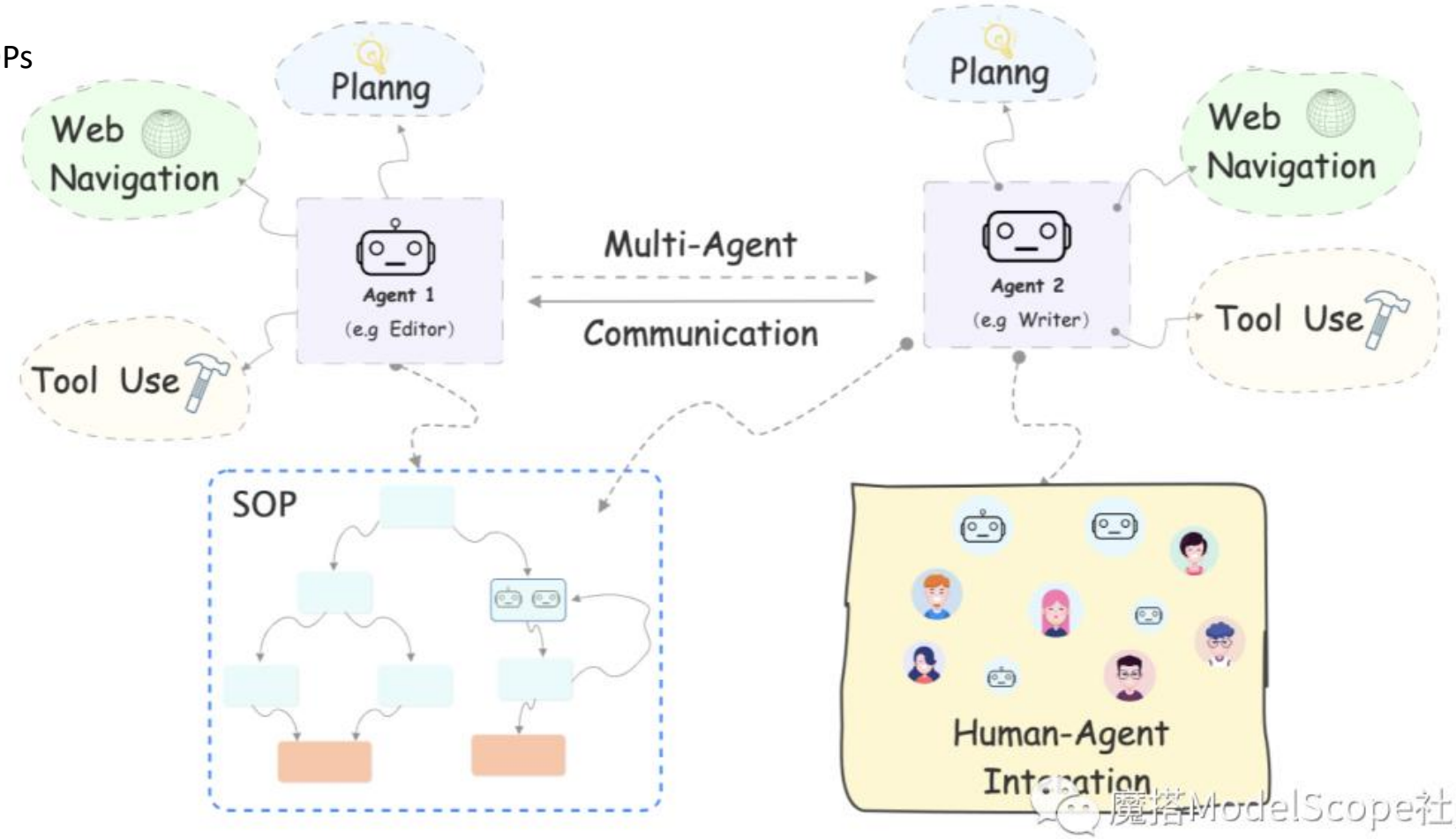
TABLE I
Comparison between Traditional Verification Methods and Intelligent Verification Methods

- **New Technical Trends**
  - Multimodal Parsing
  - RAG Integrates Domain Knowlage
  - Multi-Agent Systems
- **AI Verification Application Opportunities**
  - Design specification processing
  - Testbench construction
  - Methodological innovation
- **Challenges**
  - simple conversational approaches struggle to address such complex problems.

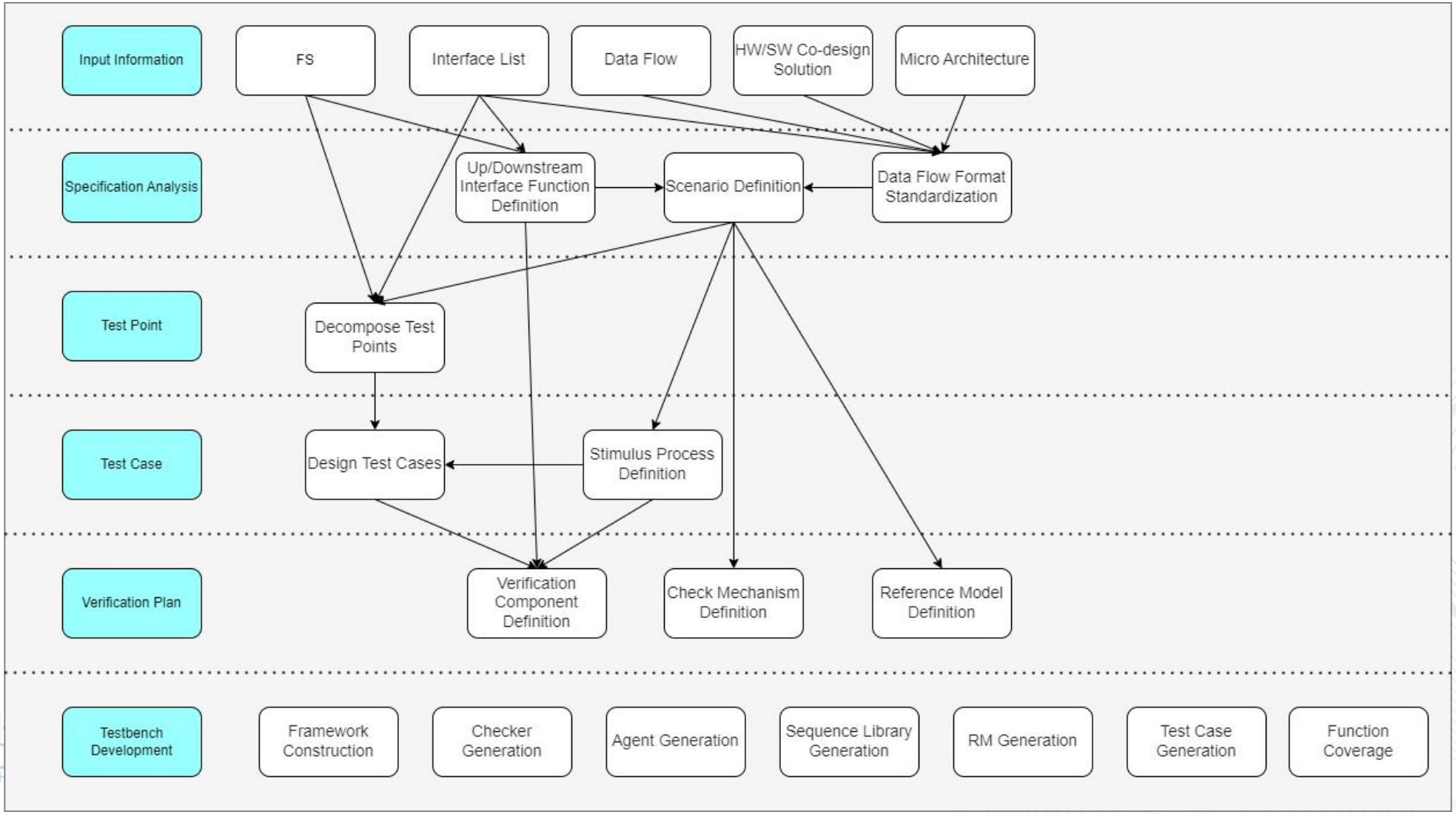
2. MAVF - Multi-Agent Verification Framework

2. MAVF : Multi-agent technology

- **Core Features:**
  - **Process Stability**: Integrates industry SOPs to achieve precise domain knowledge mapping, ensuring output consistency
  - **Role Specialization**: Differentiated intelligent agent role configurations, building systematic solutions
- **System Modules:**
  - Distributed Multi-Agent Architecture
  - Unified Interaction Environment Platform
  - Standardized Process Control Protocol
  - Quality Verification System
  - Intelligent Routing Communication Mechanism
  - Event Subscription Response Mechanism

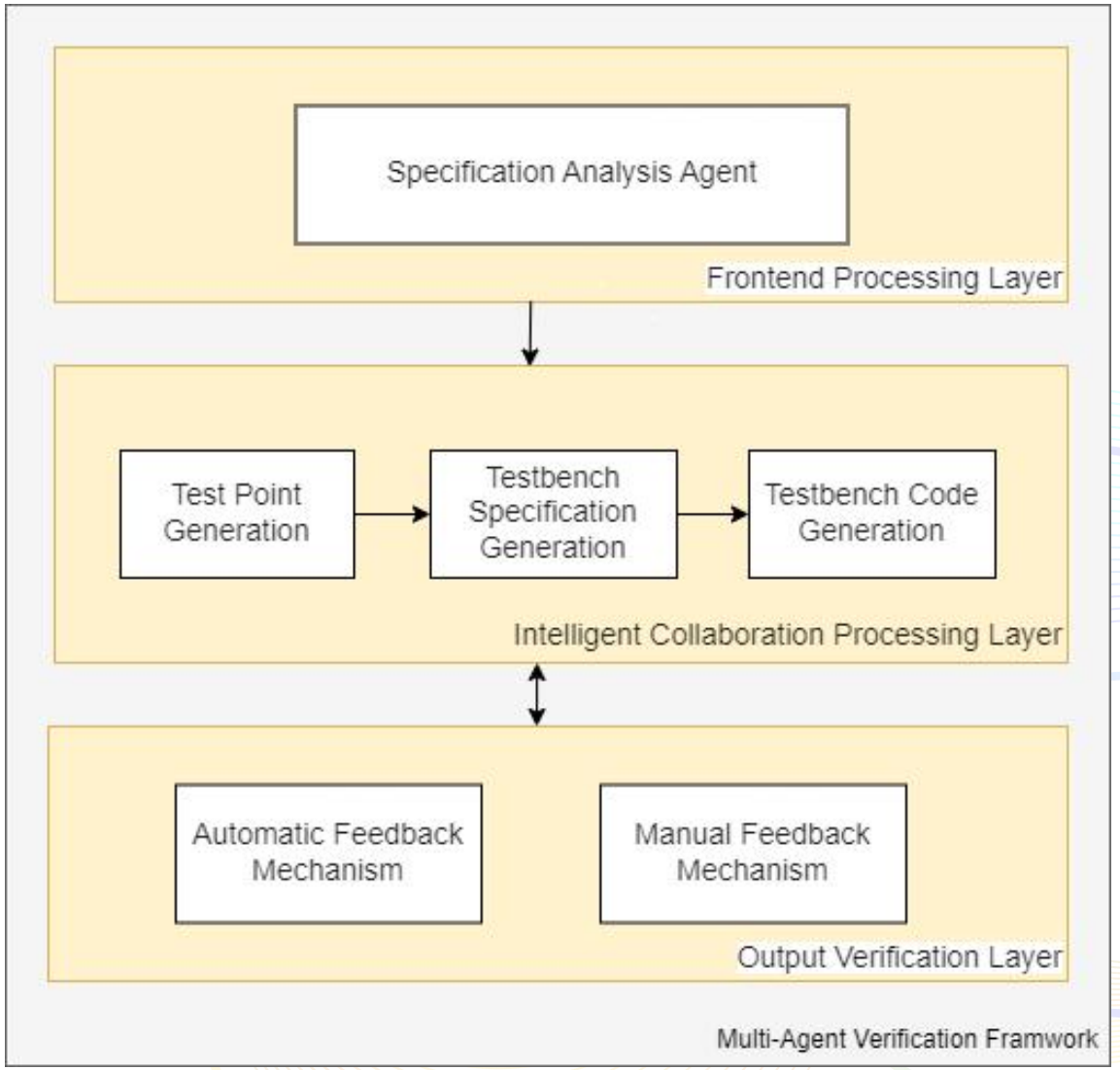


2. MAVF : Module-Level Verification Flow Analysis



2. MAVF : Architecture

- **Frontend Processing Layer**
  - Implements unified structural conversion of multi-modal design documents, generating standardized design specification libraries through normative parsing agents according to sub-SOPs, providing a unified input source for downstream processes.
- **Agent Collaboration Layer**
  - Implements three-stage collaboration based on workflow engineEach agent strictly follows the main SOP process, triggering downstream tasks through phase acceptance
- **Closed-loop Verification Layer**
  - Adopts ReAct chain of thought to implement:Forms an automated quality loop of planning-execution-verification

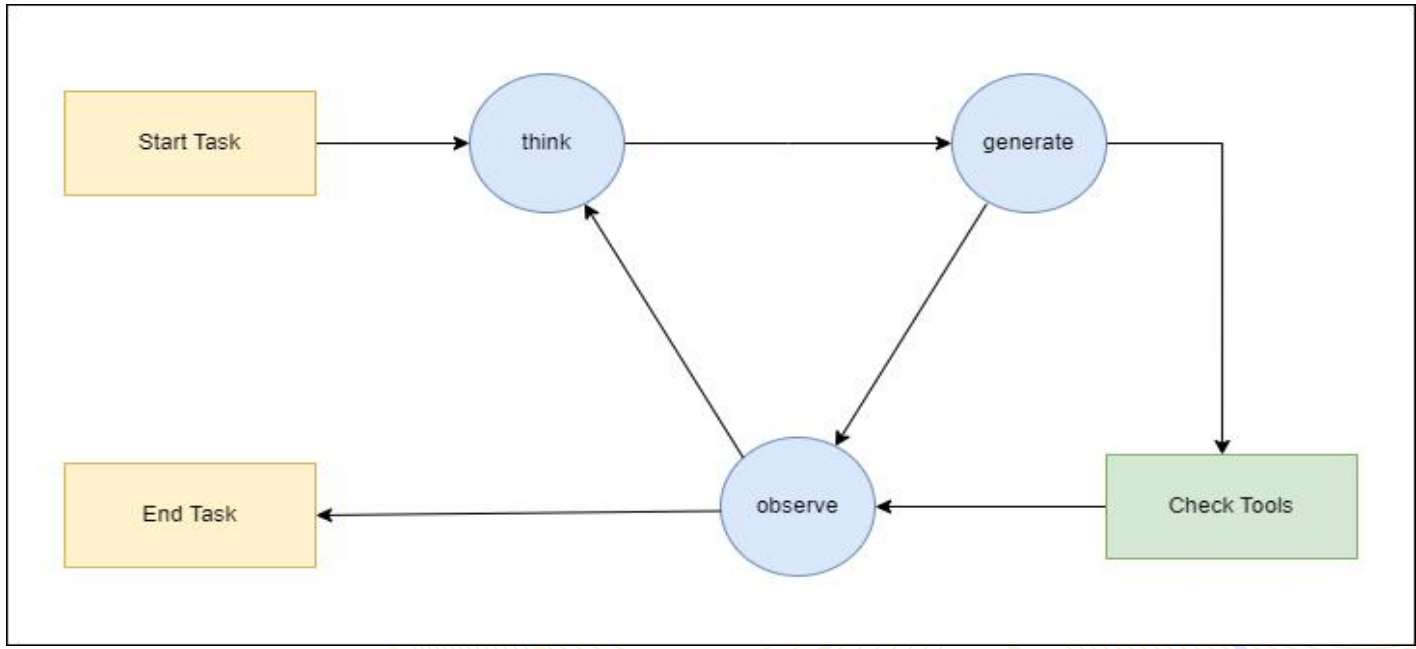


2. MAVF : Agent

- **A. Specification Parsing Agent**
  - Input Processing - Standardize document formats
  - Information Extraction
  - Output Integration - JSON template
- **B. Verification Plan Generation Agent**
  - Test point decomposition
  - Test case generation
  - Inspection mechanism
- **C. Testbench Spec Generation Agent**
  - Establishing the testbench architecture and creating a topological diagram
  - Determining the functionality, quantity, and hierarchical relationships between verification components
  - Providing specific definitions for core data structures and driving functions within components
- **D. Testbench Code Generation Agent**
  - Framework Level (UVM Architecture)
  - Transaction Level (Data Flow)
  - Scenario Level (Test Cases)
  - Industry Compatibility

2. MAVF : Quality Assurance Mechanisms

- **Dynamic Verification Cycle**
  - AI Generation → Automated Review → Human Confirmation
  - Cyclic Iteration Until Standards Are Met
- **Multi-dimensional Consistency Checking System**
  - Verification Planning Phase - Orthogonal Coverage Verification/Scenario Completeness Validation
  - Testbench Generation Phase
    - AI Semantic Check (Grammar + Functional)
    - Manual Review (Architecture Rationality/Completeness of Checking Mechanisms)
  - Execution Feedback Phase
    - EDA Tools Log Review
- **Human-Machine Collaboration Principles**
  - Quality Loop: Automated pre-screening → Expert review
- **Trustworthy AI Mechanism**
  - Establish traceable generation-verification evidence chains



- ***Enhanced assistance tool, not a replacement for human work***

2. MAVF : Specification Parsing Agent - Output


```
"FS": [
    {
        "fs_id": "FS_1",
        "description": "支持高性能64位AXI主接口",
        "sub_fs": [
            {
                "fs_id": "FS_1.1",
                "description": "支持最大8个DMA通道"
            },
            {
                "fs_id": "FS_1.2",
                "description": "支持64位数据宽度"
            },
            {
                "fs_id": "FS_1.3",
                "description": "支持16-512字节的数据缓冲区大小"
            },
            {
                "fs_id": "FS_1.4",
                "description": "支持32位地址位宽"
            }
        ]
    },
    {
        "fs_id": "FS_2",
```

```
{
    "group_name": "AXI_Read",
    "description": "AXI读主接口信号组,DMA作为AXI主设备发起读传输",
    "clock_domain": "clk",
    "signals": [
        {
            "signal_name": "ARID0",
            "direction": "output",
            "width": 1,
            "from": "dma",
            "to": "axi_slave"
        },
        {
            "signal_name": "ARADDR0",
            "direction": "output",
            "width": 32,
            "from": "dma",
            "to": "axi_slave"
        },
```

```
"reg_list": [
    {
        "offset": "0x00 + channel_num * 0x100",
        "reg_name": "CMD_REG0",
        "field": [
            {
                "field_name": "RD_START_ADDR",
                "MSB": 31,
                "LSB": 0,
                "access_type": "RW",
                "default_value": "0",
                "type": "动态配置寄存器",
                "repeat_num": 8,
                "description": "读缓冲区起始地址,以字节为单位,无对齐限制"
            }
        ]
    },
    {
        "offset": "0x04 + channel_num * 0x100",
```

```
"scenario_list": [
    {
        "scenario_name": "命令链表传输",
        "description": "DMA从内存中读取命令链表,自动执行多个传输任务",
        "数据流": "1. DMA读取当前命令的传输参数\n2. 执行配置的传输任务\n3. 检查CMD_LAST,如果为0则从CMD_NEXT_ADDR读取下一个命令\n4. 重复步骤1-3直到遇到CMD_LAST=1的命令",
        "配置流程": "1. 在内存中构建命令链表\n2. 配置通道命令寄存器:\n   - BUFFER_SIZE=0\n   - CMD_SET_INT=0\n   - CMD_LAST=0\n   - CMD_NEXT_ADDR=第一个命令的地址\n3. 配置其他必要的静态寄存器\n4. 写CH_START_REG启动通道",
        "type": "正常场景"
    },
    {
        "scenario_name": "通道暂停恢复",
```

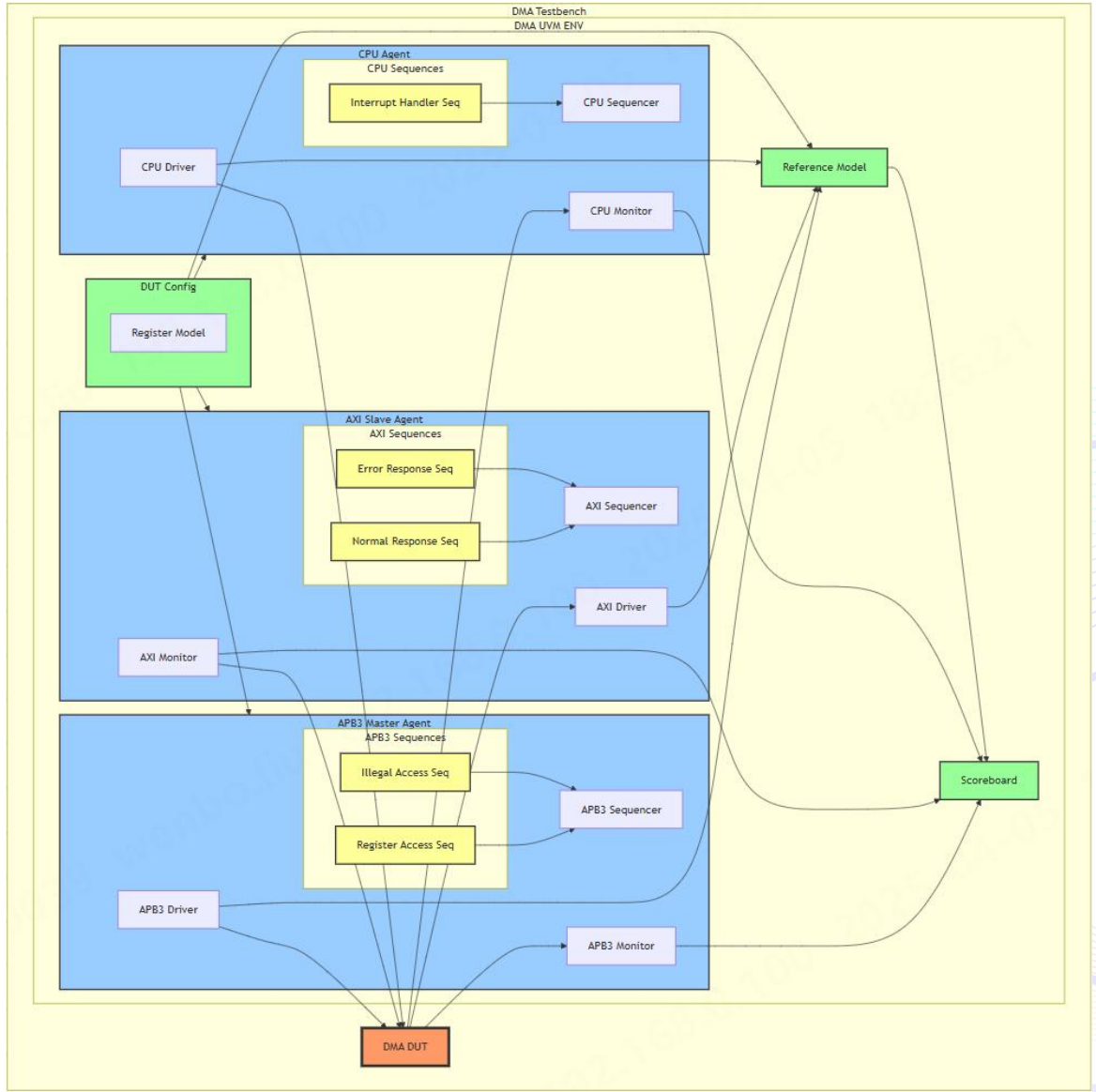
2. MAVF : Verification Plan Generation Agent


```
"TP": [
    {
        "应用场景": [
            {
                {
                    "tp_l2_name": "单通道传输",
                    "tp_l3": [
                        "读写突发长度不同的传输",
                        "读写地址不对齐的传输",
                        "读写跨4K边界的传输"
                    ]
                },
                {
                    "tp_l2_name": "多通道并发传输",
                    "tp_l3": [
                        "2/4/8个通道并发传输",
                        "通道间读写交织传输"
                    ]
                }
            ]
        },
```

```
"TC": [
    {
        "单场景随机测试用例": [
            {
                "tc_name": "tc_rand_independent_single",
                "tc_description": "独立模式单通道随机传输测试",
                "cfg_description": "随机配置传输参数,包括非对齐地址、跨4K边界等场景",
                "constrain_description": "1. 随机读写地址(对齐/非对齐)\n2. 随机传输大小\n3. 允许跨4K边界",
                "tp_map": "功能规格.数据宽度验证.64位数据传输.*, 功能规格.数据缓冲区验证.缓冲区大小配置.*"
            },
            {
                "tc_name": "tc_rand_joint_single",
                "tc_description": "联合模式单通道随机传输测试",
                "cfg_description": "随机配置传输参数,包括非对齐地址、跨4K边界等场景",
                "constrain_description": "1. 随机读写地址(对齐/非对齐)\n2. 随机传输大小\n3. 允许跨4K边界",
                "tp_map": "应用场景.联合模式传输.单通道传输.*"
            },
            {
                "tc_name": "tc_rand_cmdlist",
                "tc_description": "随机命令链表传输测试",
                "cfg_description": "随机构建命令链表,包括散列-聚集和循环缓冲区场景",
                "constrain_description": "1. 随机命令数量(2-16)\n2. 随机命令地址(连续/不连续)\n3. 随机传输参数",
                "tp_map": "应用场景.命令链表传输.*"
            }
```

2. MAVF： Testbench Specification Agent


```
[verif_components_hierarchy]
[{
    "TB": {
        "interface": [ ···
        ],
        "top_tb": [
            {
                "item": "dma_tb_top",
                "item_description": "顶层testbench,实例化DUT、接口和test",
                "reuse": "no"
            },
            {
                "item": "clk_rst_gen",
                "item_description": "时钟复位生成模块",
                "reuse": "yes"
            }
        ],
        "test": [
            {
                "env": [
                    {
                        "agent_name": "apb3_agent",
                        "agent_description": "APB3配置接口代理,负责寄存器配置访问",
                        "agent_type": "master",
                        "agent_sequence_description": "实现寄存器读写、非法访问等场景",
                        "agent_interface": "连接DUT的APB3从接口",
                        "reuse": "vip"
                    },
                    {
                        "agent_name": "axi_agent",
                        "agent_description": "AXI数据传输接口代理,负责数据传输",
                        "agent_type": "slave",
                        "agent_sequence_description": "实现正常传输、错误响应等场景",
```



2. MAVF : Testbench Code Generation Agent


```
dma_verif/
├── env/
│   ├── dma_env.sv
│   ├── dma_env_cfg.sv
│   ├── dma_env_pkg.sv
│   ├── dma_reference_model.sv
│   ├── dma_scoreboard.sv
│   └── dma_virtual_sequencer.sv
├── ral/
│   └── dma_ral_pkg.sv /* RAL model */
├── agents/
│   ├── cpu_agent/ /* user defined Agent */
│   │   ├── cpu_agent.sv
│   │   ├── cpu_driver.sv
│   │   ├── cpu_monitor.sv
│   │   ├── cpu_sequencer.sv
│   │   ├── cpu_seq_item.sv
│   │   └── cpu_pkg.sv
│   ├── apb3_agent/ /* Synopsys VIP */
│   │   └── apb3_pkg.sv
│   └── axi_agent/ /* Synopsys VIP */
│       └── axi_pkg.sv
├── sequences/
│   ├── dma_virtual_seq.sv
│   ├── dma_seq_lib.sv
│   └── dma_seq_pkg.sv
├── tests/
│   ├── dma_base_test.sv
│   ├── dma_test_lib.sv
│   └── dma_test_pkg.sv
├── tb/
│   ├── dma_tb_top.sv
│   ├── dma_clk_rst_if.sv
│   ├── dma_apb3_if.sv
│   └── dma_axi_if.sv
└── sim/
    ├── Makefile
    ├── run.f
```

```
class dma_env extends uvm_env;
  virtual function void connect_phase(uvm_phase phase);
    // Connect register model
    reg_model.default_map.set_sequencer(apb3_agt.sequencer, reg2apb);
    reg_model.default_map.set_auto_predict(1);

    // Connect virtual sequencer
    vseqr.apb3_sqr = apb3_agt.sequencer;
    vseqr.axi_sqr = axi_agt.sequencer;
    vseqr.cpu_sqr = cpu_agt.sequencer;
    vseqr.reg_model = reg_model;

    // Connect monitors to reference model
    apb3_agt.monitor.item_collected_port.connect(rm.apb_analysis_imp);
    cpu_agt.monitor.item_collected_port.connect(rm.cpu_analysis_imp);

    // Connect monitors to scoreboard
    apb3_agt.monitor.item_collected_port.connect(sb.apb_analysis_imp);
    axi_agt.monitor.item_collected_port.connect(sb.axi_analysis_imp);
    cpu_agt.monitor.item_collected_port.connect(sb.cpu_analysis_imp);

    // Connect reference model to scoreboard
    rm.predict_ap.connect(sb.rm_analysis_imp);
  endfunction
```

```
class dma_reference_model extends uvm_component;
  virtual function void predict_transfer(apb_seq_item item);
    mem_data_trans predict_trans;
    int channel_id;

    // Extract channel ID and command info
    channel_id = get_channel_id(item.paddr);

    // Create prediction transaction
    predict_trans = mem_data_trans::type_id::create("predict_trans");

    // Handle different transfer modes
    if(channel_status[channel_id].joint_mode) begin
      handle_joint_mode(channel_id, predict_trans);
    end else begin
      handle_independent_mode(channel_id, predict_trans);
    end

    // Handle command list mode
    if(cmd_queue[channel_id].cmd_list_mode) begin
      handle_cmdlist(channel_id, predict_trans);
    end

    // Send prediction to scoreboard
    predict_ap.write(predict_trans);
  endfunction
```

```
class dma_virtual_sequence extends uvm_sequence;
  task config_cmd(int ch_id, bit[31:0] rd_addr, bit[31:0] wr_addr, int size, bit set_int, bit last, bit[31:0] next_addr);
    // Configure command registers
    write_reg(reg_model.CMD_REG0[ch_id], rd_addr);
    write_reg(reg_model.CMD_REG1[ch_id], wr_addr);
    write_reg(reg_model.CMD_REG2[ch_id], size);
    write_reg(reg_model.CMD_REG3[ch_id], {next_addr[31:4], last, set_int});
  endtask
```

3. Comparative Demonstration

3. Comparative Demonstration – Test Plan Analysis

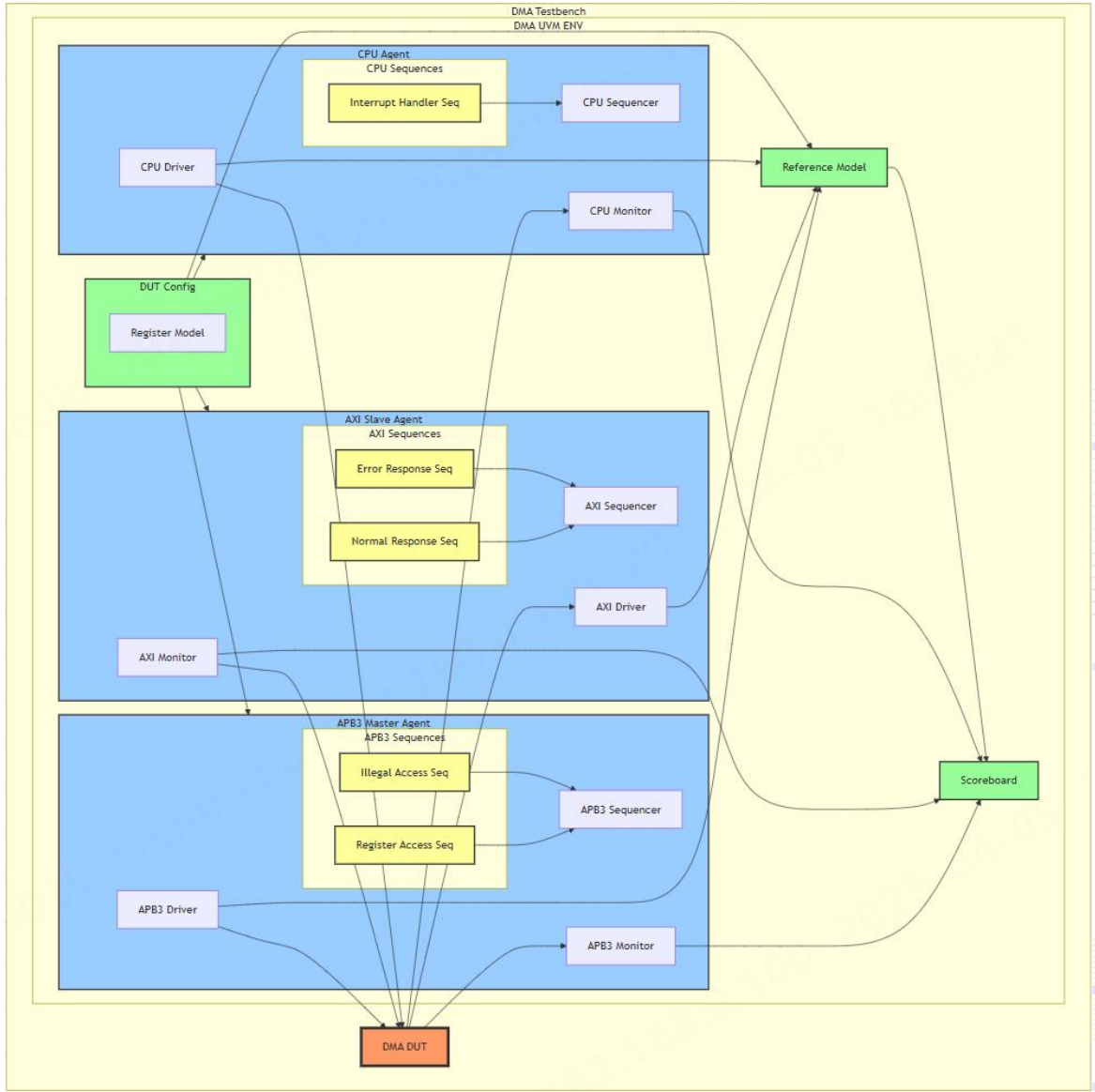
Table V
Comparison of the effectiveness of Module_A in two scenarios: fully automated execution in MAVF versus test point generation through conversational interaction with LLM. Due to space limitations, only partial content is presented.

MAVF	Chat

"TP": [
{
"tp_l1_name": "Address Width Conversion Function",
"tp_l2": [
{
"tp_l2_name": "Address Width Parameter Test",
"tp_l3": [
" Supports S_ADDR_W bit configuration",
" Supports M_ADDR_W bit configuration"
]
},
{
"tp_l2_name": "Address Conversion Calculation Test",
"tp_l3": [
"When window matches, output address = {ext_addr, 20'd0} + axi_slave_addr - region_start_addr",
"When no window matches, output address = {default_slave_address[M_ADDR_W-1:12], axi_slave_addr[11:0]}"
]
}
]
},
{
"tp_l1_name": "Window Configuration Function",
"tp_l2": [
{
"tp_l2_name": "Window Quantity Configuration Test",
"tp_l3": [
"Supports configuration of 1-64 windows",
"Register writes beyond the configured window count are invalid and read as 0"
]
},
{
"tp_l2_name": "Window Address Configuration Test",
"tp_l3": [
"Start address and end address minimum granularity is 1MB",
"Addresses must be 8-byte aligned"
]
}
]
}
]
	1. Address expansion functionality verification:
	(1) Verify that M7 (32-bit address bus) can correctly access N2 (48-bit address bus) address space through jm_soc_remap
	(2) Test address translation from 32-bit to 48-bit addresses
	2. Window configuration verification:
	(1) Test different window quantity configurations (16, 32, 64)
	(2) Verify start and end address configuration for each window
	(3) Test window enable/disable functionality (default disabled)
	(4) Verify 1MB minimum granularity for window addresses

3. Comparative Demonstration – Testbench Architecture


```
class test_prog;
  dma_env         env;
  dma_config      cfg;
  virtual axi_if  axi_vif;
  virtual apb_if  apb_vif;

  function new();
    env = new();
    cfg = new();
  end

  task env_config();···
  endtask

  // 测试执行
  task run();
    basic_transfer_test();
    mode_switch_test();
    cmd_list_test();
    error_handling_test();
  endtask

  task basic_transfer_test();
    dma_trans trans;

    trans = new();
    trans.channel = 0;
    trans.rd_addr = 32'h4000_0000;
    trans.wr_addr = 32'h5000_0000;
    trans.size = 256;
    trans.burst_size = 64;

    env.start_transfer(trans);
    wait(trans.done);
    check_results(trans);
  endtask
endclass
```



3. Comparative Demonstration – Sequence


```
class test_prog;
  dma_env         env;
  dma_config      cfg;
  virtual axi_if  axi_vif;
  virtual apb_if  apb_vif;

  function new();
    env = new();
    cfg = new();
  end

  task env_config();···
  endtask

  // 测试执行
  task run();
    basic_transfer_test();
    mode_switch_test();
    cmd_list_test();
    error_handling_test();
  endtask

  task basic_transfer_test();
    dma_trans trans;

    trans = new();
    trans.channel = 0;
    trans.rd_addr = 32'h4000_0000;
    trans.wr_addr = 32'h5000_0000;
    trans.size = 256;
    trans.burst_size = 64;

    env.start_transfer(trans);
    wait(trans.done);
    check_results(trans);
  endtask
endclass
```

```
class dma_cmdlist_seq extends uvm_sequence #(axi_slave_transaction);
    `uvm_object_utils(dma_cmdlist_seq)

    rand int channel = 0;
    bit[31:0] cmd_list[5];

    function new(string name="dma_cmdlist_seq");
        super.new(name);
        initialize_cmd_list();
    endfunction

    virtual task body();
        setup_command_list();
        start_dma_transfer();
        wait_for_interrupt();
    endtask

    virtual task setup_command_list();
        axi_slave_transaction tr;
        // Write command list to memory
        foreach(cmd_list[i]) begin
            tr = axi_slave_transaction::type_id::create("tr");
            tr.addr = 'h3000_0000 + i*16;
            tr.data = cmd_list[i];
            tr.burst = INCR;
            tr.len = 3; // 4x64bit words per command
            start_item(tr);
            finish_item(tr);
        end
    endtask

    virtual task start_dma_transfer();
        apb_master_seq_item tr;
        // Configure initial command pointer
        write_reg(channel, 'h0C, 'h3000_0000); // CMD_NEXT_ADDR
        write_reg(channel, 'h40, 1);           // CH_ENABLE
    endtask
```

```
    function void initialize_cmd_list();
        // Create scatter-gather command list
        cmd_list[0] = {'h4000_0000, 'h5000_0000, 256, 'h0}; // CMD0
        cmd_list[1] = {'h4000_1000, 'h5000_8000, 256, 'h0}; // CMD1
        cmd_list[2] = {'h4000_2000, 'h5001_5000, 256, 'h0}; // CMD2
        cmd_list[3] = {'h4000_3000, 'h5001_7000, 256, 'h0}; // CMD3
        cmd_list[4] = {'h4000_4000, 'h5002_5000, 256, 'h3}; // Last CMD
    endfunction

    task write_reg(int ch, bit[12:0] offset, bit[31:0] data);
        apb_master_seq_item tr = apb_master_seq_item::type_id::create("tr");
        tr.addr = (ch << 8) + offset;
        tr.data = data;
        tr.op = APB_WRITE;
        start_item(tr);
        finish_item(tr);
    endtask
endclass
```

4. Evaluation

4. Evaluation : Evaluation Set

Name	Input token	Output token
openai/4o-mini	$0.15/M tokesn	$0.6/M tokens
anthropic/claude-3.5-sonnet	$3/M tokens	$15/M tokens
deepseek/deepseek-r1	$0.55/M tokens	$2.19/M tokens

TABLE II
The prices of different models used in the evaluation process

Module Name	Code Size (Lines)	Documentation (Words)	Functionality Description
MODULE_A	1706	1500	Support address remapping for multiple address ranges.
MODULE_B	4565	5500	Supports multi-channel DMA with Register and Command list modes
MODULE_C	20495	21000	Supports protocol conversion and multi-Ring management

TABLE III
Different DUTs (Devices Under Test) used in the evaluation process

4. Evaluation : Evaluation Criteria

- **Evaluation Objectives**
  - Ensure the correctness/completeness of documents and testbench implementation, with accuracy as the core metric
- **Evaluation Methods**
  - Compare against manual verification baseline
- **Accuracy Evaluation System**
  - Specification Analysis: Documentation information error rate (number of errors/total volume)
  - Verification Planning: TP/TC decomposition error rate (proportion of missing/incorrect items)
  - Test Platform: Specification error rate (number of words requiring modification/total generated volume)
  - Code Generation: Code error rate (number of lines requiring modification/total generated lines)

4. Evaluation: Accuracy

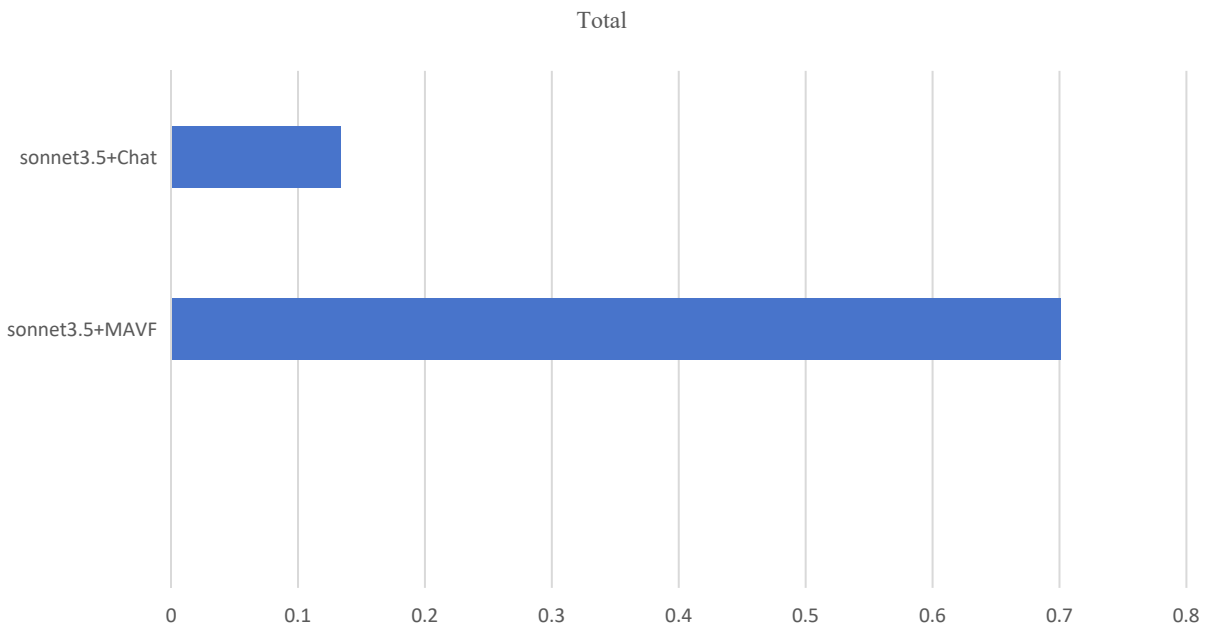


Figure 5. Evaluation Results of Two Different Mode (Summary)

sonnet3.5+MAVF represents tests using anthropic/claude-3.5-sonnet3.5 model with full MODULE_B design specifications as input, running MAVF fully automatically without human intervention.
sonnet3.5+Chat represents tests using the same model in conversational mode with full MODULE_B design specification documents as context prompts plus specific task requirements, without human intervention.

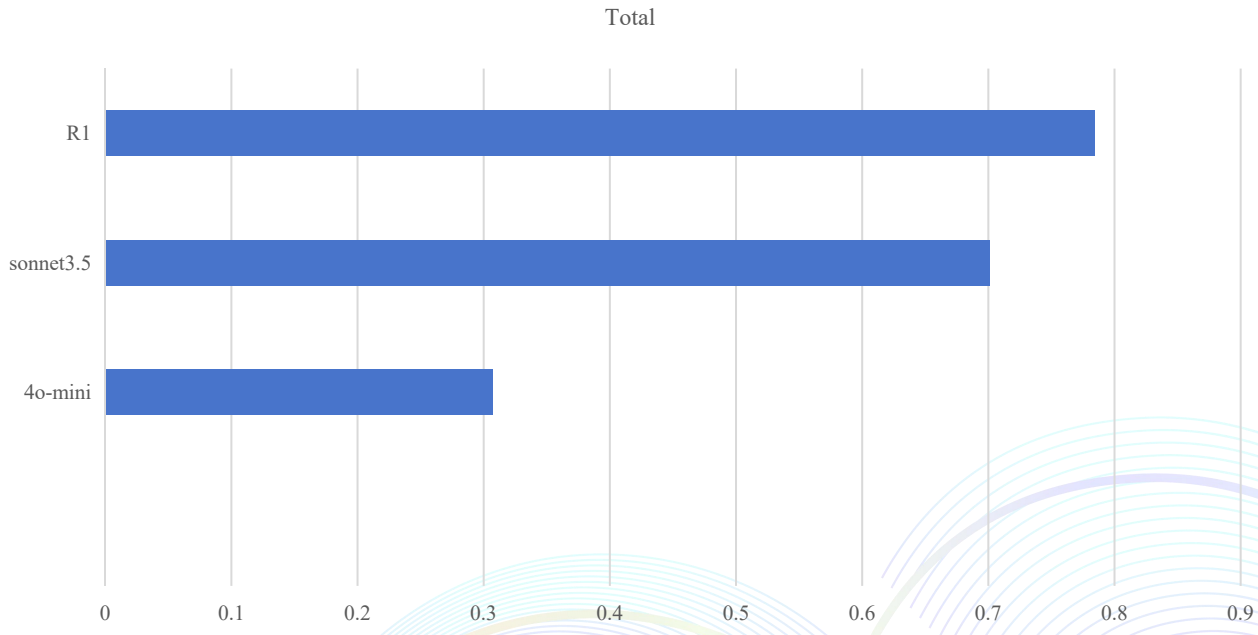


Figure 5. Evaluation Results of Three Different Models (Summary)

4o-mini represents tests using openai/4o-mini model.
r1 represents tests using deepseek/deepseek-r1 model.
Sonnet3.5 represents tests using anthropic/claude-3.5-sonnet3.5 model
All with full MODULE_B design specifications as input and fully automated MAVF execution without human intervention.

4. Evaluation: Accuracy & Efficiency

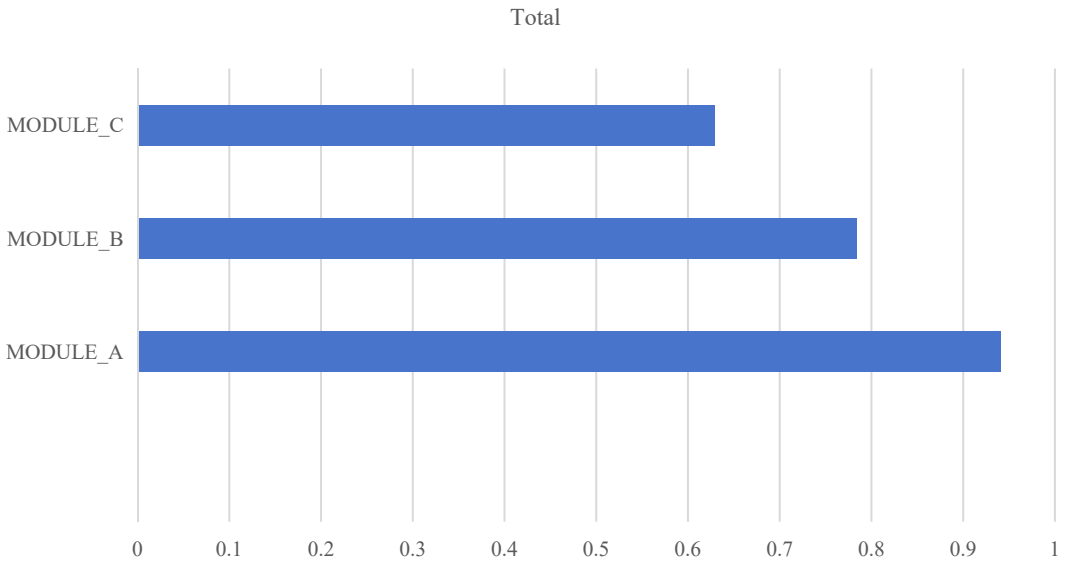


Figure 7. Evaluation Results of Three Different Modules (Summary)

Results show tests using deepseek/deepseek-r1 model on MODULE_A, MODULE_B, and MODULE_C modules respectively, using their full design specifications with fully automated MAVF execution without human intervention.

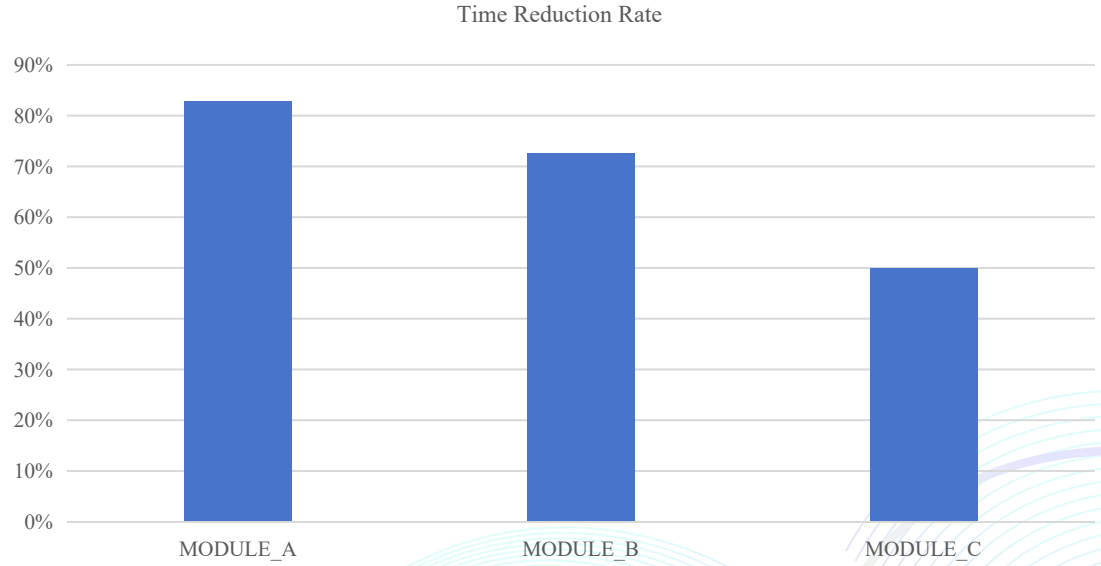


Figure 8. Time reduction rate (Summary)

Time reduction rate shows the percentage of time saved through MAVF assistance ((human time - human&MAVF time)/human time ×100%)

4. Evaluation: Cost

model	MODULE_A			MODULE_B			MODULE_C		
	input	output	total	input	output	total	input	output	total
4o-mini	378k	23k	$0.07	775k	23k	$0.13	878k	47k	$0.16
sonnet3.5	402k	33k	$1.69	603k	50k	$2.55	1204k	79k	$4.79
r1	545k	49k	$0.20	807k	76k	$0.30	1080k	111k	$0.84

Table IV

"input" shows data volume sent to models as prompts. "output" shows information volume returned by models to MAVF. "total" shows the cost calculated based on current model prices for total tokens consumed.

This demonstrates that using MAVF to assist chip verification work **offers excellent cost-effectiveness**, achieving substantial benefits **with minimal resource investment**.

5. Discussion

5. Discussion

- **Innovation Value**
  - ★ Framework: Multi-agent collaboration → Solving engineering implementation problems
  - ★ Process: Workflow decomposition → Achieving efficient GenAI integration
  - ★ Driving IC DV into a new "AI+" paradigm
- **Framework Effectiveness**
  - ✓ Performance significantly better than traditional dialogue methods
  - ✓ Complex design scenarios require high-performance models + human intervention (50%+ efficiency improvement)
- **Resource Efficiency**
  - ✓ Resource costs account for <5% of efficiency gains
  - ✓ Human input at key nodes can achieve improvements in both quality and efficiency
- **Current Challenges**
  - ! Lack of standardized evaluation sets
  - ! Large differences in module functionality (need to establish classification optimization system)
- **Future Optimization Directions**
  - ➢ Module feature classification: Establish template library for different designs
  - ➢ Framework upgrade: Reliability/maintainability/human-computer interaction optimization
  - ➢ Process optimization: Further optimize the granularity of decoupling verification process based on GenAI capabilities
  - ➢ More comprehensive evaluation sets

Q&A

Liuwenbo_7@foxmail.com