# 基于自注意力深度学习的硬件代码缺陷定位方法

刘振磊 胡 健
（重庆大学大数据与软件学院 重庆 400044）

**摘 要** 硬件代码缺陷定位是实现硬件设计可靠性、降低设计开发成本和提高硬件设计质量的关键环节。然而，现有缺陷定位方法存在输出与执行信息不匹配、代码覆盖矩阵语义信息不足、可疑值计算方法过于简单等问题，导致缺陷定位精度受限。为了解决这些问题，我们提出一种基于自注意力深度学习的硬件代码缺陷定位方法。该方法利用VCD（Value Change Dump）对比技术，匹配特定时钟周期内硬件程序的仿真结果与语句覆盖信息，构建精准的覆盖矩阵。其次，通过动态切片技术增强语义信息，保留与缺陷相关的语句，缩小代码搜索范围。最后，使用自注意力深度神经网络学习语句与缺陷之间的复杂映射关系，计算每个语句的可疑值，实现高精度的硬件代码缺陷定位。实验结果表明，该方法在缺陷定位效果上优于最新的缺陷定位方法。在Top-1指标下，本文方法的缺陷定位有效性比最新的缺陷定位方法增长了50%至200%；在MFR指标下，本文方法的缺陷定位有效性比最新的缺陷定位方法的有效性下降了51%至59%，表明该方法能更快更精确地识别缺陷位置，从而有效提高硬件设计的验证效率。

**关键词** 缺陷定位；自注意力深度学习；程序切片；覆盖矩阵；可疑值
**中图法分类号** TP391.41 **DOI号** 10.11897/SP.J.1016.2025.02508

# Self-Attention Deep Learning-Based Fault Localization for Hardware Code

LIU Zhen-Lei HU Jian
（School of Big Data and Software Engineering, Chongqing University, Chongqing 400044）

**Abstract** Hardware code fault localization plays a critical role in ensuring the reliability of hardware design, reducing development costs, and enhancing design quality. However, current defect localization methods face several challenges that limit their effectiveness. There are often mismatches between output and execution information, and the code coverage matrix used in these methods lacks sufficient semantic information. Additionally, the methods for calculating suspiciousness values are overly simplistic, all of which result in low accuracy of fault localization. To address these issues, this paper presents an advanced hardware code fault localization method based on self-attention deep learning. Initially, the method employs VCD (Value Change Dump) comparison techniques. By aligning the simulation results of the hardware program with executed statements within specific clock cycles, a precise coverage matrix is constructed. This step ensures that the information used for defect localization is accurate and reliable. Subsequently, dynamic slicing techniques are applied to bolster semantic information. This process retains statements that are potentially related to faults and narrows down the scope of code that needs to be searched. By focusing on the most relevant parts of the code, the efficiency of defect localization is significantly improved. Finally, a self-attention deep neural network is utilized to

收稿日期：2025-04-12；在线发布日期：2025-07-10。本课题得到国家自然科学基金片上系统高层等价性检验理论与关键技术（No. 61902421）资助。**刘振磊**，硕士研究生，主要研究领域为硬件代码缺陷定位。E-mail：zhenleiliu@stu.cqu.edu.cn。**胡 健**（通信作者），博士，副教授，博士生导师，中国计算机学会（CCF）会员，主要研究领域为电子设计自动化、智能软件工程。E-mail：jianhu@cqu.edu.cn。

learn the intricate mapping relationships between statements and faults. This allows for the calculation of a suspiciousness value for each statement. With these suspiciousness values, high-precision hardware code fault localization can be achieved. The experimental results demonstrate the superiority of the proposed method in terms of fault localization effectiveness when compared to the latest fault localization methods. Under the Top-1 metric, the fault localization effectiveness of our method surpasses that of the state-of-the-art methods by 50% to 200%. However, under the MFR metric, the fault localization effectiveness of our method sees a decline of 51% to 59% compared to the state-of-the-art methods. These results indicate that our method is capable of more rapidly and accurately identifying defect locations, thereby effectively enhancing the verification efficiency of hardware design.

**Keywords** fault localization; self-attention deep learning; program slicing; coverage matrix; suspiciousness

## 1 引 言

随着现代集成电路和硬件系统的规模不断扩大，硬件设计的复杂性显著增加[1]。对于大规模系统，代码缺陷不仅会影响整个集成电路的性能，还可能导致电路在实际运行中出现严重故障，造成巨大的经济损失[2]。因此，及时和精准地定位和修复硬件程序中的缺陷能够有效保障整个硬件设计的质量。在实际的硬件设计开发中，验证和调试阶段通常占据整个开发周期的70%以上，其中大部分时间用于缺陷的定位和修复[3-4]。因此，提高缺陷定位的有效性可以显著减少调试阶段的时间和资源开销，提升整体设计的开发效率。

目前，硬件代码缺陷定位方法主要包括静态缺陷定位方法和动态缺陷定位方法两大类。静态缺陷定位方法主要通过从代码中自动生成诊断模型，遍历模型的状态空间，推导可能的缺陷位置[5]。静态缺陷定位方法虽然能够定位硬件代码中的缺陷并且不需要测试用例，但该类方法需要进行形式化建模，存在状态空间爆炸等问题，无法处理大规模设计[6-7]。针对静态缺陷定位方法存在的问题，研究者们提出了动态缺陷定位方法。文献[8]直接分析与错误变量相关的仿真轨迹来定位硬件设计的缺陷。然而，这种依赖于仿真轨迹的方法并未充分利用动态仿真信息，而是通过重复的因果分析来定位缺陷，过程十分繁琐。CirFix[9]对输出结果进行采样并收集仿真过程中观察到的所有不匹配信号。利用静态切片技术构建可疑语句集合，通过对可疑语句的分析进行缺陷定位。但对于大规模程序，静态切片体积较大，缺陷定位精度不高。Tarsel[10]通过执行测试用例，生成代码的覆盖信息和执行结果，统计各个语句被成功和失败测试用例执行的情况，利用可疑值评估公式计算各个语句的可疑值。动态缺陷定位方法通过统计分析代码的覆盖信息进行缺陷定位，定位效率高，可扩展性强。然而，现有动态缺陷定位方法仍然存在以下三个问题：

(1) 硬件代码的输出结果存在时序延迟，导致语句执行信息与输出结果不匹配，产生了不准确的代码覆盖矩阵和结果向量，限制了动态缺陷定位方法的定位精度。

(2) 硬件设计中存在大量与缺陷无关的语句，导致代码覆盖矩阵的语义表达能力不足，缺乏可疑语句与程序失败的直接映射关系，限制了动态缺陷定位方法的有效性。

(3) 现有动态缺陷定位方法只使用简单的统计分析方法计算各个语句的可疑值，缺陷定位能力弱，定位有效性不足。

为了解决这些问题，本文提出了基于自注意力深度学习的硬件代码缺陷定位方法Sepal(Self-Attention Deep Learning-based Fault localization, Sepal)。该方法首先使用VCD对比技术将特定时钟周期内的程序执行结果与相应的执行语句进行匹配，构建精确的代码覆盖矩阵和结果向量。然后，使用动态切片技术[11-12]增强语义信息，构建可疑语句与失败测试用例的映射关系，缩小代码搜索空间。最后，使用自注意力深度神经网络学习语句与测试结果的复杂映射关系，实现高精度的代码缺陷定位。

综上所述，本文的主要贡献包括：

(1) 提出基于自注意力深度学习的硬件代码缺陷定位方法Sepal。该方法通过VCD对比获取精确的代码覆盖矩阵，使用动态切片增强覆盖矩阵语义

信息，利用自注意力深度神经网络计算语句的可疑值，实现高精度的缺陷定位。

（2）对四种深度神经网络进行了实验对比，包括自注意力多层感知机[13]（SAMLP，Self-Attentioon MLP）、自注意力卷积神经网络[14]（SACNN，Self-Attention CNN）、自注意力循环神经网络[15]（SARNN，Self-Attention RNN）和Transformer[16]。实验结果表明，SAMLP自注意力神经网络定位有效性更高，更适合缺陷定位。

（3）本文方法与目前最先进的动态缺陷定位方法Cirfix[9]和Tarsel[10]进行了实验对比。实验结果表明，在相同的测试指标下，Sepal能够定位更多的代码缺陷，缺陷定位有效性更高。

本文的结构如下：第2节概述与本文方法相关的研究背景；第3节详细介绍了Sepal方法；第4节给出了实验结果并进行了结果分析；第5节探讨了本文方法面临的有效性威胁；第6节分析了相关工作；第7节进行了总结，并提出了未来的工作。

## 2 相关背景

### 2.1 动态缺陷定位

动态缺陷定位技术是一种通过分析和统计程序执行的覆盖信息和测试结果来进行缺陷定位的方法[17-19]。其原理如下：

考虑一个由$N$个语句$\{s_1, s_2, \cdots, s_N\}$组成的硬件设计$Hd$和在一个由$M$个测试用例$\{t_1, t_2, \cdots, t_M\}$组成的激励文件$Tb$（测试用例集中至少包含一个失败的测试用例），运行程序P并获取每个测试用例的代码覆盖信息和测试结果，组成一个维度为$M\times N$的代码覆盖矩阵（图1），其中$x_{ij}=1$表示测试用例$t_i$覆盖了语句$s_j$，$x_{ij}=0$表示测试用例$t_i$未覆盖语句$s_j$。测试用例的结果由结果向量记录，其中$e_i=1$表示测试用例$t_i$的输出与预期输出不一致，视为失败测试用例，$e_i=0$表示测试用例$t_i$的输出符合预期输出，视为成功测试用例。

代码覆盖矩阵和结果向量能够反映出程序运行过程中的语句覆盖情况（执行或未执行）和测试用例的结果（成功或失败）。通过对其进行统计分析，可以为每个语句定义四个参数，以此构建不同类型的可疑值计算公式，计算各个语句的可疑值。四个参数的定义如下：

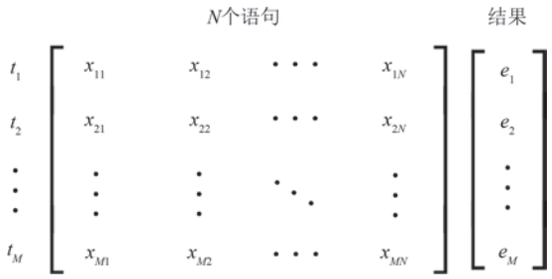


图1 代码覆盖矩阵和结果向量

$$\begin{cases} a_{np}(s_j)=\left|\{i|x_{ij}=0\wedge e_i=0\}\right| \\ a_{nf}(s_j)=\left|\{i|x_{ij}=0\wedge e_i=1\}\right| \\ a_{ep}(s_j)=\left|\{i|x_{ij}=1\wedge e_i=0\}\right| \\ a_{ef}(s_j)=\left|\{i|x_{ij}=1\wedge e_i=1\}\right| \end{cases} \tag{1}$$

$a_{np}(s_j)$表示测试用例集中未执行语句$s_j$的成功测试用例数，$a_{nf}(s_j)$表示测试用例集中未执行语句$s_j$的失败测试用例数，$a_{ep}(s_j)$表示测试用例集中执行语句$s_j$的成功测试用例数，$a_{ef}(s_j)$表示测试用例集中执行语句$s_j$的失败测试用例数，动态缺陷定位方法基于这四个参数构建可疑值评估公式并计算每个语句的可疑值。表1给出了部分可疑值评估公式。

**表1 可疑值评估公式**

| 名称 | 公式 |
|---|---|
| Optimal | $\begin{cases}-1, & a_{nf}>0\\ a_{np}, & a_{nf}\leqslant 0\end{cases}$ |
| Wong2 | $a_{ef}-a_{ep}$ |
| Wong1 | $a_{ef}$ |
| Rogot1 | $\frac{1}{2}\times\left(\frac{a_{ef}}{2\times a_{ef}+a_{nf}+a_{ep}}+\frac{a_{np}}{2\times a_{np}+a_{nf}+a_{ep}}\right)$ |
| Ochiai | $\frac{a_{ef}}{\sqrt{a_{ef}+a_{nf}}+\sqrt{a_{ef}+a_{ep}}}$ |
| Dstar | $\frac{a_{ef}^{*}}{a_{nf}+a_{ep}}$ |

注：文献[20]建议Dstar公式中*的值取2。

语句的可疑值越高，代表该语句存在缺陷的可能性越大。按照可疑值大小对语句进行排序，生成可疑语句列表，开发人员能够优先检查可疑值较高的语句，快速定位程序缺陷位置。

### 2.2 自注意力深度神经网络

深度神经网络是一种通过多层非线性变换来学习数据复杂特征的机器学习模型。将自注意力机制引入深度神经网络能够显著增强模型捕捉全局依赖关系的能力。自注意力深度神经网络由输入层、多个自注意力层和输出层组成，输入层接收序列数据，

自注意力层负责特征交互，输出层生成预测结果。

自注意力机制通过计算查询($Q$)、键($K$)和值($V$)之间的关系来动态捕捉序列中的依赖关系。其核心公式为：

$$\text{Attention}(Q,K,V)=\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (2)$$

其中，$d_k$是键的维度。该公式通过计算查询和键之间的点积生成注意力分数，并利用这些分数对值进行加权求和，从而实现对序列中不同位置的动态关注。

在训练过程中，反向传播算法用于计算损失函数$L$相对于网络权重的梯度。对于隐藏层和输出层，误差项分别定义为公式3和4，其中，$f'(z^{(l)})$是激活函数的导数。

$$\delta^{(l)}=\left(\left(\delta^{(l+1)}\cdot W^{(l+1)}\right)\odot f'(z^{(l)})\right) \quad (3)$$

$$\delta^{(L)}=\frac{\partial L}{\partial \text{Output}}\odot f'(z^{(L)}) \quad (4)$$

权重更新通过梯度下降法实现，如公式5所示，其中$\eta$是学习率，控制更新步长。在自注意力层中，查询、键和值的权重矩阵$W(Q)$、$W(K)$和$W(V)$分别通过对应的梯度进行更新。

$$W^{(l)}\leftarrow W^{(l)}-\eta\cdot\frac{\partial L}{\partial W^{(l)}} \quad (5)$$

自注意力深度神经网络通过动态计算元素间相关性权重，使模型能够自主关注输入序列中的重要区域并捕捉长距离依赖关系，因此在计算机视觉[21]、语音识别[22]和自然语言处理[16]等多个领域表现优异。

## 3　关键实现技术

图2描述了Sepal的工作流程，包括代码覆盖矩阵生成、强语义上下文构建和可疑值评估三个阶段。动态覆盖矩阵生成阶段对硬件设计代码进行仿真并获取代码覆盖信息，使用VCD对比技术来精确匹配执行路径和执行结果，构建精确的代码覆盖矩阵。强语义上下文构建阶段采用动态切片技术保留与错误输出相关的代码，过滤不相关语句，缩小缺陷定位的搜索范围。可疑值评估阶段首先构建自注意力深度神经网络模型并使用优化后的覆盖矩阵和结果向量训练模型，然后构建虚拟覆盖矩阵并输入训练后的模型，模型输出每条语句的可疑值。

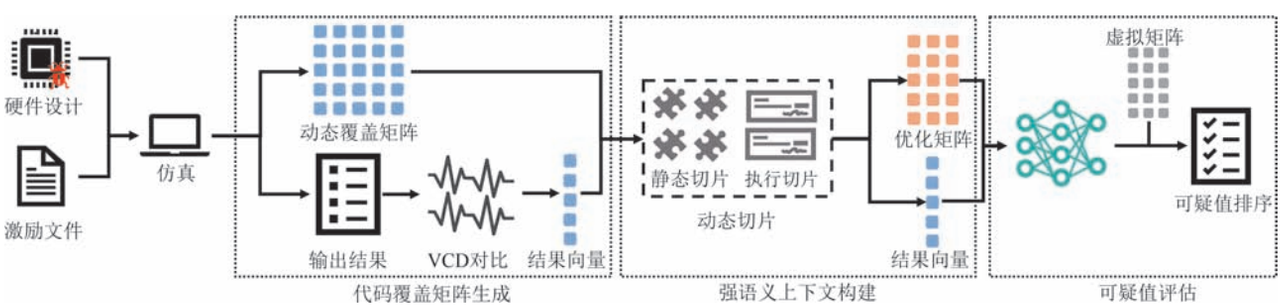


图2　Sepal工作流程

### 3.1　代码覆盖矩阵生成

Sepal使用ModelSim对被测硬件设计进行模拟仿真。将包含$N$条语句的硬件设计$Hd$和包含$M$个测试用例的激励文件$Tb$导入ModelSim。运行TCL(Tool Command Language)脚本自动执行程序，生成如图1所示的代码覆盖矩阵。

在结果向量的生成过程中，现有的动态缺陷定位方法会通过比较仿真输出文件和正确模型的输出文件来确定每个测试用例的测试结果。但由于硬件语言固有的时序延迟，输出文件记录的信号值无法实时反映信号的实际状态，导致仿真输出与真实电路行为不匹配，产生不精确的覆盖矩阵和结果向量的映射关系，降低缺陷定位的准确性。

为了解决这个问题，Sepal采用VCD对比方法来确定每一个时钟周期的执行结果。图3展示了一个带有缺陷的Verilog计数器程序，该程序的第8行为缺陷语句。图4描述了输出信号$overflow_out_{buggy}$和正确输出信号$overflow_out$的波形比较。在时钟周期为195 ns时，$overflow_out_{buggy}$的值是0，而正确输出信号$overflow_out$此时由0跳变为1。现有动态缺陷定位方法的仿真输出文件由于时序延迟会记录此刻$overflow_out$的值为0，判定该时刻的测试用例结果为正确，导致代码覆盖信息与执行结果不匹配，降低缺陷定位的精度。Sepal采用VCD对比技

术，通过解析VCD并自动对比信号的值和时序，准确地判定实际输出信号 $overflow\_out_{buggy}$ 与 $overflow\_out$ 的值不相同，匹配了覆盖信息与执行结果，能够有效提升缺陷定位的精度。

```
module first counter (
    clk,
    reset,
    enable,
    counter_out,
    overflow_out
);
1. always@(posedge clk) begin: COUNTER
2.    if(reset==1'b1) begin
3.        counter_out <= #1 4'b0000;
4.        overflow out <=#1 1'b0; end
5.    else if (enable == 1'bl) begin
6.        counter out <=#1 counter out+1; end
7.    if (counter_out == 4'b1111)begin
8.        overflow out <= 1'b0; end end
endmodule
```

图3 计数器代码示例

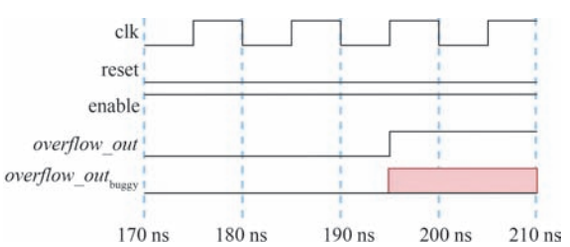


图4 计数器程序输出对比

### 3.2 强语义上下文构建

硬件代码中存在大量与缺陷无关的语句，直接使用原始覆盖矩阵的缺陷定位有效性不足。通过切片技术增强代码语义，构建与缺陷相关的上下文，只保留与缺陷相关的语句，减小缺陷定位搜索空间，提高缺陷定位精度。切片方法通常包含静态切片[23-24]和动态切片[11-12]。静态切片通过分析源代码的控制流和数据流，在不执行程序的情况下提取与特定程序点或变量相关的代码片段。动态切片基于程序实际运行时的行为，通过追踪特定输入条件下的执行路径与数据流，识别实际影响特定程序点或变量的代码段。由于动态切片考虑了运行时的信息，它通常比静态切片更精确。因此，Sepal选择动态切片来构建缺陷的强语义上下文。

图5为一个带有缺陷的Verilog有限状态机程序[9]，该程序由114行代码组成，其中第54行为缺陷语句（正确语句应为GNT0：if（req_0 == 1′b0）begin）。缺陷导致仿真过程中输出信号值与预期值不匹配。如图6所示，在时钟周期为48 ns时，输出信号 $gnt\_0_{buggy}$ 的值与正确值 $gnt\_0$ 不同。

```
39 always @ (state or req_0 or req_1 or req_2 or req_3)
40 begin
41   next_state = 0;
42   case (state)
43     IDLE :  if (req_0 == 1'b1) begin
44         next_state = GNT0;
45           end else if (req_1 == 1'b1) begin
46         next_state= GNT1;
47           end else if (req_2 == 1'b1) begin
48         next state=GNT2;
49           end else if (req_3 == 1'b1) begin
50         next_state= GNT3;
51    end else begin
52         next_state = IDLE;
53           end
54     GNT0 :  if (req_0 == 1'b1) begin
55         next_state = IDLE;
56           end else begin
57      next_state = GNT0;
58   end
59     GNT1 :  if (req_1 == 1'b0) begin
60         next_state = IDLE;
61           end else begin
62      next_state = GNT1;
63   end
64     GNT2 :  if (req_2 == 1'b0) begin
65         next_state = IDLE;
66           end else begin
67      next_state = GNT2;
68   end
69     GNT3 :  if (req_3 == 1'b0) begin
70         next_state = IDLE;
71           end else begin
72      next_state = GNT3;
73   end
74    default :  next state =IDLE;
75   endcase
76 end
```

图5 Verilog有限状态机程序

Sepal对错误触发时刻（48 ns）的测试用例进行动态分析，根据不匹配信号，遍历抽象语法树，构建错误输出的动态切片。如算法1所示，算法首先分析源程序的AST并把不匹配信号添加到队列中（1-4行）。随后，根据队列中的信号，分析其在AST中的数据流依赖和控制流依赖，并把出现在依赖路径上的新信号添加到队列中，直至没有新信号出现，从而构建静态切片集合（5-15行）。第三，找出首次出

现错误的测试用例，再将该测试用例执行的语句组成执行切片集合(16-22行)。最后将静态切片与执行切片取交集，得到动态切片结果。切片结果如表2所示。该有限状态机程序的动态切片包含11行代码。相比静态切片，动态切片将缺陷定位的搜索空间缩减了69.44%。

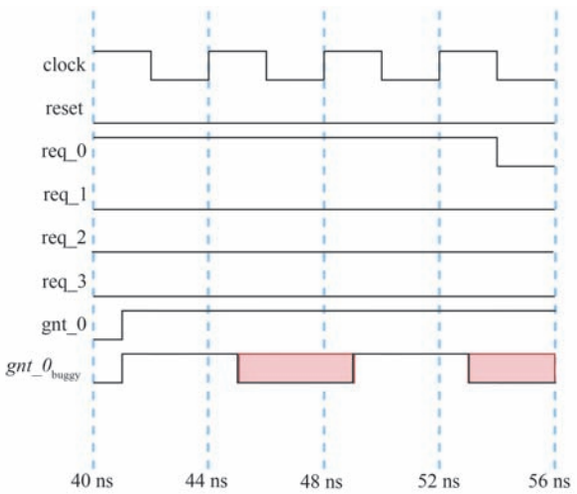


图6　有限状态机程序输出对比

**表2　不同切片技术的可疑语句集合**

| 切片技术 | 代码行 |
|---|---|
| 静态切片 | 39, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 52, 54, 55, 57, 59, 60, 62, 64, 65, 67, 69, 70, 72, 74, 78, 80, 81, 85, 87, 88, 89, 90, 95, 96, 108 |
| 动态切片 | 39, 41, 42, 54, 55, 78, 80, 87, 88, 95, 96 |

### 3.3　可疑值评估

自注意力深度神经网络能够从输入数据中提取复杂特征，并学习缺陷语句与测试结果之间的复杂映射关系。因此，Sepal通过构建多种自注意力深度神经网络模型来评估硬件程序语句的可疑值。

首先，Sepal将代码覆盖矩阵和结果向量作为样本和标签对自注意力深度神经网络进行训练。训练完成后的网络模型建立了程序语句和测试结果之间复杂的映射关系。随后，构建一个虚拟覆盖矩阵并输入训练后的神经网络，神经网络将输出每个语句的可疑值。如图7所示，虚拟覆盖矩阵包含$N$个虚拟测试用例，其中$N$为硬件程序中的语句数量。每个虚拟测试用例仅覆盖一条语句，例如，$x_{ii}=1$表示语句$i$仅被虚拟测试用例$t_i$覆盖。当一个虚拟测试用例被输入到网络中时，模型的输出即为执行该虚拟测试用例失败的概率。由于虚拟测试用例仅覆盖一条语句，模型的输出就是执行该语句并且测试失败的概率，即该语句的可疑值。可疑值的范围在0到1之间，值越大表示语句出错的可能性越高。

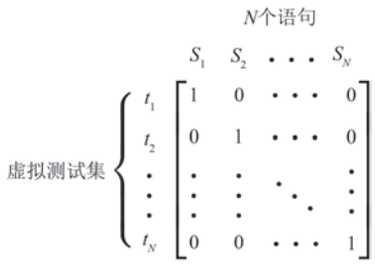


图7　虚拟覆盖矩阵

本文采用了四种自注意力深度网络模型进行训练，包括SAMLP(Self-Attention MLP)、SACNN(Self-Attention CNN)、SARNN(Self-Attention RNN)和Transformer。训练和预测的步骤如下：

(1) 构建深度神经网络模型时，Sepal对传统的MLP、CNN和RNN进行了扩展，通过引入多头注意力机制来增强模型的特征提取能力，自注意力模型的网络结构如附录图11所示。

SAMLP模型由四层全连接隐藏层构成，每层隐藏层节点数根据输入维度动态计算，以此平衡模型的表达能力与计算效率。每个隐藏层后均连接Dropout层(丢弃率25%)避免过拟合，采用ReLU激活函数增强模型非线性学习能力。在第四个隐藏层后，模型引入4头自注意力模块，通过并行计算特征间的高阶交互权重，动态调整各维度特征的聚合强度。模型的输出层通过sigmoid激活函数将可疑值限定在0到1之间。训练阶段采用带动量的随机梯度下降优化器SGD(学习率0.01，动量0.9)，以均方误差(MSE)为损失函数，固定训练180轮，批次大小动态设置为40，同时启用数据随机打乱。该设计通过动态节点分配、注意力驱动的高阶特征交互以及多重正则化策略，在保障模型灵活性的同时增强了对关键特征的辨识能力与训练稳定性。

SACNN模型包含两个卷积层，后面连接最大池化层和Dropout层。在卷积特征图后嵌入序列化多头注意力模块，将二维特征图重构为序列数据并计算跨通道注意力权重，增强局部特征的全局关联性。随后是两个全连接层，输出层使用sigmoid激活函数输出可疑值。

SARNN模型由双向LSTM特征提取组件和单层线性分类器组成，在RNN隐藏状态序列上叠加多头注意力机制，通过并行注意力头捕捉不同时间步的依赖模式，并融合多尺度时序特征。分类器将注

意力加权后的RNN输出映射到单个单元，输出层使用sigmoid函数生成可疑值。

Transformer利用自注意力机制处理序列数据，无需循环结构即可捕捉长距离依赖关系。模型包括输入嵌入层、位置编码、多个Transformer编码器层以及输出层，输出层使用sigmoid激活函数输出可疑值。

$$numbers = \mathrm{int}\left(\max\left(30, \mathrm{round}\left(\frac{input}{30}\right)\right) \times 10\right) \quad (6)$$

（2）将每个缺陷版本的代码覆盖矩阵和结果向量作为训练集，训练这些深度神经网络模型，学习语句与缺陷之间的复杂映射关系。

（3）构建虚拟测试用例作为测试集，将其输入到训练好的深度神经网络模型中。模型输出每条语句的可疑值。

（4）根据自注意力深度神经网络模型输出的可疑值对语句进行降序排列，可疑值越大的语句排名越靠前，表示含有缺陷的概率越大，代码修复人员能够较早地审查高可疑语句。

## 4 实验与结果分析

### 4.1 实验设计

（1）数据集：为了验证Sepal的有效性，本文采用CirFix[9]提供的公开硬件程序数据集进行实验。该缺陷数据集由三位专家依据实际经验植入，涵盖了工业设计中常见的缺陷类型，在硬件代码缺陷定位领域被广泛使用。数据集共有33个缺陷项目，每个项目均包含硬件电路设计、相应的激励文件、正确硬件程序版本及多个缺陷版本。缺陷语句的位置通过对比两个版本的代码确定。表3对这些缺陷项目的详细信息进行了总结。

**表3 实验程序**

| 项目名称 | 功能描述 | 代码行数 | 测试用例数量 |
|---|---|---|---|
| decoder_3to8 | 3-to-8 decoder | 25 | 56 |
| first_counter_overflow | 4-bit counter with overflow | 56 | 35 |
| flip_flop | T-flip flop | 16 | 39 |
| fsm_full | Finite state machine | 115 | 66 |
| lshift_reg | 8-bit left shift register | 30 | 44 |
| mux_4_1 | 4-to-1 multiplexer | 19 | 51 |
| i2c | Two-wire, bidirectional serial bus for data exchange between devices | 2018 | 482 |
| reed_solomon_decoder | Core for Reed-Solomon error correction | 4366 | 148 |
| sha3 | Cryptographic hash function | 499 | 824 |
| sdram_controller | Synchronous DRAM memory controller | 420 | 95 |

（2）实验环境与工具：实验环境为Windows 10操作系统，搭载Intel Core i7-9750H处理器，配备16 GB内存和NVIDIA GeForce GTX 1660 Ti显卡。实验工具包括ModelSim10.5，pyverilog 1.3.0，Python 3.9.7和PyTorch 2.2.0。

（3）评估指标：实验选择了四种在缺陷定位领域广泛使用的评估指标：

Top-K[25]：指在排名列表的前K位置中至少有一个缺陷语句的缺陷版本的数量。本文遵循先前的研究工作[26-27]，将K值设为1、3和5。Top-K的值越大，表示缺陷定位效果越好。

Mean First Rank（MFR）[28-29]：它首先计算缺陷版本中第一个缺陷语句的排名，然后计算该项目中所有缺陷语句排名的平均值。MFR值越低，表示缺陷定位有效性越好。

EXAM[30]：EXAM指标表示在找到第一个实际缺陷语句之前需要检查的语句百分比。较低的EXAM值表明缺陷定位表现更好。

Wilcoxon符号秩检验（Wilcoxon signed-rank，WSR）[31]：它是一种非参数统计检验方法，能够有效评估样本间的显著性差异。

### 4.2 实验结果

4.2.1 RQ1.

SAMLP、SACNN、SARNN和Transformer四种深度神经网络模型的缺陷定位有效性如何？

RQ1主要验证这四种深度神经网络模型在Sepal缺陷定位方法中的有效性，以便选择最好的模型进行下一步的比较。这四种深度神经网络模型分别以Sepal-SAMLP、Sepal-SACNN、Sepal-SARNN和Sepal-Transformer进行命名。

表4列出了四种模型在Top-1、Top-3、Top-5和MFR指标下的数据，其中粗体表示最优值。表4数据表明，Sepal-SAMLP在Top-K和MFR指标上表现最好，展现出了更好的缺陷定位效果。例如，在Top-1指标下，Sepal-SAMLP能够定位12个缺陷，占比36.36%，分别是Sepal-SACNN和Sepal-Transformer的3倍和4倍。同样，对于MFR指标，Sepal-SAMLP的MFR为5.36，Sepal-Transformer的MFR为6.45，降低了16.90%。

**表4　Sepal-SAMLP、Sepal-SACNN、Sepal-SARNN和Sepal-Transformer的Top-K和MFR结果对比**

| 缺陷定位技术 | Top-1 | Top-3 | Top-5 | MFR |
|---|---|---|---|---|
| Sepal-SAMLP | **12** | **24** | **26** | **5.36** |
| Sepal-SACNN | 4 | 21 | 23 | 5.70 |
| Sepal-SARNN | 3 | 20 | 23 | 6.09 |
| Sepal-Transformer | 3 | 20 | 23 | 6.45 |

注：粗体表示最优值。

实验采用EXAM指标进一步评估四种模型在Sepal中的有效性。图8展示了四种模型的EXAM分布情况，对于每条曲线，$x$轴表示已检查的可执行语句的百分比，而$y$轴表示在所有缺陷版本中已经定位缺陷的版本的百分比。图中曲线的标记点表示在所有缺陷版本中检查了给定百分比的可执行语句之后找到的缺陷的版本占所有缺陷版本的百分比。从图中可以发现，四种模型仅需检查60%的代码行，就能够定位全部缺陷版本。其中，Sepal-SAMLP检查程序中20%的语句后，已经筛选出将近70%的缺陷，比其他三种模型多定位大约30%的缺陷版本。通过Top-K、MFR和EXAM指标可以发现，Sepal-SAMLP自注意力深度神经网络模型的缺陷定位精度更高，更适合硬件代码缺陷定位。

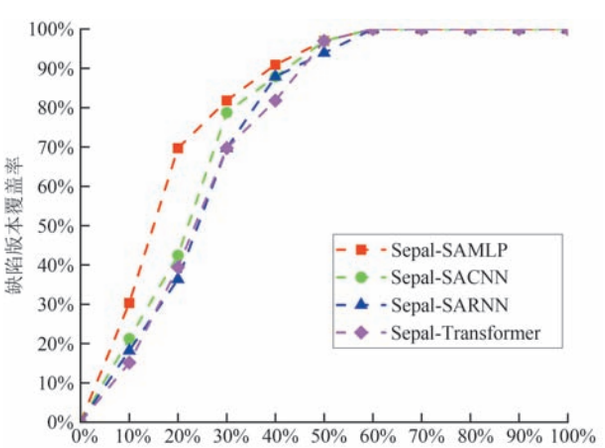


图8　Sepal的EXAM分布情况

硬件描述语言（如Verilog/VHDL）具有严格的模块化结构和层次化依赖，缺陷常涉及跨模块的时序问题或逻辑冲突。这要求模型既能捕捉局部语法细节，又能理解全局上下文。MLP的全连接层支持输入元素间的全局交互，无需堆叠多层即可直接建模长距离依赖。通过自注意力机制，模型能动态聚焦关键代码段（如信号赋值或条件分支），提升对复杂逻辑错误的敏感性。而SACNN的局部卷积核难以有效捕捉远距离依赖，即使引入自注意力，仍需额外设计（如空洞卷积）扩展感受野。SARNN顺序处理导致并行性差，且长程梯度传递不稳定，自注意力仅能部分缓解此问题。Transformer虽擅长全局建模，但其多头注意力的冗余设计可能在小型硬件数据集上过拟合，缺陷定位效果并不好.

4.2.2　RQ2.

Sepal与最先进的动态缺陷定位方法相比，有效性如何？

在RQ2中，将Sepal与目前最先进的两种动态缺陷定位方法Tarsel和Cirfix进行对比，验证Sepal方法的有效性。通过RQ1的结论，本文选择表现最好的自注意力深度神经网络Sepal-SAMLP进行实验。Tarsel使用代码覆盖信息和可疑值公式来计算硬件设计中各语句的可疑值，而Cirfix则运用程序切片技术提取缺陷相关语句，从前到后遍历切片直至定位到缺陷位置为止。本文选择了六种来自不同等价类的可疑值计算公式来实现Tarsel方法，包括Optimal[32]、Wong2[33]、Wong1[33]、Rogot1[34]、Ochiai[35]和Dstar[20].

表5展示了Sepal-SAMLP与Tarsel和Cirfix在所有缺陷版本上的定位结果。实验结果表明Sepal-SAMLP的缺陷定位有效性更高。例如，Sepal-SAMLP的Top-1、Top-3和Top-5指标分别为12、24和26，超过了Cirfix的5、10和16。对缺陷定位领域最关注的Top-1指标，Sepal-SAMLP比Tarsel增长了50%至200%。同时，Sepal-SAMLP的MFR指标也优于Tarsel和Cirfix，下降了约51%至59%。以Tarsel中表现最好的公式Dstar为例，Tarsel-Dstar的MFR为10.94，高于Sepal-SAMLP的5.36，表明Tarsel-Dstar定位缺陷需要检查更多的语句，缺陷定位有效性低于Sepal-SAMLP。

为了进一步比较Sepal-SAMLP与其他缺陷定位方法的有效性，本文使用EXAM指标进行了评估对比。图9展示了Sepal-SAMLP、Tarsel和Cirfix的EXAM分布。图中可以看出Tarsel和Cirfix的曲

**表5 Sepal-SAMLP、Tarsel和Cirfix的Top-$K$和MFR结果对比**

| 缺陷定位技术 | Top-1 | Top-3 | Top-5 | MFR |
|---|---|---|---|---|
| Tarsel-Optimal | 3 | 10 | 16 | 12.70 |
| Tarsel-Wong2 | 5 | 20 | 22 | 13.03 |
| Tarsel-Wong1 | 3 | 10 | 16 | 12.12 |
| Tarsel-Rogot1 | 6 | 20 | 23 | 11.82 |
| Tarsel-Ochiai | 7 | 17 | 21 | 11.76 |
| Tarsel-Dstar | 8 | 18 | 21 | 10.94 |
| Cirfix | 5 | 10 | 16 | 11.91 |
| Sepal-SAMLP | **12** | **24** | **26** | **5.36** |

注：粗体表示最优值。

线始终低于Sepal-SAMLP的曲线，表明Sepal-SAMLP比Tarsel和Cirfix定位有效性更高。例如，在检查40%的代码行时，Sepal-SAMLP已经定位出90.91%的缺陷版本，而Tarsel-Wong2和Cirfix分别只达到了69.70%和45.46%.

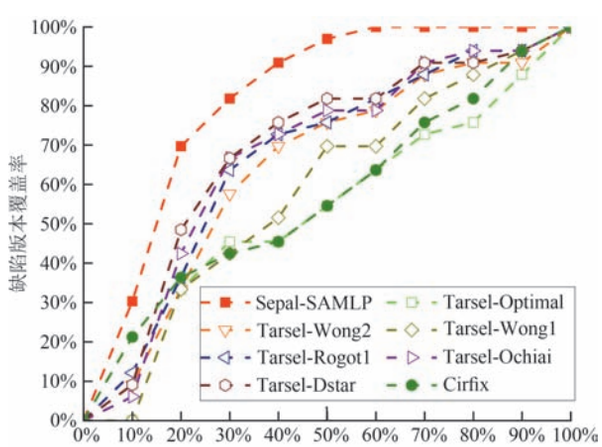


图9 Sepal-SAMLP、Tarsel和Cirfix的EXAM分布

本文采用了Top-K、MFR和EXAM指标来评估Sepal的有效性。然而，这些指标缺乏对结果的统计分析。为了进一步验证Sepal方法在统计学上的有效性，本文对各类方法的缺陷定位结果进行了Wilcoxon符号秩检验。WSR是一种用于比较两组相关样本的非参数统计检验方法。通过计算两组数据之间的差异，得到相应的p值来评估这些差异的统计显著性。该检验包括双尾检验和单尾检验，其中双尾检验用于检测两组数据之间是否存在任何方向上的差异。双尾检验的原假设是两组数据的中位数相等，备择假设是中位数不等，即存在显著差异。而单尾检验用于检测两组数据之间在特定方向上的差异。单尾检验的原假设是两组数据的中位数相等或一个方向上相等，备择假设是在特定方向上存在显著差异。如果计算得到的$p$值小于给定的显著性水平，则可以拒绝原假设，认为两组数据之间存在显著差异。

在本文中，显著性水平$\alpha$=0.05。表6展示了Sepal-SAMLP与Tarsel和Cirfix在定位结果上的统计比较。表中数据表明，Sepal-SAMLP相较于Tarsel和Cirfix的左尾检验$p$值均小于$\alpha$，从而可以拒绝原假设，即Sepal-SAMLP显著优于Tarsel和Cirfix。以Sepal-SAMLP与Tarsel-Ochiai为例，其双尾检验、右尾检验和左尾检验的$p$值分别为0.011 719、0.994 141和0.005 859。从双尾检验来看，可以拒绝原假设，认为Sepal-SAMLP与Tarsel-Ochiai存在显著差异。左尾检验结果表明Sepal-SAMLP显著优于Tarsel-Ochiai，即Sepal-SAMLP的MFR指标显著更小。

**表6 Sepal-SAMLP与Tarsel和Cirfix的统计比较**

| 缺陷定位技术 | 双尾检验 | 右尾检验 | 左尾检验 | 结论 |
|---|---|---|---|---|
| Sepal-SAMLP vs. Tarsel-Optimal | 0.009 766 | 0.997 07 | 0.004 883 | **better** |
| Sepal-SAMLP vs. Tarsel-Wong2 | 0.007 686 | 0.996 157 | 0.003 843 | **better** |
| Sepal-SAMLP vs. Tarsel-Wong1 | 0.003 906 | 0.999 023 | 0.001 953 | **better** |
| Sepal-SAMLP vs. Tarsel-Rogot1 | 0.011 719 | 0.994 141 | 0.005 859 | **better** |
| Sepal-SAMLP vs. Tarsel-Ochiai | 0.011 719 | 0.994 141 | 0.005 859 | **better** |
| Sepal-SAMLP vs. Tarsel-Dstar | 0.025 062 | 0.987 469 | 0.012 531 | **better** |
| Sepal-SAMLP vs. Cirfix | 0.037 109 | 0.986 328 | 0.018 555 | **better** |

4.2.3 RQ3.

Sepal方法中各个阶段对缺陷定位的有效性如何？

RQ3进行了消融实验，以评估Sepal方法中各个阶段对缺陷定位的贡献。Sepal方法由三个关键阶段组成：VCD对比、动态切片以及自注意力深度神经网络。为了探究每个阶段的有效性，每次移除一个步骤，得到不同Sepal变体。VCD对比和动态切片的组合为SAMLP-VS，VCD对比和深度神经网络的组合为SAMLP-VD。

消融实验未包含动态切片与深度神经网络的组合变体的原因是移除VCD对比可能导致缺陷定位失效。目前，判断测试用例成功或失败是通过对比正确输出和实际仿真输出，但由于硬件代码输出结果存在时序延迟，这可能导致语句覆盖路径和测试结果出现不一致。例如，考虑连续执行的两个测试用例$t_1$和$t_2$，其中$t_1$在执行过程中首次覆盖了缺陷语句$s_{bug}$，但由于硬件代码的时序延迟，输出文件未能及时反映状态变化，导致$t_1$被误判为通过。而$t_2$在运行过程中未覆盖$s_{bug}$，却因$t_1$的延迟输出而被误判为失败。时序问题直接影响Sepal第二阶段的动态切片结果。基于失败测试用例$t_2$的动态切片将遗漏缺陷语句$s_{bug}$，导致第三阶段DNN模型生成的可疑语句列表中缺失真正的缺陷语句，最终造成定位失败。因此，本文的消融实验保留了VCD对比步骤。

图10展示了Sepal-SAMLP与各变体的缺陷定位结果对比。实验结果表明Sepal-SAMLP的定位有效性优于其各类变体。值得注意的是，Sepal-SAMLP在i2c程序上定位有效性低于SAMLP-VS。主要原因在于该程序的缺陷位置非常靠前。以i2c的某个缺陷版本为例，缺陷语句位于第5行，而SAMLP-WS方法通过切片产生可疑语句集合，从前往后进行排序，缺陷语句排在第3位。然而，缺陷语句排在程序最前面的情况并不常见，具有一定偶然性。Sepal在绝大多数测试程序中的定位表现仍然优于SAMLP-VS变体。因此，消融实验结果表明，Sepal方法中的每个阶段都对缺陷定位的有效性有贡献。

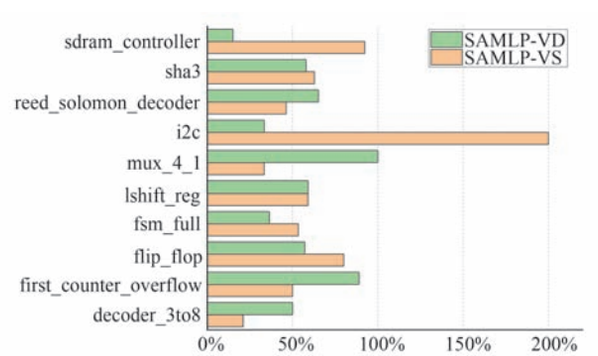


图10　Sepal-SAMLP与各变体对比

## 5　有效性威胁

（1）深度神经网络的随机性：本文的方法使用了深度神经网络，而深度神经网络的随机性可能导致每次预测的结果存在波动，这种随机性是深度神经网络的共有特征。因此，本文采用重复实验的策略，对缺陷定位过程进行了十次重复，取其平均值作为最终结果。通过这种方法，有效降低了随机性对结果的影响，确保了研究结果的可靠性。

（2）单缺陷场景：本文的研究主要针对单缺陷场景，当前方法不适用于多缺陷场景。多缺陷程序存在缺陷的相互干扰，影响缺陷定位的准确性。未来的研究将探索多缺陷场景下的缺陷定位方法，例如使用聚类技术识别失败测试用例和其对应缺陷的因果关系，将多缺陷上下文转换为单缺陷上下文，实现多缺陷程序的缺陷定位。

（3）基准测试程序：本文在实验中使用了CirFix[9]提供的基准测试程序。这些基准程序包含了多种设计规模和真实设计中常见的缺陷类型，但它们可能无法完全代表所有可能的硬件设计场景。在未来的研究中将使用更多的基准测试程序来验证本文方法的有效性。

（4）评价指标：本文实验的评估指标包括Top-K、MFR、EXAM和Wilcoxon-signed-rank符号秩检验等。这些评估指标是缺陷定位领域广泛使用的指标，这方面的有效性威胁可以忽略。

## 6　相关工作

硬件代码缺陷定位在硬件验证领域已被广泛研究。目前，主流的硬件代码缺陷定位方法分为两类：静态缺陷定位方法和动态缺陷定位方法。静态缺陷定位方法通过形式化建模被测设计和正确模型，并比较模型行为之间的差异来帮助发现代码缺陷[36-37]。然而，静态缺陷定位方法存在状态空间爆炸问题，进行大规模硬件设计的形式化建模十分困难[38]。同时，基于模型检验的缺陷定位方法[39-40]和基于布尔可满足性的调试方法[41-42]也因生成反例的复杂性和求解器能力的限制而无法广泛使用。结合了形式验证和模拟的半形式化技术常常需要大量的人工干预。无法自动化地进行代码的缺陷定位[43]。总之，静态缺陷定位方法不仅需要验证人员具备扎实的数学理论知识，还存在状态空间爆炸和求解器能力不足等问题，不适用于大规模设计。

动态缺陷定位方法可以实现自动化的代码缺陷定位，能应用于大规模硬件设计。Cirfix[9]使用静态切片技术从硬件代码仿真轨迹中定位代码的缺陷。该方法不对切片语句进行排序，而是从前往后遍历

切片直至定位缺陷。Detraque[44]执行测试用例并获取代码覆盖信息和执行结果，使用可疑值评估公式计算语句的可疑值，最终生成一个按照可疑值降序的排名列表，指导调试人员优先检查高可疑值的代码区域。Tarsel[10]通过将硬件设计的完整时钟周期划分为细粒度时间片，捕捉状态转换过程中的执行差异，生成更精确的代码覆盖矩阵，提高了缺陷定位的精度。然而，现有的动态缺陷定位方法仍然存在诸多局限性，包括未考虑硬件设计的输出延迟，导致产生不准确的动态覆盖矩阵；代码覆盖矩阵中存在大量与缺陷无关的语句，缺乏代码语句与程序失败的直接关联性；仅使用简单的统计分析方法进行可疑值计算，缺陷定位有效性不足。

本文提出的SAMLP缺陷定位方法在实际工业场景部署面临多重挑战：计算资源方面，尽管其计算复杂度低于Transformer，但处理大规模硬件代码（如数十万行RTL代码）时，全连接层的矩阵运算仍需消耗大量GPU显存与算力，对企业硬件资源配置要求高；实时性上，工业级代码审查需在分钟甚至秒级内反馈结果，而SAMLP的前向传播时间会随序列长度增加而显著增长，难以满足快速迭代开发需求；此外，模型训练依赖高质量标注数据，工业场景中代码缺陷标注成本高昂且数据分布复杂，导致模型泛化能力受限，同时模型的可解释性不足，难以向工程师直观展示缺陷定位逻辑，增加了技术落地与团队协作的难度。

## 7 总 结

本文提出基于自注意力深度学习的硬件代码缺陷定位方法Sepal。Sepal首先采用VCD对比技术解决硬件仿真中程序输出与执行语句不匹配的问题，产生准确的代码覆盖矩阵。其次，通过动态切片技术增强代码语义，建立程序语句与程序失效的关联性，保留与缺陷相关的语句，缩小了缺陷定位的搜索范围。最后，利用自注意力深度神经网络强大的特征学习能力，学习程序代码与缺陷之间复杂的映射关系，实现高精度的缺陷定位。实验结论包括两个方面：(1)相比于SACNN、SARNN和Transformer等自注意力深度神经网络，SAMLP自注意力深度神经网络的缺陷定位有效性更高，更适合硬件代码的缺陷定位。(2) 本文提出的Sepal方法缺陷定位有效性优于目前最先进的动态缺陷定位方法Tarsel和Cirfix。在未来的工作中，将进行更多的实验，验证本文方法的通用性。同时，将关注硬件程序中的多缺陷定位，扩大本文方法的使用范围。

## 参 考 文 献


[1] Flake P, Moorby P, Golson S, et al. Verilog HDL and its ancestors and descendants. ACM Transactions on Programming Languages, 2020, 4: 1-90

[2] Foster H. Assertion-based verification: industry myths to realities//Proceedings of the International Conference on Computer Aided Verification. Princeton, USA, 2008: 5-10

[3] Vineesh V, Kumar B, Shinde R, et al. Enhanced design debugging with assistance from guidance-based model checking. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2020, 40(5): 985-998

[4] Cimatti A, Giunchiglia E, Pistore M, et al. Integrating BDD-based and SAT-based symbolic model checking//Proceedings of the International Workshop on Frontiers of Combining Systems. Santa Margherita, Italy, 2002: 49-56

[5] Friedrich G, Stumptner M, Wotawa F. Model-based diagnosis of hardware designs. Artificial Intelligence, 1999, 111(1-2): 3-39

[6] Clarke E, Klieber W, Nováček M, et al. Tools for practical software verification. Berlin, Germany: Springer, 2011

[7] Groote J, Kouters T, Osaiweran A. Specification guidelines to avoid the state space explosion problem. Software Testing, Verification and Reliability, 2015, 25(1): 4-33

[8] Nalla P, Gajavelly R, Baumgartner J, et al. The art of semi-formal bug hunting//Proceedings of the IEEE/ACM International Conference on Computer-Aided Design. Austin, USA, 2016: 1-8

[9] Ahmad H, Huang Y, Weimer W. Cirfix: automatically repairing defects in hardware design code//Proceedings of the ACM International Conference on Architectural Support for Programming Languages and Operating Systems. Lausanne, Switzerland, 2022: 990-1003

[10] Wu J, Zhang Z, Yang D, et al. Fault localization for hardware design code with time-aware program spectrum//Proceedings of the International Conference on Computer Design. Olympic Valley, USA, 2022: 537-544

[11] Weiser M. Program slicing. IEEE Transactions on software engineering, 2009, SE-10(4): 352-357

[12] Li X, Orso A. More accurate dynamic slicing for better supporting software debugging//Proceedings of the International Conference on Software Testing, Validation and Verification. Porto, Portugal, 2020: 28-38

[13] Hinton G, Osindero S, Teh Y. A fast learning algorithm for deep belief nets. Neural Computation, 2006, 18(7): 1527-1554

[14] Turaga S, Murray J, Jain V, et al. Convolutional networks can learn to generate affinity graphs for image segmentation. Neural Computation, 2010, 22(2): 511-538

[15] Gers F, Schraudolph N, Schmidhuber J. Learning precise timing with LSTM recurrent networks. Journal of Machine

Learning Research, 2002, 3(1): 115-143

[16] Vaswani A, Shazeer N, Parmar N, et al. Attention is all you need//Proceedings of the Annual Conference on Neural Information Processing Systems. Long Beach, USA, 2017: 6000 - 6010

[17] Abreu R., Zoeteweij P., Van Gemund A. J. C. On the accuracy of spectrum-based fault localization//Proceedings of the Testing: Academic and Industrial Conference Practice and Research Techniques-MUTATION. Windsor, UK, 2007: 89-98

[18] Abreu R, Zoeteweij P, Van Gemund A. Spectrum-based multiple fault localization//Proceedings of the IEEE/ACM International Conference on Automated Software Engineering. Auckland, New Zealand, 2009: 88-99

[19] Yan Y, Jiang S, Zhang Y, et al. A fault localization approach based on fault propagation context. Information and Software Technology, 2023, 160: 103520

[20] Wong W, Debroy V, Gao R, et al. The DStar method for effective software fault localization. IEEE Transactions on Reliability, 2014, 63(1): 290-308

[21] Krizhevsky A, Sutskever I, Hinton G. ImageNet classification with deep convolutional neural networks. Communications of the ACM, 2017, 60(6): 84-90

[22] Graves A, Mohamed A, Hinton G, et al. Speech recognition with deep recurrent neural networks//Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing. Vancouver, Canada, 2013: 6645-6649

[23] Binkley D, Harman M. A large-scale empirical study of forward and backward static slice size and context sensitivity//Proceedings of the International Conference on Software Maintenance. Amsterdam, The Netherlands, 2003: 44-53

[24] Binkley D, Gold N, Harman M. An empirical study of static program slice size. ACM Transactions on Software Engineering and Methodology, 2007, 16(2): 8-es

[25] Kochhar P, Xia X, Lo D, et al. Practitioners' expectations on automated fault localization//Proceedings of the International Symposium on Software Testing and Analysis. Saarbrücken, Germany, 2016: 165-176

[26] Xie H, Lei Y, Yan M, et al. A universal data augmentation approach for fault localization//Proceedings of the International Conference on Software Engineering. Pittsburgh, USA, 2022: 48-60

[27] Hu J, Xie H, Lei Y, et al. A light-weight data augmentation method for fault localization. Information and Software Technology, 2023, 157: 107148

[28] Li X, Zhang L. Transforming programs and tests in tandem for fault localization. ACM Transactions on Programming Languages, 2017, 1: 1-30

[29] Li Y, Wang S, Nguyen T, et al. Fault localization with code coverage representation learning//Proceedings of the IEEE/ACM International Conference on Software Engineering. Madrid, Spain, 2021: 661-673

[30] Pearson S, Campos J, Just R, et al. Evaluating and improving fault localization//Proceedings of the IEEE/ACM International Conference on Software Engineering. Buenos Aires, Argentina, 2017: 609-620

[31] Wilcoxon F. Breakthroughs in statistics: Methodology and distribution. New York: Springer, 1992

[32] Naish L, Lee H, Ramamohanarao K. A model for spectra-based software diagnosis. ACM Transactions on Software Engineering and Methodology, 2011, 20(3): 1-32

[33] Wong W, Qi Y, Zhao L, et al. Effective fault localization using code coverage//Proceedings of the Annual International Computer Software and Applications Conference. Beijing, China, 2007: 449-456

[34] Rogot E, Goldberg I. A proposed index for measuring agreement in test-retest studies. Journal of Chronic Diseases, 1966, 19(9): 991-1006

[35] Abreu R., Zoeteweij P., Van Gemund A. J. C. An evaluation of similarity coefficients for software fault localization//Proceedings of the Pacific Rim International Symposium on Dependable Computing. Riverside, USA, 2006: 39-46

[36] Huang Sai-Jie, Chen Ming-Song, Jin Nai-Yong. A Static Analysis Approach for Verilog Based on Constraint Solving. Computer Applications and Software, 2015, 32(12): 90-95 (in Chinese)
(黄赛杰，陈铭松，金乃咏. 一种基于约束求解的Verilog语言静态分析方法. 计算机应用与软件，2015，32(12): 90-95)

[37] Feldman A, Pill I, Wotawa F, et al. Efficient model-based diagnosis of sequential circuits//Proceedings of the AAAI Conference on Artificial Intelligence. New York, USA, 2020: 2814-2821

[38] Peischl B, Wotawa F. Automated source-level error localization in hardware designs. IEEE Design & Test of Computers, 2006, 23(1): 8-19

[39] Vineesh V, Kumar B, Adhaduk J. Identification of effective guidance hints for better design debugging by formal methods//Proceedings of the International Symposium on VLSI Design and Test. Indore, India, 2019: 413-427

[40] Barbon G, Leroy V, Salaün G. Debugging of behavioural models using counterexample analysis. IEEE Transactions on Software Engineering, 2021, 47(6): 1184-1197

[41] Mahzoon A, Grosse D, Drechsler R, et al. Combining symbolic computer algebra and boolean satisfiability for automatic debugging and fixing of complex multipliers//Proceedings of the IEEE Computer Society Annual Symposium on VLSI. Hong Kong, China, 2018: 351-356

[42] Veira N, Poulos Z, Veneris A. Searching for bugs using probabilistic duspect implications. IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems, 2020, 39(12): 5267-5280

[43] Kumar B, Vineesh V, Nemade P, et al. Aries: A semiformal technique for fine-grained bug localization in hardware designs. IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems, 2022, 41(12): 5709-5721

[44] Wu J, Zhang Z, Xu J, et al. Detraque: Dynamic execution tracing techniques for automatic fault localization of hardware design code. Plos One, 2022, 17(9): 19

**附录1.**

**算法1.** 动态切片算法.

输入:源程序 *Program*，覆盖矩阵 ***Matrix***，结果向量 ***error***，不匹配信号集合 *MismatchSignals*.

输出:动态切片 *DynamicSlice*.

初始化:*Visited*={},*Worklist*={},*StaticSlice*={},*ExecuteSlice*={},*DynamicSlice*={}.

```
AST=PyverilogParser(Program)
FOR signal IN MismatchSignals
    Worklist.add(signal)
END FOR
WHILE Worklist NOT EMPTY
    current_sig = Worklist.pop()
    IF current_sig ∉ Visited THEN
        Visited.add(current_sig)
        related_lines, new_signals=AST.GetDependencies(current_sig)
        StaticSlice = StaticSlice ∪ related_lines
        FOR new_sig IN new_signals
            IF new_sig NOT IN Visited THEN
                Worklist.add(new_sig)
END WHILE
FOR e_i IN error
  IF e^i== 1 THEN
        FOR j IN Matrix.column
            IF Matrix_ij == 1 THEN
                ExecuteSlice.add(j)
        BREAK
END FOR
DynamicSlice = StaticSlice ∩ ExecuteSlice
```

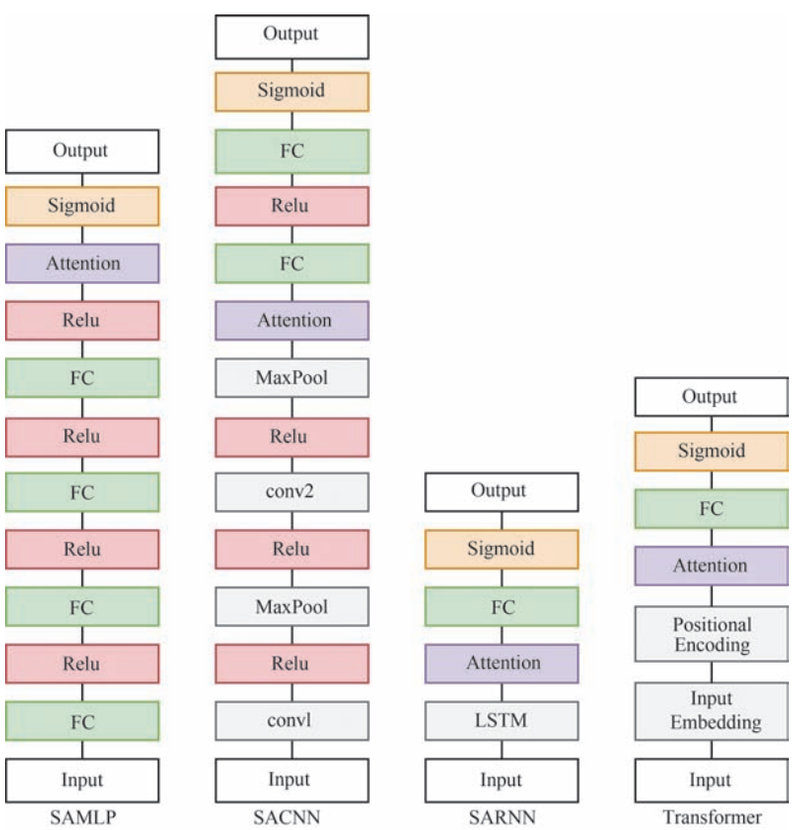


图11 各模型结构示意图

**LIU Zhen-Lei**, M. S. candidate. His research interest is hardware code fault localization.

**HU Jian**, Ph. D., associate professor. His main research interests include electronic design automation and intelligent software engineering.

**Background**

Hardware design code verification is a crucial step in ensuring the quality of hardware products. Dynamic fault localization uses statement coverage information to statistically analyze the probability of faults occurring in specific code segments to achieve localization. However, this method has several limitations, such as mismatches between program outputs and execution information, insufficient semantic information in the code coverage matrix, and simple statistical analysis methods for calculating suspicious values.

In this paper, we propose a self-attention deep learning-based fault localization method for hardware code. First, we use VCD comparison to match the program execution results with the corresponding execution statements within specific clock cycles, constructing precise code coverage matrix and result vector. Second, we employ dynamic slicing technology to enhance semantic information and establish a mapping relationship between suspicious statements and failed test cases, thereby narrowing down the code search space of fault localization. Finally, we utilize a self-attention deep neural network to learn the complex mapping relationships between statements and test results, achieving high-precision fault localization in the hardware code.

We conducted experiments on the publicly available dataset from Cirfix. Experimental results show that among commonly used deep neural networks, the SAMLP deep neural network is more suitable for fault localization. Additionally, our method Sepal outperforms the state-of-the-art dynamic fault localization methods Tarsel and Cirfix.

This work is supported by the National Natural Science of China under Grant No. 61902421. This project aims to improve the effectiveness of fault localization for hardware design code.