arXiv:2601.07593v1 [cs.AR] 12 Jan 2026

带状态变异的GRPO：改进基于大语言模型的硬件测试计划生成

Dimple Vijay Kochar[1]，Nathaniel Pinckney[2],Guan-Ting Liu[3],Chia-Tung Ho[4],Chenhui Deng[4], Haoxing Ren[2],Brucek Khailany[2]1电气工程与计算机科学，麻省理工学院，剑桥，马萨诸塞州2英伟达研究院，奥斯汀，德克萨斯州3英伟达研究院，台湾4英伟达研究院，圣克拉拉，加利福尼亚州
电子邮件：dkochar@mit.edu,npinckney@nvidia.com,dannliu@nvidia.com, chiatungh@nvidia.com, cdeng@nvidia.com, haoxingr@nvidia.com, bkhailany@nvidia.com

摘要—**RTL设计在设计周期早期往往严重依赖临时创建测试平台。尽管大语言模型（LLM）在RTL代码生成方面展现出潜力，但它们推理硬件规格说明并生成有针对性的测试计划的能力仍鲜有研究。我们首次系统研究大语言模型在RTL验证激励生成方面的推理能力，并建立了一个两阶段框架，将测试计划生成与测试平台执行分解开来。我们的基准测试表明，DeepSeek-R1和Claude-4.0-Sonnet等最先进模型，在生成能够通过黄金RTL设计的激励方面，成功率仅为15.7- 21.7%。为改进大语言模型生成的激励，我们制定了一套全面的训练方法，将监督微调与一种新颖的强化学习方法——带状态变异的GRPO（GRPO-SMu）相结合，通过改变输入变异来增强探索能力。我们采用基于树的分支变异策略，构建由等价树和变异树组成的训练数据，突破线性变异方法的局限，提供丰富的学习信号。在这一精选数据集上训练后，我们的70亿参数模型达到33.3%的黄金测试通过率和13.9%的变异检测率，较基线绝对提升17.6%，并优于规模大得多的通用模型。这些结果表明，专门的训练方法能够显著增强大语言模型在硬件验证任务中的推理能力，为半导体设计工作流程中的自动化子单元测试奠定基础。**

索引词—测试计划、大语言模型、GRPO、测试平台、寄存器传输级、激励

I. 引言

硬件验证遵循一条成熟的发展路径，从系统级行为模型逐步发展到功能实现
最终寄存器传输级设计 [1]。系统级和单元级的验证工作流程依赖黄金参考模型、外部检查器和约束随机生成 [2]–[4] 来验证正确性。然而，如此全面的测试套件、参考实现以及大量工程基础设施，通常难以应用于单个寄存器传输级模块或子单元的层级 [5]，，尤其是在设计过程的早期阶段。这凸显了对低级寄存器传输级代码验证的有效自动化方法的需求。

近年来，大语言模型（LLM）取得的进展为自动化RTL工作流 [6]–[11]，带来了希望，

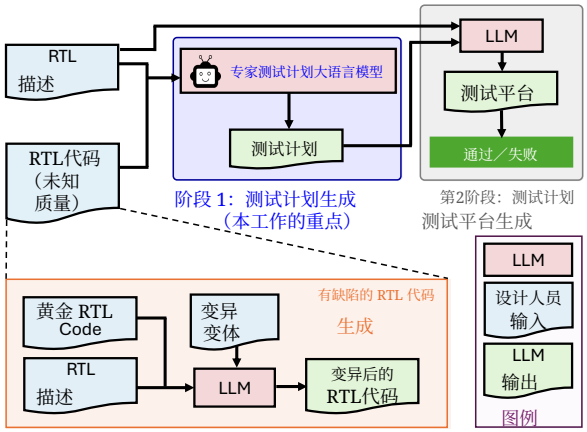

图 1. 我们提出的基于大语言模型的测试计划生成与实现两阶段框架（上）。用于测试计划评估的有缺陷寄存器传输级代码生成（下），详见第III-D节。

专用模型展现出了代码生成和调试任务方面的能力。然而，这些方法主要关注生成后的纠正或测试平台自动化，而非模型内部主动开展验证推理。以往的方法还利用基于大语言模型的智能体生成更简单的 Python 功能模型，以用于验证 [12], [13]。然而，这类庞大基础设施框架在理解具有复杂时序行为的复杂RTL方面的有效性，仍未得到充分研究。

为此，我们在工作中探讨以下核心问题：大语言模型能否“理解”自然语言规范，并结合质量未知的寄存器传输级代码，生成用于调试的高质量定向激励？这项任务要求同时对设计意图、实现细节和潜在故障模式进行推理。这些能力可以在很大程度上实现自动化，减少所需的人工工作，并显著简化子单元验证。

为探究这一问题，我们开发了一种新颖的两阶段测试计划生成框架（如图 1上方所示）。第一阶段负责生成子单元测试计划，以验证质量未知的寄存器传输级代码；第二阶段则创建可执行的测试平台来实现这些测试计划。我们提出的中间测试计划是对大语言模型推理过程的结构化文字表述

，用于区分 RTL 描述与 RTL 设计代码。我们通过在黄金（正确）和变异（含缺陷）的 RTL 上测试最终测试平台来评估该框架，并将正确版本通过率和变异检测率作为评估指标。总体而言，得益于任务分解，我们的两阶段框架不仅将下游验证性能提升了 10-12%（详见第 V-D节），还增强了人工干预能力，以便审查并改进测试计划。我们使用这种方法对最先进的大语言模型进行基准测试，展示了表现最佳的大语言模型如何获得 $\sim 22\%$ 的分数；这揭示了大语言模型在硬件验证推理方面的显著局限。

受这一局限的启发，我们首先探索监督微调（SFT）以增强模型推理能力。具体而言，我们在精选数据集上对 Deepseek-R1蒸馏版Qwen-7B 大语言模型进行微调，该数据集包含展示逐步验证规划的推理轨迹，用于区分正确和存在错误的寄存器传输级实现。然而，SFT仅带来了有限的改进，这表明单靠模仿学习无法捕捉有效测试计划生成所需的复杂推理能力。

最后，我们应用组相对策略优化（GRPO） [14]，这一强化学习（RL）技术，使大语言模型直接从验证结果中学习，而不是模仿推理模式。在此基础上，我们提出带状态变异的GRPO（GRPO-SMu），通过在训练回合中系统地改变输入变异来增强探索能力。该方法使模型能够在单个训练回合中接触多种多样的错误模式。为实现这种训练方法并针对每个寄存器传输级生成多个变异，我们开发了一种基于树的代码变异策略，突破了线性变异方法的限制。我们的方法创建功能等价的寄存器传输级实现，并构建分支变异树，通过经验证的变换系统地探索错误空间。在这些精选数据上进行训练后，我们的7B大语言模型达到了33.3%的黄金通过率，相较于未训练基线提升了 $2\times$，并超越了规模大得多的通用大语言模型。

总而言之，我们提出以下贡献：

- 我们开发了一个两阶段框架，用于系统地评测大语言模型在测试计划生成方面的推理能力
- 我们提出了带状态变异的GRPO（GRPO-SMu），通过在单个训练回合中为每个规格说明变换输入变异，增强强化学习的探索能力
- 我们提出了一种新颖的基于树的变异策略，用于数据集整理：根据给定的寄存器传输级代码构建分支等价树和变异树，突破线性变异方法的局限

II. 相关工作

A. RTL代码生成与验证

近年来，大语言模型的进展推动了自动化RTL工作流领域的大量研究。领域特定的微调方法 [6]–[11]表明，专门化模型在Verilog代码基准测试上的RTL代码生成方面可以胜过通用替代方案 [15],[16]。更复杂的系统采用多智能体框架 [17]–[21]，、多候选采样 [22]，[23],和强化学习 [24]，以提升生成质量。

调试方法也探索了检索增强生成 [25]，[26],迭代精炼 [27]–[29], 用于覆盖率的刺激箱 [30], 以及对比嵌入技术 [31]。自动化测试平台生成 [32]–[35]也已成为一个重要的研究领域。然而，这些方法主要针对生成后的修正或测试平台自动化，而非模型内部主动进行验证推理。

B. 数据集整理与变异测试

VeriDebug 等先前工作 [31]通过插入错误来构建数据集，但未验证功能差异，也未针对每份寄存器传输级代码提供多个变体。BugGen [36]采用智能体策略进行基于模块的错误注入，但仅覆盖少数设计。我们的工作不仅更为全面，还处理了数据污染问题 [11], [37],：专有模型可能因训练数据截止日期而包含基准测试相关信息 [15], [16] 。

C. 语言模型的强化学习

监督微调提供了基础推理能力，但在复杂决策任务中存在局限性。近期的推理系统表明，SFT 可在策略优化前稳定模型，缓解循环或推理链条不连贯等失效模式 [38]。组相对策略优化（GRPO） [14] 通过采样多个输出并使用平均奖励作为基线，无需单独的价值网络。GRPO 已在长时间推理任务中展现出有效性 [39]，近期也在 Verilog 代码生成方面取得了令人鼓舞的成果 [24]。

III. 问题构建：测试计划生成

我们的工作重点是开发有效的自动化方法，以辅助低层级 RTL 设计验证，尤其是在缺少黄金参考模型的情况下。我们通过引入测试计划的中间表示，构建了一个两阶段测试计划生成框架来形式化这一任务。下面将深入阐述我们的问题构建。

A. 两阶段框架构建

图 1 （上）展示了我们提出的两阶段测试计划生成流程，该流程引入了一种新的分解方法以提升性能。第一阶段中，给定自然语言 RTL 描述以及可能含有错误的 RTL 设计，模型需要生成测试计划（见第 III-B 节）。第二阶段中，模型生成测试平台，以实现第一阶段生成的测试计划。我们在正确/黄金和存在错误/变异的 RTL代码上运行最终测试平台（详见第 III-D 节）。根据测试平台输出，我们评估并报告整个流程的两个关键指标：（M1）正确版本通过率：黄金设计是否

```
您是数字设计和验证方面的专家。您的任务是生成单元测试
推理.....提供给您的 RTL代码可能正确，也可能存在错误。提供的描述.....
根据问题描述和 RTL实现类，提供：
1. 推理： 如何测试 RTL代码的分步分析.....
2. 单元测试描述： 详细的测试策略和预期行为
问题描述：
设计一个 Verilog 模块，用作流水线寄存器，在特定条件下将控制标志和数据从输入传输到
输出。该模块必须包含.....数据存储器输出和算术逻辑单元结果
为 32 位值，而寄存器地址为 5 位。请使用适当的位宽定义这些输入和输出，并
确保正确处理复位、使能信号和反相标志。
寄存器传输级实现：
模块 dut(
      输入 时钟,
      输入 复位信号,
...
      输出寄存器 [31:0]  ALU结果输出,
      输出寄存器 [4:0]  写寄存器地址输出
);
  始终块 @(上升沿 时钟 or 下降沿 复位信号) begin
          if (!复位信号) begin
              ...
          结束 else 开始
              如果（使能信号）开始
                  寄存器写使能标志输出  <=  反相信号 ? regWriteFLAG_IN : ~regWriteFLAG_IN;
                  存储器到寄存器标志输出  <=  反相信号 ? memToRegFLAG : ~memToRegFLAG;
                  数据存储器输出  <=  数据存储器输入;
                  算术逻辑单元结果 <=  ALU_RESULT_IN;
                  写寄存器地址输出  <=  寄存器写入地址输入;
              end
          end
      end
结束模块
说明：
- 全面分析寄存器传输级代码的功能
- 识别关键行为、边界情况和潜在失效模式
- 设计全面的测试策略，以区分正确的和
错误的实现.....
输出格式：
推理：[对寄存器传输级代码和测试策略进行逐步分析]
单元测试描述：[对测试用例、输入激励和输出的详细描述]
```

图2. 阶段 1 测试计划生成的大语言模型提示示例，如第III-B

```
<思考> 错误的实现修改了一个逻辑门.....为揭露这一点，我将创建..... </思考>
### 差异化策略/思维链推理
故障实现将 `regWriteFLAG` 和 `memToRegFLAG` 输出反相了两次，
实际上抵消了预期的反相。这意味着.....通过应用一个满足以下条件的测试用例：
`inv` 为高电平，且 `regWriteFLAG_IN` 为 1，我们就能观察到这种差异。

### 输入激励
我将应用以下输入：
- 时钟：一个时钟脉冲（用于触发更新）
- 复位信号：1（确保流水线不处于复位状态）
- 使能信号：1（使能寄存器）
- 反相信号：1（测试反相逻辑）
- regWriteFLAG_IN：1（测试反相逻辑的影响）
- 存储器到寄存器标志输入：0（与此测试无关）...

### 预期正确输出
对于给定的输入激励，正确的实现将产生以下输出：
- `regWriteFLAG_OUT`：0（由于 `inv` 为高电平，相对于输入进行了反相）
- `memToRegFLAG_OUT`：1（由于 `inv` 为高电平，相对于输入进行了反相）
- `dataMemoryOut_OUT`：32'h0（未改变）.....

此测试用例可以有效地区分这两个实现.....
```

图3. 阶段 1 测试计划生成的大语言模型响应示例，如第 III-B

正确通过；（M2）变异检测率：是否正确区分变异设计与黄金设计，同时黄金设计通过。

B. 阶段 *1*：测试计划生成：

我们将测试计划定义为一种中间推理评估代理，用于区分黄金（参考）行为与任何有缺陷的 RTL 实现。这种表示方式模拟了人工验证者在编写手动单元测试时的思考过程。在我们的工作中，我们通过一种结构化形式对其进行正式表述，其中包括：(1) 期望的设计描述与已实现的 RTL代码之间的差异，

```
你是一个 Verilog 测试平台生成器。给定一个 Verilog 模块描述，以及一个现有的
测试平台包含随机测试，并结合单元测试推理，生成一个全面的
测试平台，以实现特定的单元测试。

你的任务是：
1. 以提供的基准测试平台为基础
2. 根据单元测试，将随机测试生成替换为定向测试用例.....
3. 纳入标准输出——每个测试输入对应的预期正确输出
4. 实现验证逻辑，将参考实现和被测设计的输出与
标准输出进行比较
5. 确保测试平台能够验证参考实现是否产生正确的
结果，并检测变异/故障实现是否产生错误结果...

描述： {same as in Stage 1 prompt}
测试平台： module stimulus_gen_tb(...);
   初始 begin
      for(sti_for_i=0; sti_for_i<1000000; sti_for_i=sti_for_i+1) begin
         等待(时钟  ==  1'b1);
         复位信号  <=  { $random }; 使能信号 <=  { $random }; 反相信号 <=  { $random };...
         regWriteFLAG_IN  <=  { $random };
         存储器到寄存器标志输入  <=  { $random };...
         wait(时钟  ==  1'b0);
         复位信号  <=  { $random }; 使能信号 <=  { $random };...
模块 测试平台();
  stimulus_gen_tb stim1(.时钟(时钟), .复位信号(复位信号), .使能信号(使能信号),...)
  assign tb_match  =  ({ 寄存器写使能标志输出_参考, 存储器到寄存器标志输出_参考,
参考数据存储器输出, ...
   始终块 @(上升沿 时钟 或 下降沿 时钟) begin
      检查计数  =  检查计数  +  1;
      if(!测试平台_match) begin
         错误计数  =  错误计数  +  1;...
推理/单元测试： {与阶段 1 响应中相同}

重要输出要求：
- 如果原始/参考模块根据所提供的单元测试产生预期输出，则打印 "PASS_ORIGINAL"
根据所提供的单元测试
- 如果被测设计/变异模块产生错误输出，则打印 "FAILED_MUTATED"
- 如果同时满足 PASS_ORIGINAL 和 FAILED_MUTATED 条件，则打印 "SUCCESSFUL_UT"
则.....关键要求：仅在最后根据总体结果打印上述字符串一次，而不是在每个单独的测试之后打印。
如果原始/参考模块通过
所有内容，只有这样才打印“PASS_ORIGINAL”。如果被测设计/变异模块哪怕失败一次，
则打印“FAILED_MUTATED”。如果原始版本失败，不要打印“PASS_ORIGINAL”.....
```

图4. 第2阶段测试计划执行的大语言模型提示示例，如第 III-C

(2) 能够引发这种差异的输入激励，(3) 用于验证的预期输出，以及 (4) 支持性推理。我们在图3中展示了一个测试计划示例。

此阶段的测试计划生成要求大语言模型理解自然语言规范，评估质量未知的给定 RTL代码，并通过测试计划生成正确的激励以揭示错误。生成这一中间测试计划有两大优点：(1) 性能更佳：与以往的任务分解工作类似 [40]，，我们展示了采用两阶段任务分解后，性能提升了 10-12%（第 V-D 节）；与单阶段测试平台生成相比，(2) 控制力更强：与单阶段黑盒测试平台生成相比，中间测试计划让人工验证者能够更好地编辑和改进测试计划。在本研究中，我们主要致力于提升大语言模型在测试计划生成第一阶段的表现，并在图中展示了该阶段使用的提示词。2。

C. 第*2*阶段：测试计划执行

为了执行第一阶段生成的测试计划逻辑，在第二阶段中，一个大型大语言模型（我们使用 LLaMa-3.1-405B）会处理生成的测试计划，并将其与测试平台模板结合，以生成测试平台。测试平台模板是

```
模块 tb;
 reg 时钟;
 reg 复位信号;...
 wire 参考寄存器写入标志输出;
 wire regWriteFLAG_OUT_dut;
 wire 参考寄储器到寄存器标志输出;
 wire 存储器到寄存器标志输出_dut;...
流水线寄存器 good1 (.时钟(时钟), .复位信号(复位信号), .使能信号(使能信号), .反相信号(.反相信号)... )
dut dut1 (.时钟(时钟), .复位信号(复位信号), .使能信号(使能信号), .反相信号(.反相信号)... )
initial begin
   复位信号 = 1;
   使能信号 = 1;
   反相信号 = 1;...
   regWriteFLAG_IN = 1;
   存储器到寄存器标志输入 = 0;
   数据存储器输入 = 32'h0;
...
   if (参考寄存器写入标志输出 == 0 && 参考存储器到寄存器标志输出 == 1 && 参考数据存储器输出
== 32'h0 && ALU结果输出_ref == 32'h0 && 写寄存器地址输出_ref == 5'h0) begin
    如果 (regWriteFLAG_OUT_dut != 0 || memToRegFLAG_OUT_dut != 1 || 数据存储器输出_dut !=
32'h0 || 算术逻辑单元结果_OUT_dut != 32'h0 || 写寄存器地址输出_dut != 5'h0) 开始
     $display("原始版本通过"); $display("变异版本失败"); $display("单元测试成功");
    结束 否则 开始
     $display("单元测试失败"); $display("变异版本通过");
    end
   结束 否则 开始
    $display("FAILED_ORIGINAL"); $display("FAILED_UT");
   end
   $finish;
 end
endmodule
```

图5. 第2阶段测试计划执行的示例大语言模型提示，如第 III-C 节所述

一个简单的自动生成的测试平台骨架存根，有助于避免测试平台生成过程中的语法错误。我们在图4和图5中展示了该阶段的大语言模型提示词和响应。

D. 评估数据创建

在评估中，我们使用了来自 ScaleRTL数据集 的 500 份寄存器传输级代码 [11]。为开展全面研究，我们提示一个大语言模型，为每份寄存器传输级代码生成逼真的变异版本 [36]，并采用我们自行定义的 14 种错误类型之一（详见第 IV-C 节）。我们在图 1 （下方）中展示了这一流程。该流程为每份代码生成三个变异版本，最终得到包含 1,500 份变异寄存器传输级代码的评估数据集。此变异版本生成流程可解决数据污染问题 [37]，，因为专有模型可能包含先前的基准测试信息 [15], [16]，这是由于其训练数据截止日期晚于这些信息的公开发布时间 [11]。

IV. 方法

在测试计划生成的第一阶段，我们使用小语言模型（SLM）。在此，我们介绍针对该任务微调这些小语言模型的训练方法。首先，我们采用监督微调（SFT），以促使模型生成推理轨迹和结构化输出。接下来，我们部署 GRPO-SMu（一种强化学习方法），利用实际 M1/M2 结果提供的信号来微调大语言模型。最后，我们将讨论一种新颖的变异生成策略，用于筛选适合 GRPO-SMu 的高质量数据。下文将介绍这些阶段以及本方法的其他技术细节。

A. 模型准备：监督微调（*SFT*）

图 6（左）展示了我们的 SFT 数据整理流程。我们从 ScaleRTL数据集选取 RTL 设计，并通过注入三种相互独立的变异来构造 SFT 数据对（表 II）到每个代码实例中。此方法优先考虑高收益、具区分性的监督，而非分布覆盖，从而提高刺激信号区分黄金代码与变异代码的概率。该流程如下：从单个 RTL 设计开始，生成变异代码，提示大语言模型创建用于区分正确代码与变异代码的刺激信号测试计划，再提示大语言模型根据该计划创建测试平台，并对其进行仿真。只有满足 M2 条件的样例才会被接受（即黄金代码通过，且变异代码被正确区分）。我们使用 LLaMa-3.1-405B 和 Claude3.7-Sonnet 整理出 1,902 个 SFT 实例。每条被接受的输入—输出轨迹都会补充由 Claude3.7-Sonnet 生成的推理轨迹，从而形成（提示、推理 +测试计划）训练数据对。最后，我们使用 SFT 数据集训练小语言模型，该数据集分为 1,521 个训练样本和 381 个验证样本。图6 （中左）通过一个示例展示了训练样本的输入—输出结构。

B. 强化学习微调（*RL*）：*GRPO-SMu*

我们的工作对组相对策略优化（GRPO）的强化学习微调方法进行了修改和改进 [14]。在 GRPO 中，每个训练回合包含一个输入状态s （即相同的初始提示），并据此采样多个动作 a_1，a_2，..... a_n （即温度不同所产生的不同输出）。每个动作 a_i 都会通过外部奖励模型获得奖励r_i 。奖励分布会形成偏好信号（优势），用于训练原始的动作采样大语言模型。然而，当任务复杂且奖励信号稀疏时（我们的任务设置正是如此），奖励分布可能缺乏变化，即所有奖励都可能同样低劣。在这种情况下，优势趋近于零，导致学习信号极少 [41]。

为了解决这一局限，我们提出了一种新的策略优化方法：带状态变异的GRPO（GRPO-SMu）。具体而言，我们不再从固定状态中采样多个动作，而是从经过系统性多样化的一组状态中采样动作，从而提升动作和奖励的多样性，进而改善大语言模型的训练信号。下文将介绍更多技术细节。

数学表述：在传统GRPO中（图6，右），对于每个状态 s，会从策略 π_{θ_t}中采样多个动作 a_1，.....， a_G。对于我们的任务，状态由RTL描述和变异后的寄存器传输级代码构成，策略模型是大语言模型，动作则是输出的测试计划。根据奖励模型给出的奖励 $r(s,\ a_j)$ （详见第IV-D节），组相对优势的计算方式如下：

$$A^{\pi_{\theta_t}}(s, a_j) = \frac{r(s, a_j) - \mu}{\sigma} \tag{1}$$

其中， μ, σ 分别表示分组奖励的均值和标准差。当所有奖励都相同时，上述优势将为零，也就不会产生训练信号。这种情况尤其可能出现在动作多样性受限的困难样本中，从而导致探索受限。

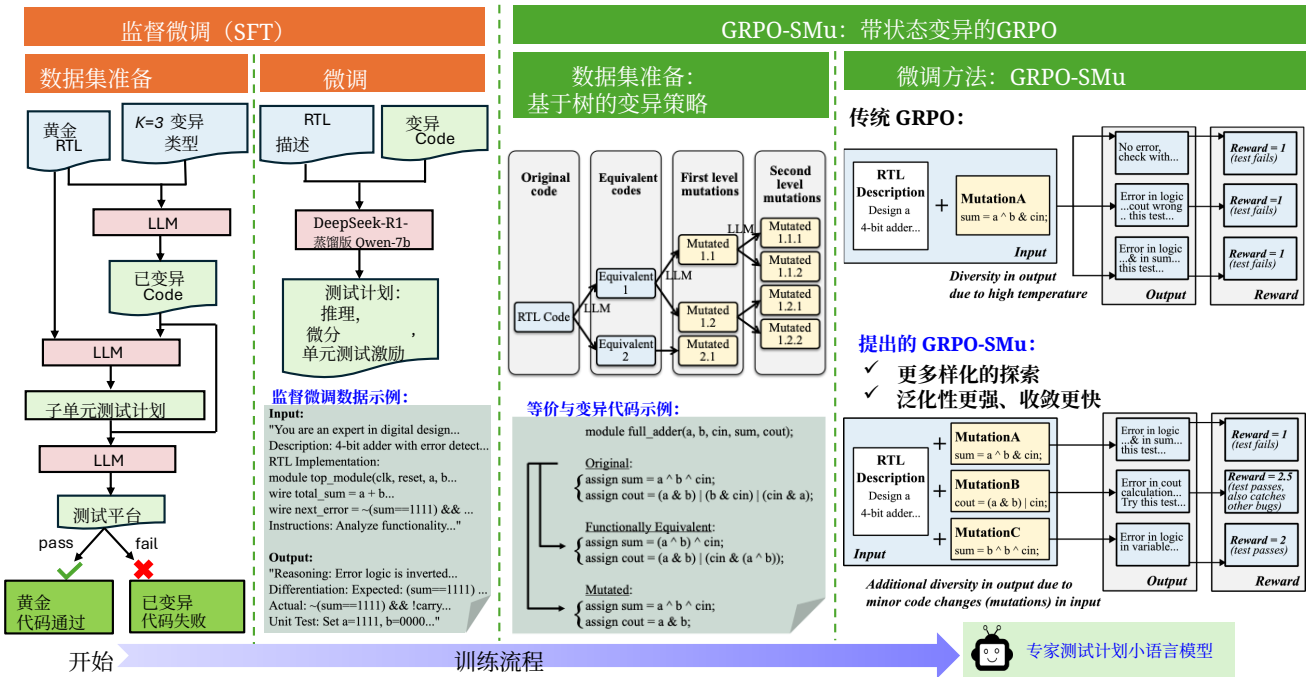

图6. 整体训练流程：通过数据集整理进行 SFT（左），使用示例进行 SFT（中左）；用于强化学习微调的数据集（中右），以及 GRPO-SMu 与传统 GRPO 的比较（右），展示了多样化的探索。

为克服这一局限并鼓励更充分的探-索，我们的 GRPO-SMu 方法（图 6 （右））通过引入多个相关状态来丰富初始状态空间，具体如下：

$$s' = s + \Delta s_{\text{mutation}} \tag{2}$$

其中 $\Delta s_{\text{mutation}_i}$ 表示同一 RTL代码的不同变异版本。由于这些新多样化状态仅略有不同（它们共享相同的描述以及生成变异所依据的基础 RTL代码），这些状态仍足够相似，能够进行有意义的优势计算（式 1）。然而，从这组新的多样化状态中采样动作 $\mathcal{S}' = \{s'_1, s'_2, \dots\}$ 能够产生更多样化的动作。新的优势计算如下：

$$A^{\pi_{\theta_t}}(s'_j, a_j) = \frac{r(s'_j, a_j) - \mu_{\{\mathcal{S}'\}}}{\sigma_{\{\mathcal{S}'\}}} \tag{3}$$

其中 $\mu_{\{\mathcal{S}'\}}$ 和 $\sigma_{\{\mathcal{S}'\}}$ 是根据多样化状态 $\mathcal{S}'$集合中的奖励计算得出的。这种多样化促进了更充分的探索，进而提高了奖励分布$\sigma_{\{\mathcal{S}'\}}$ 的方差，因为不同的变异 $\Delta s_{\text{mutation}_i}$ 会暴露出不同的错误模式。最终，这将产生信息量更丰富的优势信号，反映模型性能的不同方面，进而使大语言模型训练更加高效、有效。

给定训练样本的最终 GRPO-SMu 目标函数为：

$$L(s'_i, a_i, \theta_t, \theta) = \text{clip}\left(\frac{\pi_\theta(a_i|s_i)}{\pi_{\theta_t}(a_i|s_i)}, A^{\pi_{\theta_t}}(s'_i, a_i)\right) - \beta D_{KL}(\pi_{\theta_t}||\pi_\theta)$$

表 I 功能等效代码生成类别

功能等价代码类别	
风格类	不同的命名、格式和语句重组方式
架构类	行为级与结构级、有限状态机编码、分解
实现类	始终块/case/if 结构调整、阻塞赋值选择、重构
优化类	资源与时序权衡、强度削减
结构类	模块层次结构、信号因式分解、布局重组

其中 D_{KL} 为 KL 散度损失，裁剪函数定义如下：

$$\text{clip}(r, A) = \begin{cases} \min(r, 1+\epsilon) \cdot A & \text{if } A > 0 \\ \max(r, 1-\epsilon) \cdot A & \text{if } A \le 0 \end{cases}$$

C. 强化学习训练数据集：基于树的变异策略

GRPO-SMu 需要为每段寄存器传输级代码生成多个变异变体，以创建多样化的状态。为此，我们提出了一种基于树的方法，其生成的变异比现有的数据集整理方法更全面 [31], [36]。

图 6 （中右）展示了我们的分支树构建策略。从 [11], ScaleRTL 数据集获取的黄金 RTL代码 $n = 5$ 出发，我们为每个规格说明生成 I 个功能等价的 RTL实现（表中的多个根节点）。对于每个根节点，我们引入 $n_1 = 3$ 来自 14 个高层错误类别的第一级变异（表 II）。随后，我们使用由 CVDP基准测试故障分析构建的、包含 71 个细粒度算子的扩展算子集，继续对每个错误变体进行变异 [42]，从而产生更深层的分支。这将线性过程（黄金 →加少量随机错误）转换为分支树：黄金 → n等价变体→ $n \times n_1$、第一级错误 →，依此类推 .....，从而提高变异空间的覆盖率。

表 II 用于构建分支树的变异类型

第一级变异类别		
逻辑门互换	always 块类型	阻塞与非阻塞
信号反相	条件边界	索引偏差一位
运算符互换	时序结构	复位逻辑错误
敏感列表	竞争条件	有限状态机状态损坏
位宽不匹配	行内变量交换	

表 III 通过改进的优势计算缓解稀疏奖励

Case	奖励	标准优势值
情况 1	[1.0, 1.0, 1.0, 1.0, 1.0, 2.0, 1.0, 1.0]	[−0.38, −0.38, −0.38, −0.38, −0.38, 1.0, −0.38, −0.38]
情况 2	[1.0, 1.0, 1.0, 1.0, 1.0, 2.0, 0.8, 1.0]	[−0.29, −0.29, −0.29, −0.29, −0.29, 1.36, −0.91, −0.29]

注： 情况 1 表示稀疏奖励场景（占训练样本的 ∼30%），在这种情况下，我们的修改能够为高奖励样本提供更强的学习信号。情况 1 中奖励为 2.0 时的固定优势：2.83。

a) 功能验证： 所有等价变体和变异都通过随机测试生成器进行验证，该生成器为每个 RTL设计生成 100 万个测试用例[1] 。对于每个候选项（等价变体或变异），我们在黄金（参考）实现和候选实现上分别运行这些测试用例，并统计可观测输出中的不匹配情况。根据不匹配统计：如果不存在不匹配，则该变异没有功能影响，并将其加入干净 _ 代码；否则，该变异在功能上存在差异，并将其加入 变异_代码。这种功能等价性检查方法确保所有转换都经过功能验证，而不是依赖纯粹的语法标准。

b) 数据统计：我们的已验证语料库包含：基础数据集 = 2,452 个寄存器传输级代码；等价代码 + 基础代码 = 14,061 个代码；单级变异 = 17,926 个代码；1,204 个样本用于二级变异；22,051 个双级变异。GRPO-SMu 训练使用了 1,318 个样本，而最终评估则筛选了 500 个样本。

D. 奖励建模

为了区分能够捕获更多变异的测试计划，我们创建了一个采用 0–3 分制的精细奖励模型：

$$R = r_o + w_m r_m + r_j + r_c \tag{4}$$

其中：

- $r_o \in \{0, 1\}$ 衡量基本功能（M1：黄金代码通过）
- $r_m \in [0, 1]$ 衡量错误检测能力（M2：变异失败率），其中 $r_m = \frac{\#\text{失败的变异数量}}{\#\text{变异总数}}$
- $r_j \in \{0, 0.8\}$ 表示外部大语言模型质量评估
- $r_c \in \{0, 0.2\}$ 用于防止非英文字符生成

仅当满足 $r_o = 1$ 时，权重 w_m 才设为 1，否则设为 $w_m = 0$，从而确保只有在实现基本功能后才会给予变异检测奖励，以防奖励投机。

E. 训练

我们的训练流程将寄存器传输级描述和变异代码作为输入。小语言模型从我们的监督微调检查点初始化后，会生成推理内容和单元测试计划，以区分黄金实现与变异实现。为了评估测试计划质量，我们将生成的测试计划与随机测试生成器生成的测试平台模块结合起来，并用其提示 LLaMa-3.1-405B，由后者执行测试并报告通过/失败状态。至关重要的是，这种双组件分离可防止奖励作弊：推理模型生成测试计划时并不知道评估大语言模型所用的具体提示或奖励结构，从而维持输出质量。

训练挑战： 针对我们任务的训练中，最大的挑战之一是奖励稀疏性。事实上，在最初的 100 个训练步骤中，只有 ∼ 30% 的生成获得了奖励；其中仅有一次生成获得奖励 > 1，而 $G - 1$ 为 1。我们通过以下两种方式应对这一问题。

a) 禁用令牌级损失： 使用 DAPO [41]，中提出的令牌级损失，会导致大语言模型生成的大多是信息量不足的内容，同时保持较低的令牌数，并偶尔生成高奖励输出，以最大化组内平均表现。关闭令牌级损失有助于缓解这一问题。

b) 改进优势计算： 在稀疏奖励场景下，留一基线对高奖励样本使用标准差 1，从而导致学习信号较弱（如表 III 所示）。为应对此类情况，我们修改了优势计算，改为使用整个总体的标准差（而非留一基线）。这使训练信号放大了超过 2×。

V. 实验与结果

A. 基线与实现细节

对于基线，我们考虑了多种最先进的通用大语言模型，例如 DeepSeek-R1 、 Claude-4.0-Sonnet 和 LLaMa-3.1-405B。我们还使用了 ScaleRTL-32B [11], ，这是一种针对寄存器传输级特定任务进行微调的大语言模型。对于小语言模型，我们将基础版 DeepSeek-R1-distill-Qwen-7B 模型、基础版监督微调模型和基础版 GRPO 模型作为基线。所有模型的推理温度均设为 0.7。对于监督微调，我们将学习率设为 2e-6，并分别将全局批次大小和微批次大小设为 32 和 2。对于 GRPO，我们使用 64 的批次大小，每步 16 个样本、每个样本生成 8 个结果，学习率为 5e-7，温度为 1，KL 系数为 0.01。

B. *GRPO* 与 *GRPO-SMu* 对比

首先，我们将提出的 GRPO-SMu 方法与传统的 GRPO 直接进行比较。图 7(a) 显示，与 GRPO 相比，GRPO-SMu 的平均奖励提高了 10%。黄金通过率的提升也印证了奖励的改善（∼15% 更高），如图 7(b) 所示。最后，在图7(c) 中，我们展示了 GRPO-SMu 如何提高训练组中至少有一个测试计划在黄金 RTL 上通过 8 次生成的比例，这意味着奖励多样性更高，探索效果更好。

[1]我们仅使用此随机测试生成器来验证训练样本中的变异效果，不会影响大语言模型的推理延迟。

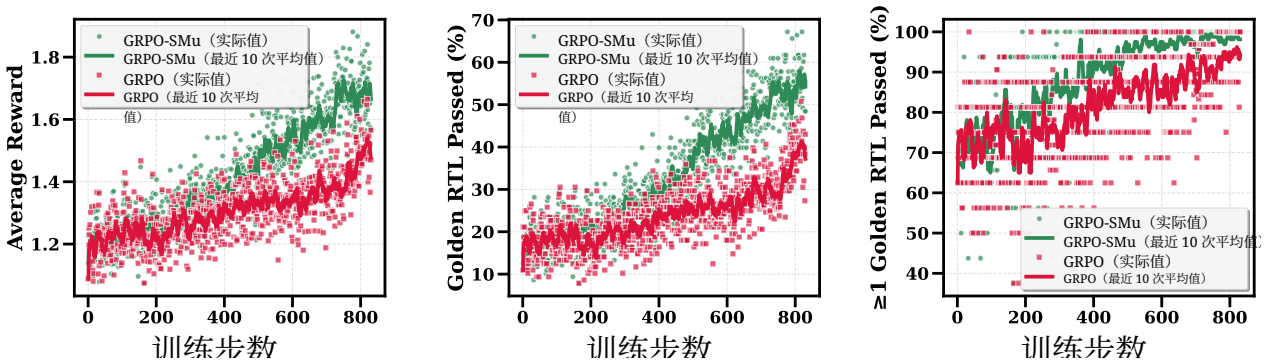

图 7。（a）传统 GRPO 与 GRPO-SMu 的训练奖励对比，显示在相同步数下训练性能提升了 10%；（b）黄金通过率提高；（c）至少有一次生成成功的样本百分比更高。

C. 主要结果

这里，我们给出了所有模型在包含 1500 个测试样本的主要测试设置下的基准测试结果（第 III-D 节）。具体而言，给定变异后的 RTL 代码和自然语言描述后，大语言模型需要生成单元测试计划。随后，由 LLaMa-3.1-405B 对这些测试计划进行处理，以生成测试平台。最后，我们报告两个关键指标：黄金 RTL 通过率和变异检测率。结果见表 IV。

表 IV 黄金测试生成与变异代码检测的模型性能比较

模型	黄金通过率 (%)	变异检测率 (%)	变化得分
小型语言模型（*SLM*）			
DeepSeek-R1-distill-Qwen-7B	15.7	6.7	基线
基础监督微调	18.2	7.5	+2.5
基础群体相对策略优化	27.2	10.5	+11.5
base-GRPO-SMu（我们的方法）	**33.3**	**13.9**	**+17.6**
大语言模型（*LLM*）			
DeepSeek-R1	21.7	10.1	+6.0
ScaleRTL-32B [11]	21.6	8.9	+5.9
Claude-4.0-Sonnet	20.6	10.6	+4.9
LLaMa-3.1-405B	16.3	6.4	+0.6
LLaMa-3.1-405B-with-golden	20.2	8.7	+4.5

注：变化分数表示相对于基础小语言模型 DeepSeek-R1-distill-Qwen-7B，在黄金通过率指标上的百分点提升。

实验结果表明，我们提出的 GRPO-SMu 方法表现最佳，黄金测试通过率达到 33.3%，变异检测率达到 13.9%。相较于未经训练的基础模型 DeepSeek-R1-distill-Qwen-7B，GRPO-SMu 的表现提升超过 2×，最高提升 17.6%。从 SFT 到 GRPO 再到 GRPO-SMu 逐步训练时，我们观察到的提升分别为 2.5%、8% 和 6%，这凸显了微调对于提升大语言模型在测试计划生成方面的推理能力的重要性。

在所有大语言模型中，Deepseek-R1 和 ScaleRTL-32B 的表现最佳，正确版本通过率为 21.6%–21.7%。然而，GRPO-SMu 7B 模型的正确版本通过率比这两种最先进模型高出 11–12%，进一步证明了我们专门训练方法的有效性。

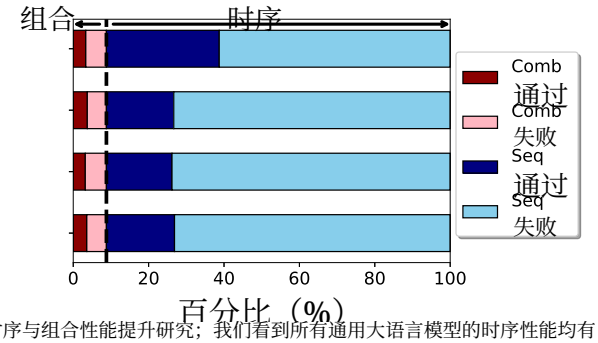

图 8. 时序与组合性能提升研究；我们看到所有通用大语言模型的时序性能均有 ∼ 1.6× 提升。

D. 消融两阶段方法

为验证我们提出的两阶段流程，我们将其与针对两个大语言模型 LLaMa-3.1-405B 和 Claude-4.0-Sonnet 的单阶段（独立）基线进行比较。对于独立提示，我们使用与两阶段方法相同的提示，但去掉了有关推理和单元测试计划的内容。如表 V所示，对于这两个大语言模型，我们的两阶段方法始终优于对应的单阶段方法，正确版本通过率高出 10–12%。尽管 Claude-4.0-Sonnet 在第二阶段表现更好，但由于 API 成本较高，我们选择 LLaMa-3.1-405B 用于第二阶段。最后，我们还表明，两阶段的 7B GRPO-SMu 优于所有这些大语言模型组合，是最佳的测试计划生成模型。

表 V 两阶段架构的合理性说明

模型配置	黄金通过率提升 相对于基线的通过率（%）	
LLaMa-3.1-405B（单独使用）	5.7	–
LLaMa-3.1-405B → LLaMa-3.1-405B	16.3	+10.6
Claude-4.0-Sonnet（单独使用）	18.9	+13.2
Claude-4.0-Sonnet → LLaMa-3.1-405B	20.6	+14.9
Claude-4.0-Sonnet → Claude-4.0-Sonnet	30.2	+24.5
base-GRPO-SMu →LLaMa-3.1-405B（我们的方法）	**33.3**	**+27.6**

注：所有改进均相对于 LLaMa-3.1-405B 独立基线计算。两阶段表示：第二阶段1 → 第二阶段。

E. 电路类型错误分析

我们通过 4 个大语言模型的多数投票，将基准测试中的每个样本分类为组合逻辑或时序逻辑，以此分析测试基准。在 500 个测试样本中，44 个（8.8%）属于组合逻辑，其余属于时序逻辑，这验证了我们聚焦于时序调试的合理性。我们的训练数据也呈现类似的分布。图 8 中的性能分析表明，所有模型在较简单的组合电路上表现更好。然而，对于时序电路，相较于所有其他模型，我们观察到 GRPO-SMu 带来了近 2× 的提升，这凸显了其性能提升的来源。

VI. 结论与未来工作

本文提出了一种新的任务分解方法，并通过将语言化测试计划作为子单元验证的中间表示，构建了一个两阶段框架。这类测试计划不仅提升了性能，也增强了人在回路控制能力，有助于构建更好的测试计划。我们深入评估和研究发现，尽管最先进的大语言模型具备强大的推理和代码生成能力，但在我们的硬件验证任务中，其成功率仅为15%–22%。为进一步提升其表现，我们提出了一种增强型强化学习技术GRPO-SMu，通过提高输入变异多样性来增强探索能力。我们还为GRPO-SMu的训练数据生成设计了一种新颖的基于树的变异策略。我们展示了如何在一个小型7B大语言模型上应用GRPO-SMu，使其黄金测试通过率达到最高的33.3%，优于其他规模更大且经过微调的大语言模型。尽管大语言模型距离可投入生产的验证仍有很大差距，但我们的工作为构建自主调试框架奠定了重要基础，有望取代手动创建测试并提高生产效率。未来工作可以利用我们的框架和经验，开发可扩展、可部署的系统与智能体工作流，从根本上重塑硬件验证，助力更快推进下一代半导体设计。

参考文献

[1] Janick Bergeron.《编写测试平台：*HDL*模型的功能验证》. 施普林格科学与商业媒体，第2版，2003年。 [2] Anmol Mathur 和 Venkat Krishnaswamy. 系统级模型和寄存器传输级中的面向验证设计。载于第*44*届设计自动化大会论文集，第193–198页，2007年。 [3] Aharon Aharon 等。使用动态偏置伪随机测试程序生成器验证 IBM RISC System/6000。《*IBM*系统期刊》，30(4):527–538，1991年。 [4] Jun Yuan 等。使用二叉决策图对仿真中的设计约束和偏置进行建模。载于*1999*年*IEEE/ACM*计算机辅助设计国际会议论文集（技术论文摘要）（目录号*99CH37051*），第584–589页。IEEE，1999年。 [5] Bruce Wile 等。全面功能验证：完整的产业周期。摩根考夫曼，2005年。 [6] Hammond Pearce、Benjamin Tan 和 Ramesh Karri。DAVE：从英文自动生成 Verilog。载于*2020*年*ACM/IEEE*计算机辅助设计机器学习研讨会论文集，第27–32页，2020年。 [7] Shailja Thakur 等。VeriGen：用于 Verilog 代码生成的大语言模型。《*ACM*电子系统设计自动化汇刊》，29(3):1–31，2024年。 [8] Shang Liu 等。RTLCoder：凭借我们的开源数据集和轻量级方案，在设计寄存器传输级生成方面超越 GPT-3.5。载于*2024*年*IEEE*大语言模型辅助设计国际研讨会。IEEE，2024年。 [9] Zehua Pei 等。BetterV：通过判别式引导实现受控 Verilog 生成。*arXiv*预印本 *arXiv:2402.03375*，2024年。

[10] Yi Liu 等。Deeprtl：通过统一表示模型连接 Verilog 理解与生成。arXiv 预印本 arXiv:2502 15832，2025。。[11] Chenhui Deng 等。Scalertl：利用推理数据和测试时计算扩展大语言模型，实现准确的 RTL 代码生成。arXiv 预印本 arXiv:2506.05566，2025。[12] Ruidi Qiu 等。CorrectBench：面向硬件描述语言设计，使用大语言模型进行功能自我纠正的自动测试平台生成。载于欧洲设计、自动化与测试会议（DATE），2025。[13] Yujie Zhao 等。Pro-v：用于自动 RTL 验证的高效程序生成多智能体系统。arXiv 预印本 arXiv:2506 12200，2025。。[14] Zhihong Shao 等。Deepseekmath：突破开放语言模型数学推理能力的极限。arXiv 预印本 arXiv:2402 03300，2024。。[15] Mingjie Liu、Nathaniel Pinckney、Brucek Khailany 和 Haoxing Ren。Verilogeval：评估大语言模型的 Verilog 代码生成能力。载于2023年IEEE/ACM国际计算机辅助设计会议（ICCAD），第1–8页。IEEE，2023。[16] Yao Lu 等。Rtllm：用于大语言模型设计 RTL 生成的开源基准。载于2024年第29届亚洲及南太平洋设计自动化会议（ASP-DAC），第722–727页。IEEE，2024。[17] Fan Cui 等。Origen：通过代码到代码增强和自我反思提升 RTL 代码生成。载于第43届IEEE/ACM国际计算机辅助设计会议论文集，第1–9页，2024。[18] Mingzhe Gao 等。Autovcoder：使用大语言模型进行自动化 Verilog 代码生成的系统化框架。arXiv 预印本 arXiv:2407 18333，2024。。[19] Humza Sami 等。Aivril：将验证融入流程的人工智能驱动 RTL 生成。arXiv 预印本 arXiv:2409.11411，2024。[20] Zhendong Mi 等。Promptv：利用大语言模型驱动的多智能体提示实现高质量 Verilog 生成。arXiv 预印本 arXiv:2412.11014，2024。[21] Chia-Tung Ho、Haoxing Ren 和 Brucek Khailany。Verilogcoder：采用基于图的规划和基于抽象语法树（AST）的波形跟踪工具的自主 Verilog 编码智能体。载于AAAI人工智能会议论文集，第39卷，第300–307页，2025。[22] Zhuorui Zhao 等。Vrank：通过自洽性提升大语言模型的 Verilog 代码生成能力。arXiv 预印本 arXiv:2502.00028，2025。[23] Yujie Zhao 等。Mage：用于自动化 RTL 代码生成的多智能体引擎。arXiv 预印本 arXiv:2412.07822，2024。[24] Zhirong Chen 等。Chipseek-r1：通过分层奖励驱动强化学习，借助大语言模型生成超越人类水平的 RTL。arXiv 预印本 arXiv:2507.04736，2025。[25] Yun-Da Tsai、Mingjie Liu 和 Haoxing Ren。Rtlfixer：使用大语言模型自动修复 RTL 语法错误。arXiv 预印本 arXiv:2311.16543，2023。[26] Xufeng Yao 等。Hdldebugger：利用大语言模型简化硬件描述语言调试。arXiv 预印本 arXiv:2403.11671，2024。[27] Shailja Thakur 等。Autochip：利用大语言模型反馈实现硬件描述语言生成自动化。arXiv 预印本 arXiv:2311.04887，2023。[28] Ke Xu 等。Meic：使用大语言模型重新思考 RTL 调试自动化。arXiv 预印本 arXiv:2405.06840，2024。[29] Hanxian Huang 等。迈向大语言模型驱动的 Verilog RTL 助手：自我验证与自我纠正。arXiv 预印本 arXiv:2406.00115，2024。[30] Zixi Zhang 等。Llm4dv：使用大语言模型生成硬件测试激励。载于2025年IEEE第33届年度现场可编程定制计算机国际研讨会（FCCM），第133–137页。IEEE，2025。[31] Ning Wang 等。Veridebug：通过对比嵌入和引导式纠正实现统一的 Verilog 调试大语言模型。arXiv 预印本 arXiv:2504.19099，2025。[32] R Ma 等。Verilogreader：大语言模型辅助硬件测试生成。载于2024年IEEE大语言模型辅助设计研讨会（LAD），第1–5页。IEEE，2024。[33] R Qiu 等。Autobench：使用大语言模型为硬件描述语言设计自动生成和评估测试平台。载于2024年ACM/IEEE计算机辅助设计机器学习国际研讨会论文集，第1–10页，2024–。[34] R Qiu 等。Correctbench：面向硬件描述语言设计，使用大语言模型进行功能自我纠正的自动测试平台生成。arXiv 预印本 arXiv:2411.08510，2024。

[35] Yujie Zhao 等。Pro-v：用于自动 RTL 验证的高效程序生成多智能体系统。*arXiv* 预印本 *arXiv:2506 12200*，2025。。[36] Surya J asper 等。BugGen：用于真实 RTL 错误合成的自我纠正多智能体大语言模型流水线。*arXiv* 预印本 *arXiv:2506.10501*，2025。[37] Zeng Wang 等。Vericontaminated：评估大语言模型驱动的 Verilog 编码中的数据污染。*arXiv* 预印本 *arXiv:2503.13572*，2025。[38] Daya Guo 等。DeepSeek-R1：通过强化学习激发大语言模型的推理能力。*arXiv* 预印本 *arXiv:2501.12948*，2025。[39] Mingjie Liu 等。Prorl：延长强化学习拓展大语言模型的推理边界。*arXiv* 预印本 *arXiv:2505.24864*，2025。[40] Tushar Khot 等。分解式提示：解决复杂任务的模块化方法。*arXiv* 预印本 *arXiv:2210.02406*，2022。[41] Qiying Yu 等。DAPO：可大规模扩展的开源大语言模型强化学习系统。*arXiv* 预印本 *arXiv:2503.14476*，2025。[42] Nathaniel Pinckney 等。综合 Verilog 设计问题：用于评估大语言模型和智能体在 RTL 设计与验证方面能力的下一代基准数据集。*arXiv* 预印本 *arXiv:2506.14074*，2025。