针对形式化规范的迭代电路修复

Matthias CoslerCISPA 亥姆霍兹信息安全中心 matthias.cosler@cispa.de

Frederik Schmitt
CISPA 亥姆霍兹信息安全中心
frederik.schmitt@cispa.de

Christopher Hahn
斯坦福大学
hahn@cs.stanford.edu

Bernd Finkbeiner
CISPA 亥姆霍兹信息安全中心
finkbeiner@cispa.de

arXiv:2303.01158v1 [cs.LG] 2 Mar 2023

摘要

我们提出了一种深度学习方法，用于修复符合线性时序逻辑（LTL）形式化规约的时序电路。给定一个有缺陷的电路及其形式化规约，我们训练 Transformer 模型生成满足相应规约的电路。我们提出了一种分离式分层 Transformer，用于对形式化规约和电路进行多模态表征学习。我们引入了一种数据生成算法，使模型能够泛化到更复杂的规约和分布外数据集。此外，我们提出的修复机制显著提升了利用 Transformer 从 LTL 规约自动化综合电路的效果。在留出实例上的表现提升了 6。8 个百分点，在来自年度反应式综合竞赛的分布外数据集上的表现提升了 11。8 个百分点，超越了当前最先进水平。

1 引言

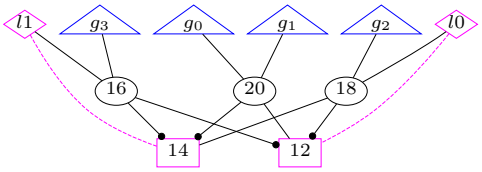
时序电路修复（Katz & Manna，1975年）是指：给定一项形式化规约和一个有缺陷的电路实现，自动计算出满足该形式化规约的实现。电路修复尤其应用于形式化验证。例如，模型检测之后的自动电路调试（Clarke，1997年），或修正由神经网络等启发式方法预测出的故障电路实现（Schmitt 等人，2021b）。本文设计并研究了一种针对线性时序逻辑（LTL）规范（Pnueli，1977年）的深度学习电路修复方法，该方法还提升了利用神经网络合成时序电路的最先进水平。

我们考虑与环境持续交互的时序电路实现。例如，管理共享资源访问的仲裁器会通过向进程授予对共享资源的互斥访问权限与进程交互。线性时序逻辑（LTL）及其变体（例如 STL Maler & Nickovic（2004年）或 CTL Clarke & Emerson（1981年））广泛用于学术界和工业界，以描述时序电路的行为（例如，Godhal 等人（2013年）；IEEE（2005年）；Horak 等人（2021年））。一个典型示例是响应属性（$r \rightarrow \Diamond g$），$\Box$它规定请求 r 始终（ ）最终（ ）会得到授权 g。我们可以通过结合请求 r_0，……，r_3 和授权 g_0，……，g_3 的响应模式与互斥属性，来描述一个管理四个进程对共享资源访问的仲裁器，如下所示：

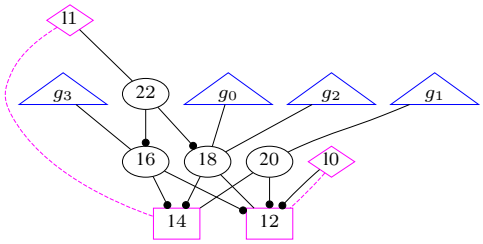
$$\Box(r_0 \rightarrow \Diamond g_0) \wedge \Box(r_1 \rightarrow \Diamond g_1) \wedge \Box(r_2 \rightarrow \Diamond g_2) \wedge \Box(r_3 \rightarrow \Diamond g_3) \qquad \text{响应属性}$$

$$\Box((\neg g_0 \wedge \neg g_1 \wedge (\neg g_2 \vee \neg g_3)) \vee ((\neg g_0 \vee \neg g_1) \wedge \neg g_2 \wedge \neg g_3)) \qquad \text{互斥属性}$$

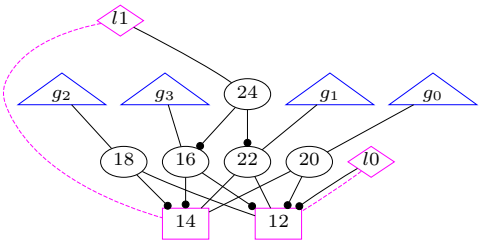
该规范的一种可能实现是一个基于轮询调度器发出授权的电路。然而，对该规范运行神经反应式综合（Schmitt 等人，2021b）后，得到的却是一个有缺陷的电路，如图1a所示。对该实现进行模型检测后，我们发现该电路没有跟踪计数（缺少一个与门），并且违反了互斥属性（同一个变量控制了授权 g_0 和 g_1）。



(a) 综合模型预测的故障电路。



(b) 修复模型第一次迭代后的故障电路。



(c) 正确电路。修复模型在第二次迭代中的最终预测（左侧为 DOT 可视化，右侧为模型以 AIGER 格式生成的输出）。

```
aag 12 5 2 5 5
2          输入0 (输入0)
4          输入1 (寄存器2)
6          输入2（寄存器0）
8          输入3 (寄存器3)
10         输入4(r1)-
12 13      锁存器 0 (l0)
14 24      锁存器 1 (l1)
16         输出0(g3) -
18         输出1 (g-2)
20         输出2(g0) -
22         输出3(g1) -
0          输出4(o4)
16 15 13   和 -门
18 15 12   以及 -个门
20 14 13   以及 -个门
22 14 12   以及 -个门
24 23 17   与门-gates
```

图 1：4进程仲裁器实现的电路在 DOT 可视化中的表示：三角形表示输入和输出，矩形表示变量，菱形节点表示锁存器（触发器），椭圆表示与门，黑点表示反相器（非门）。我们的修复模型的输出是一个 AIGER 电路（右下方）。

我们提出了首个用于修复此类故障电路的深度学习方法，其灵感来自深度学习在 LTL 轨迹生成（Hahn 等人，2021年）和反应式综合问题（Schmitt 等人，2021b）中的成功应用。我们引入了一种新的 Transformer 架构——分离式分层 *Transformer*，该架构考虑了问题输入的不同特征。分离式分层 Transformer 将分层 Transformer（Li 等人，2021年）的优势与 LTL 规范和故障电路的多模态表征学习结合起来。具体而言，它利用了 LTL 规范通常由重复出现的模式构成这一特点。这种架构能够在电路修复问题上成功训练。例如，我们的模型仅用两次迭代，就通过修复图 1a 中的故障电路，生成了符合轮询策略的正确电路实现。每次迭代都会根据规范和作为输入的故障电路预测一个电路。第一次迭代的结果如图1b所示。该电路仍有故障，四个授权中仍有两个由同一个变量控制。不过，电路已朝着实现正常工作的计数器迈进一步：锁存器 $l1$ 现在由与门和反相器组合而成，足以表示一个计数器。第二次迭代最终得到了正确的实现，如图1c所示。

为了有效训练修复模型并促进进一步研究，我们提供了用于电路修复问题监督训练的开源数据集和开源实现[1]。我们证明，经过训练的分离式层次化 Transformer 架构能够泛化到未见过的规范和故障电路。此外，我们展示了该方法可以与现有的用于时序电路综合的神经方法（Schmitt 等人，2021b）相结合，通过修复其错误预测，显著提升整体准确率。在留出实例上，我们将准确率提高了 6.8 个百分点，达到 84%；而在分布外数据集上，提升幅度更为显著：来自年度反应式综合竞赛 SYNTCOMP（Jacobs 等人，2022a）的样本准确率提高了 11.8 个百分点。

[1]https://github.com/reactive-systems/circuit-repair

2 相关工作

电路修复。 修复问题是一个活跃的研究领域，其历史可追溯至 Katz & Manna（1975年）。Jobstmann 等人（2005年；2012年）提出了一种基于博弈的方法，用于利用 LTL 规范修复程序。Baumeister 等人（2020年）提出了一种方法，通过迭代执行修复步骤，从 LTL 规范中综合反应式系统。Ahmad 等人（2022年）提出了一个框架，用于自动修复 Verilog 等硬件设计语言中的缺陷。Staber 等人（2005年）将时序系统的故障定位与纠正同 LTL 规范相结合。
用于时序逻辑和硬件的深度学习。Hahn等（2021）；Schmitt 等人（2021a）开启了深度学习应用于时序逻辑的研究，表明 Transformer 能够理解时序逻辑和命题逻辑的语义。Schmitt 等人（2021b）成功将 Transformer 应用于反应式综合问题。Kreber & Hahn（2021）表明，配备 Transformer 编码器的（W）GAN 能够为 LTL 问题生成合理且具有挑战性的训练数据。Luo 等人将深度学习应用于 LTL_f 可满足性问题。Mukherjee 等人（2022）提出了一种深度学习方法，用于学习 LTL 模型检测的图表示。Vasudevan 等人（2021）将深度学习应用于学习硬件设计的语义抽象。在 Hahn 等人（2022）的研究中，作者从非结构化自然语言生成形式化规约。

反应式综合。 硬件综合问题可追溯至阿隆佐·丘奇在1957年提出的工作（Church，1963年）。Buchi & Landweber（1990年）在1969年便已提供了解决方案，不过当时仅具有理论意义。此后，该领域在算法方面取得了重大进展，例如针对奇偶博弈的拟多项式算法（Calude 等人，2020年）；在概念方面，出现了分布式综合（Pneuli & Rosner，1990年）和有界综合（Finkbeiner & Schewe，2013年）；在高效片段方面，也取得了进展，例如 GR(1)（Piterman 等人，2006年）综合。研究人员还为超属性开发了综合算法（Finkbeiner 等人，2020年）。最近，深度学习已成功应用于硬件综合问题（Schmitt 等人，2021b）。与经典综合相比，深度学习方法可能更加高效，但其代价是具有内在的不完备性。该领域可以利用丰富的工具资源（例如（Bohy 等人，2012年；Faymonville 等人，2017年；Meyer 等人，2018a））。每年都会在 CAV 举办一次竞赛（SYNTCOMP）（Jacobs & P´erez）。

3 数据集

我们以 Schmitt 等人（2021b）的反应式综合数据集为基础，其中每个样本由两个 条目组成：一条采用 LTL 表示的形式化规约，以及一个以 AIGER 格式实现该规约的目标电路。我们构建了一个用于电路修复问题的数据集，其中包含三个 条目：一条采用 LTL 表示的形式化规约、一个有缺陷的电路，以及修正后的目标电路。第 3.1 节详细介绍了电路修复问题输入所用的领域特定语言。第 3.2 节描述数据生成过程，并总结由此生成、用于训练性能最佳模型的数据集（消融实验见第 6 节）。我们从两个角度着手生成数据集：1）收集错误预测，即神经网络模型预测为故障的电路；2）向正确电路引入语义错误，使其模拟人类错误。

3.1 线性时序逻辑（LTL）与与反相器图（AIGER）

LTL 规约。该规范由两个子规约列表组成：假设和保证。假设对环境行为施加限制，而保证则描述电路必须如何响应环境。二者共同构成如下 LTL 规范：$spec \coloneqq (assumption_1 \wedge \cdots \wedge assumption_n) \rightarrow (guarantee_1 \wedge \cdots \wedge guarantee_m)$。如果存在一个实现所需行为的电路，则称该规范为可实现；如果不存在这样的电路，则称其为不可实现。例如，如果所要求的行为相互矛盾，或者环境假设的限制不够充分，则实现可能不可实现。形式上，LTL 规范定义在电路的轨迹之上。若电路 C 满足一个 LTL 规范 φ，则电路的所有可能轨迹 轨迹$_C$ 都满足该规范，即 $\forall t \in$ 轨迹$_C . t \models \varphi$。例如，第 3.1 节仲裁器示例中的 LTL 公式 $(\neg g_0 \vee \neg g_1)$ 要求电路的所有轨迹都遵循 g_0与 g_1之间的互斥行为。如果规范可实现，目标电路就代表该实现；如果规范不可实现，目标电路则代表环境的反策略，表明不存在这样的实现。LTL 的形式语义见附录 A。

算法 1为正确电路实现引入错误的算法。

输入：电路 C，变更数量标准差 σ_c，最大更改次数 M_c，
新变量数量标准差 σ_v，删除行的概率 p_{delete}
输出：电路 C

$changes \sim \mathcal{N}^d(0,\ {\sigma_c}^2,\ 1,\ M_c)$ ▷ 从离散截断高斯分布中抽取样本
对于 $i = 0$至 $changes$ **执行**
  以 概率 p_{delete} **执行**
    $l \sim \mathcal{U}(1, C$中的行数$)$ ▷ 均匀抽取行号
    删除 C中的第 l 行
  else
    $pos \sim \mathcal{U}(1, C$中的位置数量$)$ ▷ 均匀采样位置
    $var' \leftarrow var \leftarrow C$中位置 pos 处的变量编号
    当 $var = var'$ **时执行**
      $var' \sim \mathcal{N}^d(var,\ {\sigma_v}^2,\ 0,\ 61)$ ▷ 从离散截断高斯分布中采样
    将 pos 位置的变量编号替换为 C 中的 var'

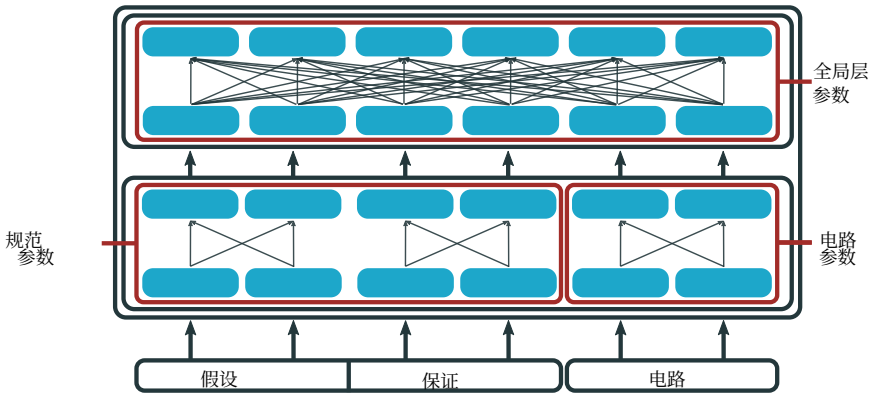
AIGER 格式。有缺陷的电路和目标电路采用一种称为 AIGER 格式（Brummayer 等人，2007）的基于文本的与反相器图表示；例如，参见图 1 右下角。AIGER 格式中的一行通过定义相应节点的输入和输出，来描述锁存器（触发器）和与门等节点。节点之间的连接由用作节点输入和输出的变量编号描述。锁存器由一个输入连接和一个输出连接定义，而两个输入连接和一个输出连接则定义一个与门。整个电路的输入和输出通过仅含一个变量编号的行定义，该编号描述与某个节点的连接。变量编号的奇偶性隐式表示否定。因此，两个连续的数字描述同一连接，其中奇数表示前一个偶数变量编号所代表值的否定值。数字 0 和 1 是常量假和真。AIGER 电路的完整定义见附录 B。

3.2 数据生成

我们复现了 Schmitt 等人（2021b）的神经电路合成模型，并使用其数据集中的所有规格对模型进行了评估，同时保持训练集、验证集和测试集相互分离。我们采用束宽为 3 进行评估，得到数据集 RepairRaw，其中训练集约有 580 000 条规格及其对应的（可能有缺陷的）电路，验证集和测试集则分别约有 72 000 条。我们使用 nuXmv（Cavada 等人，2014）对每个预测电路相对于其规格进行模型检验，并将有缺陷的实现分为以下几类。如果预测电路有缺陷，即违反规格（55%），则该样本为违反。如果综合模型的预测与数据集中的目标电路完全一致（16%），则该样本为匹配。最后，如果预测电路满足规格（或表示正确的反策略），但与目标电路不匹配（29%），则该样本为满足；由于一个规格可能有多个正确的实现，这种情况经常发生。例如，考虑引言中提到的轮询调度器：该规格并未规定各个进程按什么顺序获得资源访问权。

我们通过两个步骤从 RepairRaw 构建最终数据集。第一步，我们考虑违反规范的样本，即神经电路综合网络的错误预测；这些样本自然适合作为电路修复数据集的候选样本。第二步，我们将受人为错误启发的错误引入正确的实现中（有关数据集生成过程及其参数的概述，请参见附录中的图 6）。下文将详细介绍这两个步骤。

神经电路合成的错误预测。我们首先考察 RepairRaw 中违反规范的样本。正如一项规范可能有多个正确的实现，有缺陷的电路也可能有多种修复方法，从而得到正确但彼此不同的实现。对于给定的有缺陷电路，修复方法可能简单直接，也可能冗长复杂。在监督学习设置下，这会带来误导性目标电路的问题。这指的是这样的样本：只有通过冗长复杂的修复故障电路，才能得到目标电路，



Fig图 2：分离式分层 Transformer 中全局层和局部层的结构。Fo r sim为简洁起见，图中仅展示单个假设、单个保证，且每个仅包含两个词元。

尽管较小的修复也能得到正确但不同的实现。我们通过使用综合模型搜索替代解（束宽最高为 4），来识别误导性目标。如果模型找到一个正确的替代电路，且该电路与故障电路之间的莱文斯坦距离更小（定义见附录 C），那么通向替代电路的修复就比通向原始目标的修复更小。随后，我们会用替代电路相应地替换目标电路。我们选取所有与目标电路的莱文斯坦距离为 ≤ 50 的样本，作为最终数据集。

引入错误。 我们提出算法 1，以概率方式在正确电路中引入类似人为失误的错误。这些错误包括遗漏锁存器、与门或反相器，以及组件之间的连接错误。首先，我们确定要在电路中引入的错误或改动数量。为此，我们从均值为零、标准差为 7。5、取值范围为 1 至 50 的离散截断正态分布中进行采样。对于每项改动，我们掷一次硬币，以 $p_{delete} = 0.2$ 的概率删除 AIGER 电路中的一行，以 $1 - p_{delete} = 0.8$ 的概率更改变量编号。对于删除操作，我们从 AIGER 电路中均匀随机选择一行并将其移除。为与数据集保持一致，我们不会删除输入或输出。对于更改变量编号的操作，我们均匀随机选择一个变量编号的位置。该位置可以是输入、输出、锁存器或与门的入边或出边。我们将该位置上的变量编号替换为一个新变量编号；新编号通过从以旧变量编号为中心、标准差为 10、取值范围为数据集中变量编号最小值与最大值（0 至 61）的离散截断正态分布中采样得到。新变量编号不能等于均值，以确保改动确实发生。离散截断正态分布的可视化示例见附录中的图 7。最后，我们通过模型检验对修改后的电路进行抽查，以确定引入的改动是否导致电路出现故障。只有不到 2% 的情况下，电路仍满足规范。

最终数据集。 在最终数据集 Repair 中，61% 的样本包含由算法 1 引入错误的电路，其余样本则基于误预测电路。在 38% 的案例中，修复电路与目标电路之间的莱文斯坦距离小于 10。总体而言，该数据集中的莱文斯坦距离均值为 15。7，标准差为 12。77，中位数为 13（其构成见附录 D 中的图 8）。

4 架构

本节将介绍分离式层次化 Transformer 架构，这是 Li 等（2021）提出的分层 Transformer 的一种变体，并进一步说明我们的架构设置。研究表明，分层 Transformer 在许多应用中都优于原始 Transformer，包括逻辑和数学问题 Li 等（2021）；Schmitt 等人（2021b）。分层 Transformer 与新颖的分离式分层 Transformer 均不受规范中假设和保证顺序的影响。

4.1 分离式分层 Transformer

分层 Transformer 的编码器包含两种具有层次结构的层。局部层只能看到输入的部分内容，而 全局层则处理所有局部层的组合输出。与原始 Transformer 不同，输入在送入局部层之前会先进行分区。位置编码会分别应用于输入的每个分区。局部层之间共享模型参数，但不同分区中的词元之间无法计算注意力。分层 Transformer 有助于理解数学中的重复结构（Li 等人，2021年），并且在处理线性时序逻辑规范方面表现优于其他方法（Sch mitt 等人（2021b））。

我们将分层 Transformer 扩展为分离式分层 Transformer，它包含两种类型的局部层：每个分离式局部层都是一个独立编码器；因此，分离式局部层之间不共享任何模型参数。每个局部层都会独立进行注意力计算。图 2 展示了所提出架构的可视化结果。共享局部层与分层 Transformer 中的局部层相同。一个分离式局部层包含一个或多个共享局部层。分离式局部层和共享局部层的结果会拼接后输入全局层。共享局部层的数量不会改变模型规模，而多个分离式局部层会引入少量额外的模型参数。分离式分层 Transformer 能更好地处理多个在结构、类型或长度上各不相同的独立输入。

4.2 架构设置

我们通过一个独立的局部层将规范与故障电路分开。规范被划分为保证和假设，并输入共享局部层。令注意力为 Vaswani等（2017）的注意力函数（参见附录 K）。在根据规范属性 $assumption_1, \cdots, assumption_n$ 识别假设和保证 $guarantee_1, \cdots, guarantee_m$ $p_1, \ldots, p_{n+m}$时，共享局部层会执行以下计算：

$$\mathit{Attention}(H_{p_i} W_S^Q, H_{p_i} W_S^K, H_{p_i} W_S^V) \quad \text{where} \quad p_i \in \{p_1, \ldots, p_{n+m}\} \ ,$$

其中， H_{p_i} 表示规范属性 p_i所有位置的堆叠表示。因此，注意力计算仅限于每个保证中的词元之间以及每个假设中的词元之间，而学习到的参数 W_S^Q， W_S^K， W_S^V则由所有保证和假设共享。处理电路的独立局部层执行注意力计算：

$$\mathit{Attention}(H_C W_C^Q, H_C W_C^K, H_C W_C^V) \ ,$$

其中， H_C 表示电路所有位置的堆叠表示。因此，计算在电路中的所有词元上进行，但参数 W_C^Q， W_C^K， W_C^V与规范所用的参数不同（见图 2）。

在嵌入和词元化方面，我们针对仅包含少量符号的线性时序逻辑公式和 AIGER 电路的领域特定语言（DSL）进行了专门设计。对于该领域特定语言中的每个符号，我们都引入一个词元。属性（即假设和保证）中的变量限定为五个输入 $i_0 \cdots i_4$和五个输出 $o_0 \cdots o_4$，并为每个变量分别引入一个词元。在 AIGER 格式（用于故障电路和目标电路）中，我们将变量编号固定在 0 到 61 的范围内，从而间接限制电路规模，同时保留合理的表达能力。我们在电路嵌入前添加一个特殊词元，以编码规范的预期可实现性。这决定了电路表示的是满足规范的电路还是反例策略。我们通过独热编码对词元进行嵌入，并将其与学习到的嵌入矩阵相乘。属性采用树位置编码（Shiv & Quirk，2019），正如 Hahn 等人（2021）对线性时序逻辑公式的处理。这种编码将线性时序逻辑公式的树结构融入位置编码，并便于计算树关系。对于电路，我们采用 Vaswani 等人（2017）的标准线性位置编码。

5 实验

本节报告实验结果。我们首先在第 5.1 节介绍训练设置，然后采用两种不同的方法评估模型。模型评估 展示了评估

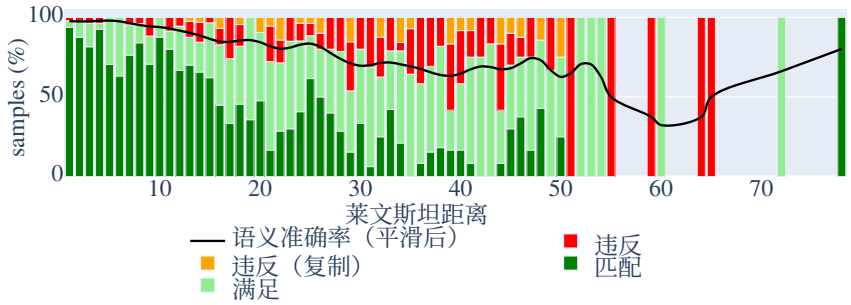


图3：按故障电路与目标电路之间的莱文斯坦距离划分的准确率。

Repair 数据集分布上的修复模型（第 5.2 节）。在综合流水线评估中，先在综合模型的预测上评估修复模型，再反复在其预测上进行评估（第 5.3 节）。我们遵循 Hahn等（2021）的做法，区分语法准确率和语义准确率。如果预测满足规范，则认为样本在语义上正确；如果预测与目标完全相同，则认为预测在语法上正确。

5.1 训练设置

我们训练了一个分离式分层 Transformer，其所有注意力层均包含 4 个注意力头，两个分离的局部层中各有 4 个堆叠的局部层，全局层中有 4 个堆叠层。解码器堆栈包含 8 个堆叠的解码器。解码器和编码器的嵌入维度均为 256，所有前馈网络的维度均为 1024，并使用修正线性单元（ReLU）激活函数。我们使用 Adam 优化器（Kingma 和 Ba，2017），$\beta_1 = 0.9$，$\beta_2 = 0.98$ 和 $\epsilon = 10^{-9}$，并采用 4000 个预热步骤来提高学习率，方法参照 Vaswani 等（2017）的提议。我们在一台 NVIDIA DGX A100 系统的 GPU 上进行训练，批量大小为 256，共训练 20 000 步。我们将规范输入限制为 5 个输入和 5 个输出，属性不超过 12 个（假设+ 保证），且属性的抽象语法树（AST）大小不超过 25。

5.2 模型评估

我们将模型评估到束宽为 16。模型评估的关键结果见表 1 顶部。束宽为 16 时，模型在单次尝试中有 84% 的情况下能够输出正确的实现。分析各束中的结果时，我们发现模型在修复电路时展现出极大的多样性。几乎一半的结果都是正确的实现。为了调查模型是否执行了修复操作，我们找出模型复制了有缺陷的电路的样本（违反（复制））。模型仅复制了 1024 个样本中的 31 个。此外，我们还追踪预测的电路是否包含语法错误，而这种情况很少发生（每个束总计 8 个错误）。我们通过分析以下两点来深入了解模型的表现：a）究竟是什么让模型难以解决某个样本；b）即使预测违反了规范，模型是否仍能显著改进目标电路。

难度度量。我们考虑三个参数来衡量解决特定修复问题的难度：规范的规模（线性时序逻辑公式的抽象语法树）、目标电路的规模（与门 +锁存器），以及有缺陷的电路与目标电路之间的莱文斯坦距离。莱文斯坦距离是决定样本难度的主要指标（见图3）。然而，规范和电路规模对难度的影响或许出人意料地较小（见附录中的图11和图10）。这表明，增大模型规模后，我们的方法有潜力扩展到更大的电路。

改进度量。我们从语义和语法两方面近似评估违反规约的预测是否仍优于故障输入电路。对于语法改进，我们计算故障输入与目标电路之间的距离 $lev(C_i,\ C_t)$，以及预测与目标电路之间的距离 $lev(C_p,\ C_t)$，并求两者之差。如果该差值小于零：$lev(C_p,\ C_t) - lev(C_f,\ C_t) < 0$，则模型在语法上使故障电路朝目标电路有所改进。在我们的测试集上，违反规约的电路平均改进了 -9.98 次编辑操作。对于语义改进，我们通过对原始规格中的每条

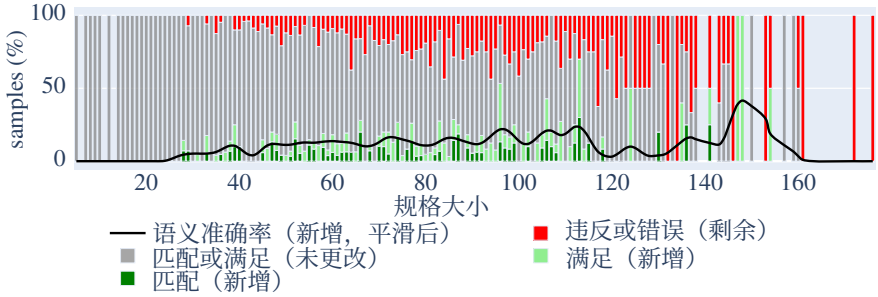


图 4：反应式综合留出测试集上的改进情况（参见表 1 中的测试），按规格抽象语法树的大小分类。我们汇总了在束宽为 16 时所有迭代中的最佳结果。注释 *new* 和 *unchanged* 表示从综合模型评估到修复模型评估期间，状态是否有所改善。

保证分别创建一个新规格，从而得到一组子规约：设 a_1 到 a_n 为原始假设，g_1 到 g_m 为原始保证，则子规约集合定义为 $\{(a_1 \wedge \cdots \wedge a_n) \rightarrow g_i \mid 1 \leq i \leq m\}$。我们近似认为，电路满足的子规约越多，其语义上就越接近正确电路。在我们的测试集中，75.9% 的情况下，预测满足的子规约更多；2.4% 的情况下，预测满足的子规约更少。为了更深入地了解情况，我们在附录中提供了语法改进（图 12）和语义改进（图 13）的小提琴图。由于按照这两种近似评估方式，即使违反规约的预测也优于故障输入，因此自然会引出这样一个问题：通过对模型的预测进行迭代查询，能否提升模型的性能？下一节中，我们将通过实际案例，对神经电路综合模型的预测迭代应用修复模型，深入探讨这一问题。

5.3 综合流程评估

我们展示了如何利用我们的方法提升神经反应式综合的当前最先进水平（见图 5）。首先，我们评估了根据 Schmitt 等人（2021b）复现的综合模型。如果预测的电路违反规范，我们就将该规范与违反规范的电路一同输入修复模型。如果应用一次修复模型后，预测仍违反规范，我们便将该规范和新的违规电路再次输入修复模型，直至问题得到解决。使用所介绍的流水线，我们显著提升了 Schmitt 等人（2021b）的结果，如表 1 下半部分所示。我们评估了合成数据集中的留出样本和分布外基准测试，并筛除了超出输入限制的样本（见第 5.1 节）。测试数据集分别包含从修复数据集和神经综合数据集中随机抽取的留出实例。图 4 深入分析了应用综合流水线时样本状态的变化。

表 1：模型（上）和流水线（下）评估的句法准确率与语义准确率 n.

束宽	1	16
语义准确率	58。3%	84。8%
句法准确率	29.4%	53。2%
每个样本的正确束数	0.58	6.57

	综合模型	第一次迭代后	最多经过 n 次迭代后	n
测试（修复数据集）	-	84.2%	87.5%（+3.3）	5
测试（合成数据集）	77。1%	82.6%（+5.5）	83.9%（+6.8）	5
超时	26。1%	32.5% (+7.4)	34。2% (+8.1)	5
SYNTCOMP	64。1%	71。7% (+7.6)	75。9%（+11.8）	2
智能家居	42.9%	61.9% (+19)	66.7% (+23.8)	2

浅绿和深绿表示通过我们的修复方法额外解决的实例；灰色表示综合网络最初已解决的实例。随着目标电路规模增大，问题变得越来越具有挑战性。总体而言，我们提升了 6.8 个百分点。为证明我们的提升并非源于模型规模扩大，而是新训练数据、架构和迭代评估共同作用的结果，我们还扩大了 Schmitt 等人（2021b）的模型规模，使其参数数量与我们的模型相当或更多。参数数量匹配的模型相较于基模型仅带来微不足道的提升（见附录中的表 2）。我们还找出了一组留出样本，在这些样本上，我们的方法显著优于经典的最先进综合工具 Strix（Meyer 等人，2018b）：对于超时样本，Strix 在 1 小时内无法解决；而我们仍解决了其中的 34.2%，提升了 8.1 个百分点。在年度综合竞赛的真实样本和分布外基准测试中，我们观察到了更显著的提升：智能家居数据集包含针对智能家居应用属性综合的基准测试（Geier 等人，2022），我们的成绩提升了 11.8 个百分点。SYNTCOMP 数据集包含年度反应式综合竞赛的基准测试（Jacobs 等人，2022a;b），模型流水线将最先进水平提升了 23.8 个百分点；仅应用一次也能提升 19 个百分点。

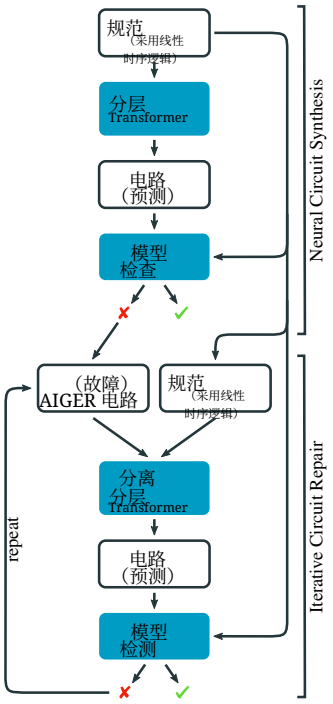


图5：流水线结构

6 消融实验

我们进行了多项消融研究，感兴趣的读者可在附录中查阅。具体而言，我们对数据生成过程进行了参数化，以构建电路修复数据集（见附录 D 中的图 6）。我们通过 GitHub[2]提供了 100 多个生成数据集的丰富集合。我们基于这些数据集和不同的超参数训练了多个模型，相关内容也可在 GitHub 上获取。附录 I 中的图3展示了超参数研究。附录 J 对综合流水线中测试的不同模型的结果进行了深入分析。

7 结论

本文研究了神经网络在电路修复问题中的首次应用。我们引入了分离式分层 Transformer，以处理该问题的多模态输入。我们提出了一种数据生成方法，并设计了一种新颖的算法，用于向电路实现中引入错误。我们成功训练了一个分离式分层 Transformer 模型，用于修复有缺陷的时序电路。由此得到的模型显著提升了神经电路合成领域的最新水平。此外，我们的实验表明，分离式分层 Transformer 有潜力扩展到规模更大的电路。

我们的方法可应用于更广泛的硬件验证领域。可能的应用包括在模型检测或测试后对有缺陷的硬件进行自动化调试。凭借其高效性，性能良好的神经修复方法可以减少硬件设计过程中所需的人为干预。深度学习方法用于电路修复问题的优势在于深度神经网络具备可扩展性和泛化能力：当传统方法受制于该问题较高的计算复杂度时，这使得故障电路能够高效地重新输入网络。此外，神经网络能够泛化到传统修复操作之外，而传统方法的转换能力有限，例如只能替换布尔函数。未来的工作包括将我们的方法扩展到硬件描述语言（如 VHDL 或 Verilog），以及扩展到其他表达安全策略的规约语言，如非干扰或观察确定性。

[2]https://github.com/reactive-systems/circuit-repair

可复现性声明

所有模型、数据集、代码和指南均可在相应的代码仓库中获取。论文中提及的所有数据集和模型、数据生成方法的代码，以及用于训练新模型和评估现有模型的代码，均采用开源 MIT 许可证授权。我们提供了多本 Jupyter 笔记本，引导感兴趣的读者使用代码，以便轻松复现我们的所有结果，并鼓励其他研究人员使用、扩展我们的工作并在此基础上继续开展研究。

致谢

本研究部分由欧洲研究委员会（ERC）HYPER 授权（编号：101055412）提供支持。

参考文献

Hammad Ahmad、Yu Huang 和 Westley Weimer。Cirfix：自动修复硬件设计代码中的缺陷。载于第 *27* 届 *ACM* 编程语言与操作系统架构支持国际会议论文集，第 990–1003 页，2022 年。

Tom Baumeister、Bernd Finkbeiner 和 Hazem Torfah。可解释的反应式综合。载于自动化验证与分析技术国际研讨会，第 413–428 页。施普林格，2020年。

Aaron Bohy、V´eronique Bruyere、Emmanuel Filiot、Naiyong Jin 和 Jean-Franc¸ois Raskin。Acacia+：一种线性时序逻辑综合工具。载于国际计算机辅助验证会议，第 652–657 页。施普林格，2012年。

Robert Brummayer、Alessandro Cimatti、Koen Claessen、Niklas Een、Marc Herbstritt、Hyondeuk Kim、Toni Jussila、Ken McMillan、Alan Mishchenko 和 Fabio Somenzi。AIGER 与反相器图（AIG）格式，版本 20070427。2007年。

J Richard Buchi 和 Lawrence H Landweber。通过有限状态策略求解序列条件。载于*J. Richard B¨uchi*文集，第 525–541 页。施普林格，1990年。

Cristian S Calude、SanjayJain、Bakhadyr Khoussainov、Wei Li 和 Frank Stephan。《准多项式时间内判定奇偶博弈》。《*SIAM* 计算机学杂志》，（0）：STOC17–152，2020。

Roberto Cavada、Alessandro Cimatti、Michele Dorigatti、Alberto Griggio、Alessandro Mariotti、Andrea Micheli、Sergio Mover、Marco Roveri 和 Stefano Tonetta。《nuXmv 符号模型检查器》。载于 Armin Biere 和 Roderick Bloem（编），《计算机辅助验证——第 *26* 届国际会议，*CAV 2014*，作为维也纳逻辑夏季会议 *VSL 2014* 的一部分举行，奥地利维也纳，*2014* 年 *7* 月 *18–22* 日。论文集》，第 8559 卷，《计算机科学讲义》，第 334–342 页。施普林格，2014 年。doi：10.1007/978-3-319-08867-9\ 22。网址：https://doi.org/10.1007/978-3-319-08867-9_22。

阿隆佐·丘奇。递归算术在电路综合问题上的应用。196 3.

Edmund M Clarke。 模型检测。载于国际软件技术基础与理论计算机科学会议 技术与理论计算机科学，第54–56页。施普林格，1997年。

Edmund M Clarke 和 E Allen Emerson。使用分支时态逻辑设计与综合同步骨架。载于程序逻辑研讨会，第52–71页。施普林格，1981年。

Peter Faymonville、Bernd Finkbeiner 和 Leander Tentrup。BoSy：有界综合实验框架。载于国际计算机辅助验证会议，第325–332页。施普林格，2017年。

Bernd Finkbeiner 和 Sven Schewe。有界综合。《软件工具与技术转移国际期刊》，15(5):519–539，2013年。

Bernd Finkbeiner、Christopher Hahn、Philip Lukert、Marvin Stenger 和 Leander Tentrup。从超属性进行综合。*Acta informatica*，57(1):137–163，2020年。

Gideon Geier、Philippe Heim、Felix Klein 和 Marvin Stenger。J.A.R.V.I.S. TSL/TLSF 基准测试套件，2022年。网址：https://github.com/SYNTCOMP/benchmarks。Yashdeep Godhal、Krishnendu Chatterjee 和 Thomas A. Henzinger。从形式化规约合成 AMBA AHB：一项案例研究。《国际软件工具与技术转移杂志》，15(5-6)：585–601，2013年。doi：10.1007/s10009-011-0207-9。网址：https://doi.org/10.1007/ s10009-011-0207-9。

Christopher Hahn、Frederik Schmitt、Jens U. Kreber、Markus N. Rabe 和 BerndFinkbeiner。向神经网络教授时序逻辑。发表于国际学习表征会议，2021年。网址：https://openreview.net/forum?id= dOcQK-f4byz。

Christopher Hahn、Frederik Schmitt、Julia J Tillman、Niklas Metzger、Julian Siber 和 Bernd Finkbeiner。来自自然语言的形式化规约。*arXiv* 预印本 *arXiv:2206.01962*，2022年。

Tom Horak、Norine Coenen、Niklas Metzger、Christopher Hahn、Tamara Flemisch、Juli´an M´endez、Dennis Dimov、Bernd Finkbeiner 和 Raimund Dachselt。通过对超属性进行可视化分析来理解模型检测结果。*IEEETransactions onVisualization and ComputerGraphics*，28(1):357–367，2021年。

IEEE。IEEE 属性规约语言（PSL）标准。*IEEE Std 1850-2005*，第 1–143 页，2005年。doi:10.1109/IEEESTD.2005.97780。

Sven Jacobs 和 Guillermo A. P´erez。第七届反应式综合竞赛：SYNTCOMP 2020。http://www.syntcomp.org/syntcomp-2020-results/。

Swen Jacobs、Guillermo A. Perez、Remco Abraham、Veronique Bruyere、Michael Cadilhac、Maximilien Colange、Charly Delfosse、Tom van Dijk、Alexandre Duret-Lutz、Peter Faymonville、Bernd Finkbeiner、Ayrat Khalimov、Felix Klein、Michael Luttenberger、Klara Meyer、Thibaud Michaud、Adrien Pommellet、Florian Renkin、Philipp Schlehuber-Caissier、Mouhammad Sakr、Salomon Sickert、Gaetan Staquet、Clement Tamines、Leander Tentrup 和 Adam Walker。SYNT- COMP 2020年结果 | 反应式综合竞赛，2022年6月a。网址：http://www. syntcomp.org/syntcomp-2020-results/。

Swen Jacobs、Guillermo A. Perez、Remco Abraham、Veronique Bruyere、Michael Cadilhac、Maxim- ilien Colange、Charly Delfosse、Tom van Dijk、Alexandre Duret-Lutz、Peter Faymonville、Bernd Finkbeiner、Ayrat Khalimov、Felix Klein、Michael Luttenberger、Klara Meyer、Thibaud Michaud、Adrien Pommellet、Florian Renkin、Philipp Schlehuber-Caissier、Mouhammad Sakr、Salomon Sickert、Gaetan Staquet、Clement Tamines、Leander Tentrup 和 Adam Walker。反应式综合竞赛（SYNTCOMP）：2018-2021，2022年6月b。网址：http://arxiv.org/abs/2206.00251。编号：arXiv:2206.00251 arXiv:2206.00251 [cs]。

Barbara Jobstmann、Andreas Griesmayer 和 Roderick Bloem。将程序修复视为博弈。载于国际计算机辅助验证会议，第 226–238 页。施普林格，2005年。

Barbara Jobstmann、Stefan Staber、Andreas Griesmayer 和 Roderick Bloem。查找并修复故障。《计算机与系统科学杂志》，78(2)：441–460，2012年。出版商：爱思唯尔。

Shmuel Katz 和 Zohar Manna。迈向程序自动调试。*ACM SIGPLAN* 通讯，10(6)：143–155，1975年。

Diederik P. Kingma 和 Jimmy Ba。Adam：一种随机优化方法，2017年1月。网址：http://arxiv.org/abs/1412.6980。arXiv:1412.6980 [cs]。

Jens U Kreber 和 Christopher Hahn。使用 Transformer 生成对抗网络生成符号推理问题。*arXiv* 预印本 *arXiv:2110.10054*，2021年。

Wenda Li、Lei Yu、Yuhuai Wu 和 Lawrence C. Paulson。IsarStep：高层次数学推理基准。技术报告 arXiv:2006.09265，arXiv，2021年3月。网址：http: //arxiv.org/abs/2006.09265。arXiv:2006.09265 [cs，stat]type：文章。

WeilinLuo、Hai Wan、Jianfeng Du、XiaodaLi、YuzeFu、RongzhenYe 和DelongZhang。向神经网络传授 LTLf 可满足性检查。

Oded Maler 和 Dejan Nickovic。《连续信号的时序性质监测》。载于《定时与容错系统的形式化技术、建模与分析》，第 152–166 页。施普林格，2004年。

Philipp J. Meyer、Salomon Sickert 和 Michael Luttenberger。《Strix：显式反应式综合卷土重来！》。载于 计算机辅助验证——第 *30* 届国际会议，*CAV 2018*，作为联合逻辑会议 *FloC 2018* 的一部分，于英国牛津举行，*2018*年*7*月*14*日至*17*日，会议录，第一部分，《计算机科学讲义》第 10981 卷，第 578–586 页。施普林格，2018a。doi：10.1007/978-3-319-96145-3\ 31。

Philipp J. Meyer、Salomon Sickert 和 Michael Luttenberger。《Strix：显式反应式综合卷土重来！》。载于 Hana Chockler 和 Georg Weissenbacher（编），计算机辅助验证，《计算机科学讲义》，第 578–586 页，Cham，2018b。施普林格国际出版。ISBN 978-3-319-96145-3。doi：10.1007/978-3-319-96145-3 31。URL：https://strix.model.in.tum.de/。

Prasita Mukherjee、Haoteng Yin、Susheel Suresh 和 Tiark Rompf。Octal：用于线性时序逻辑模型检测的图表示学习。*arXiv* 预印本 *arXiv:2207.11649*，2022年。

Nir Piterman、Amir Pnueli 和 Yaniv Sa'ar。反应式（1）设计的综合。载于 E. Allen Emerson 和 Kedar S. Namjoshi （编），验证、模型检测与抽象解释：第七届国际会议 *VMCAI 2006* 论文集，*2006*年*1*月*8*日至*10*日，美国南卡罗来纳州查尔斯顿，第3855卷《计算机科学讲义》，第364–380页。施普林格，2006年。doi：10.1007/11609773\ 24。网址：https://doi.org/10.1007/11609773_24。

Amir Pneuli 和 Roni Rosner。分布式反应式系统难以综合。载于论文集*[1990] 31*计算机科学基础年度研讨会，第746–757页。IEEE，1990年。

A. Pnueli。程序的时序逻辑。载于第*18*届计算机科学基础年度研讨会（*sfcs 1977*），第46–57页，1977年10月。doi：10.1109/SFCS.1977.32。

Frederik Schmitt、Christopher Hahn、Jens U Kreber、Markus N Rabe 和 Bernd Finkbeiner。用于时序逻辑的深度学习。第六届人工智能与定理证明会议（*AITP*），2021a。

Frederik Schmitt、Christopher Hahn、Markus N. Rabe 和 Bernd Finkbeiner。基于规范模式的神经电路合成。载于神经信息处理系统进展第*34*卷预印本，2021b。网址：https://proceedings.neurips.cc/paper/2021/hash/ 8230bea7d54bcdf99cdfe85cb07313d5-Abstract.html。

Vighnesh Shiv 和 Chris Quirk。用于支持基于树的变换器的新型位置编码。载于神经信息处理系统进展，第32卷。Curran Associates, Inc.，2019年。网址：https://proceedings.neurips.cc/paper/2019/hash/6e0917469214d8fbd8c517dcdc6b8dcf-Abstract.html。

Stefan Staber、Barbara Jobstmann 和 Roderick Bloem。查找并修复故障。载于正确硬件设计与验证方法高级研究工作会议，第 35–49 页。施普林格，2005年。Shobha Vasudevan、Wenjie Joe Jiang、David Bieber、Rishabh Singh、C Richard Ho、Charles Sutton 等。学习用于验证硬件设计的语义表示。神经信息处理系统进展，34：23491–23504，2021年。Ashish Vaswani、Noam Shazeer、Niki Parmar、Jakob Uszkoreit、Llion Jones、Aidan N Gomez、Łukasz Kaiser 和 Illia Polosukhin。注意力机制就是你所需要的一切。载于神经信息处理系统进展，第30卷。Curran Associates, Inc.，2017年。网址：https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html。

线性时序逻辑（LTL）

线性时序逻辑（LTL）的语法由 Pnueli（1977）给出如下。

$$\varphi ::= p \mid \varphi \wedge \varphi \mid \neg\varphi \mid \bigcirc\varphi \mid \varphi \mathcal{U} \varphi \ ,$$

其中 p 是一个原子命题 $p \in AP$。在此背景下，我们假设原子命题集合 AP 可以划分为输入 I 和输出 O：$AP = I \dot{\cup} O$。

LTL 的语义定义在一组轨迹上：$TR ::= (2^{AP})^{\omega}$。设 $\pi \in TR$ 为轨迹，$\pi_{[0]}$ 为轨迹 π 的起始元素，并且对于 $k \in \mathbb{N}$，$\pi_{[k]}$ 为轨迹 π的第 k 个元素。我们用$\pi_{[k,\infty]}$ 表示从 k开始的 π 的无限后缀。我们写作 $\pi \models \varphi$ 轨迹 π 满足公式 φ。

对于轨迹 $\pi \in TR$、$p \in AP$和公式 φ，LTL 的语义定义如下：

- $\pi \models \neg\varphi$ iff $\pi \not\models \varphi$
- $\pi \models p$ iff $p \in \pi_{[0]}$; $\pi \models \neg p$ iff $p \notin \pi_{[0]}$
- $\pi \models \varphi_1 \wedge \varphi_2$ iff $\pi \models \varphi_1$and $\pi \models \varphi_2$
- $\pi \models \bigcirc\varphi$ iff $\pi_{[1]} \models \varphi$
- $\pi \models \varphi_1 \mathcal{U} \varphi_2$ iff $\exists l \in \mathbb{N} : (\pi_{[l,\infty]} \models \varphi_2 \wedge \forall m \in [0, l-1] : \pi_{[m,\infty]} \models \varphi_1)$.

我们还使用可由上述定义的运算符推导出的其他时序运算符和布尔运算符。这包括作为布尔运算符的 $\vee$，$\rightarrow$，$\leftrightarrow$ 以及以下时序运算符：

- $\varphi_1 \mathcal{R} \varphi_2$ （释放）定义为 $\neg(\neg\varphi_1 \mathcal{U} \neg\varphi_2)$
- $\Box\varphi$ （全局）定义为假$\mathcal{R}\varphi$
- $\Diamond\varphi$ （最终）定义为真$\mathcal{U}\varphi$。

反应式综合

反应式综合的任务是寻找一个满足给定形式化规约 φ 的电路 C，即$\forall t \in$ 轨迹$_C . t \models \varphi$，或者确定不存在这样的电路。我们考虑用线性时序逻辑（LTL）表示的、基于一组原子命题（AP）的形式化规约。该规约根据一组输入和输出变量定义系统的期望行为。我们将电路视为系统，更确切地说，是与反相器图的一种文本表示，称为 AIGER 电路。与反相器图使用与门、非门（反相器）和存储单元（锁存器）连接输入边和输出边。

B 与反相器图

与反相器图是用于描述硬件电路的图。这种图通过与门、锁存器和隐式非门，将输入边与输出边连接起来。我们通常使用一种称为 AIGER 格式的文本格式来表示这种图。Brummayer 等人（2007）AIGER 格式使用变量编号来定义变量。变量可以解释为电路中的有线连接，也可以解释为图中的边，其中门和锁存器是节点。

- 否定通过区分偶数和奇数变量编号进行隐式编码。两个连续的变量编号表示同一个变量，偶数变量编号表示未取反的变量，奇数变量编号表示取反的变量。变量编号 0 和 1 分别对应常量值假和真。
- 每条输入边和输出边都分别由一个变量编号定义。
- 与门由三个变量编号定义。第一个变量编号定义与门的出边，后两个变量编号则是入边。出边变量的值由两个入边变量值的合取决定。

- 锁存器由两个变量编号定义：一个出边和一个入边。出边变量的值等于上一个时间步入边变量的值。在第一个时间步，出边变量初始化为假。

AIGER 格式以文件头开头，文件头以字母 aag 开始，后面跟着五个非负整数 M、I、L、O、A，其含义如下：

M=最大变量索引 I =输入数量
L=锁存器数量 O=输出数量 A=与门数量

文件头之后，每一行依次表示输入、锁存器、输出或与门的定义。文件头中的数字决定了每种类型对应的行数。电路定义之后可能会跟着一个可选的符号表，用于为输入、输出、锁存器和与门定义名称。在此语境下，电路可以描述一个满足系统，也可以描述针对该规范的反例策略。

C 莱文斯坦距离

莱文斯坦距离是一种编辑距离度量，用于衡量两个字符串之间的差异程度。设 s_1 和 s_2 是给定的两个字符串，那么莱文斯坦距离 $lev(s_1, s_2)$是将字符串 s_1 转换为字符串 s_2 （反之亦然）所需操作的次数。可能的操作包括删除、插入和替换。

D 数据生成

图 6 展示了数据生成过程的示意图。该过程以评估用于神经电路合成的模型为基础。此过程是参数化的，可以采用多个束宽。对于误预测样本，我们会替换误导性目标（见第 3.2 节）。这一步是可选的，但我们的实验表明，训练会从这一步中获益。在给定的莱文斯坦距离范围内，我们收集样本，构成最终数据集。所有其他样本（莱文斯坦距离更大或预测正确的样本）均按照第 3.2 节和算法 1 进行处理。此过程是可选的，可仅应用于部分样本，也可以进行参数化。根据不同参数，处理结果可用于最终数据集，也可被丢弃。

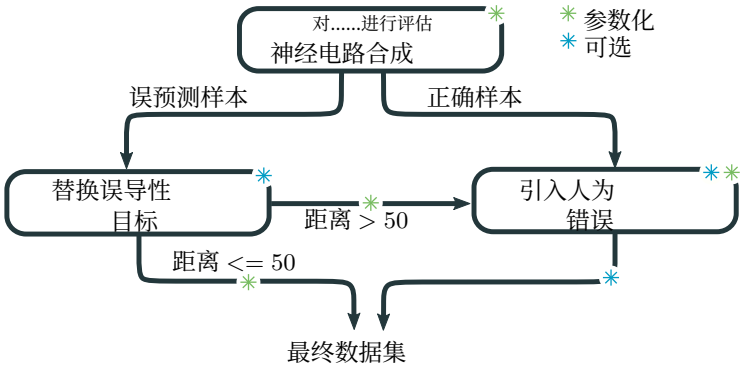


图 6：数据生成过程概览。

图 7 展示了算法 1 中所使用的截断正态分布的概率质量函数。图 8 展示了最终数据集的构成。样本根据其故障电路与目标电路之间的莱文斯坦距离被分入不同的分箱。黄色表示最终数据集中的所有样本，而蓝色 仅表示最终数据集中基于第 3.2 节的样本，红色 仅表示基于第 3.2 节的样本。

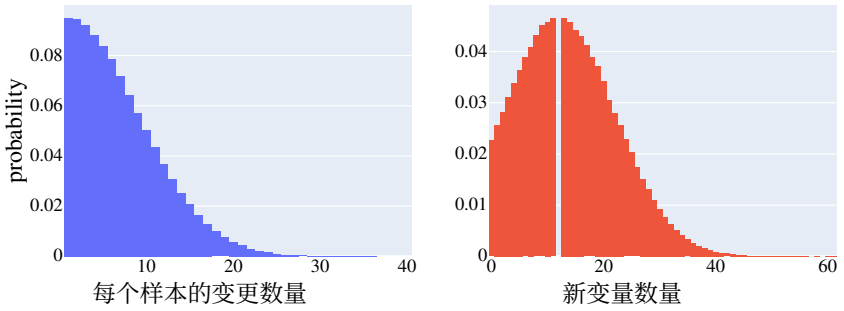


图 7：截断正态分布的概率质量函数。左：采样变更数量的分布。右：在示例旧变量编号 12 的情况下，采样新变量数量的分布。

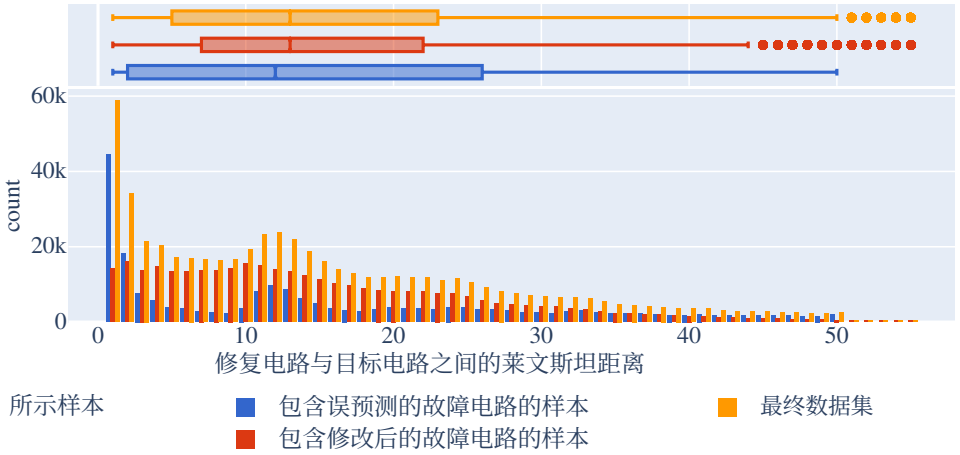


图 8：最终数据集的构成。未显示异常值 > 55 。

在图9中，我们展示了三个替代数据集的构成。样本根据其故障电路与目标电路之间的莱文斯坦距离被分入不同的分箱。数据集 scpa-repair-alter-19（蓝色）展示了一个完全基于第 3.2 节的数据集。数据集 scpa-repair-gen-108 和 scpa-repair-gen-96（红色 和黄色）是我们训练的所有数据集中表现最好的两个数据集，均基于第 3.2 节和第 3.2 节的混合内容。数据集 scpa-repair-gen-96（黄色）就是本文介绍的数据集。

E 难度度量

图10 和图11（以及图3）展示了所提出模型的预测结果，按规范、目标大小以及故障输入电路与目标电路之间的莱文斯坦距离分箱。我们使用束搜索（束宽为 16），仅展示最佳束的结果。不同颜色表示样本所属的不同类别：对于违反规范的预测，标为违反；对于还与故障输入相同的预测，标为违反（复制）。对于正确预测，标为满足；对于还与目标电路相同的预测，标为匹配 。曲线表示经过多个分箱平滑处理后的语义准确率。

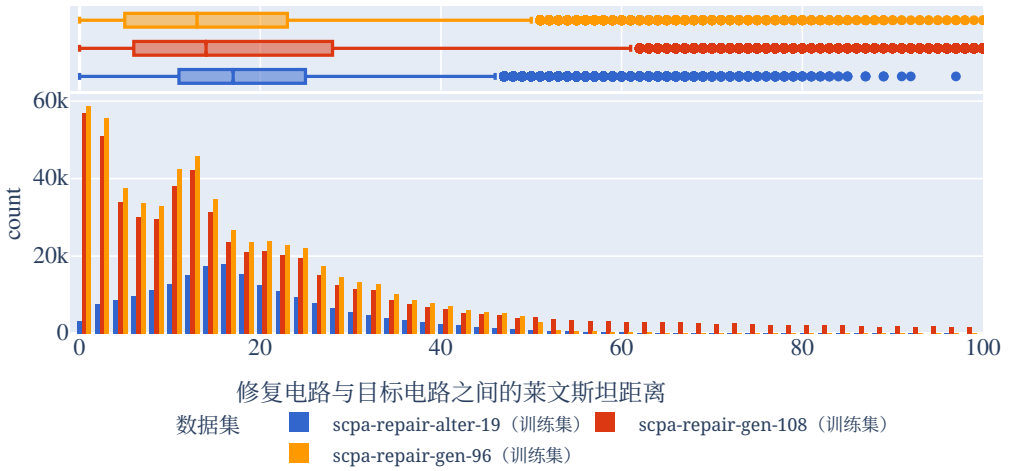


图9：表现最佳的两个数据集与仅基于修改后的电路数据的数据集的比较。范围：0 到 100

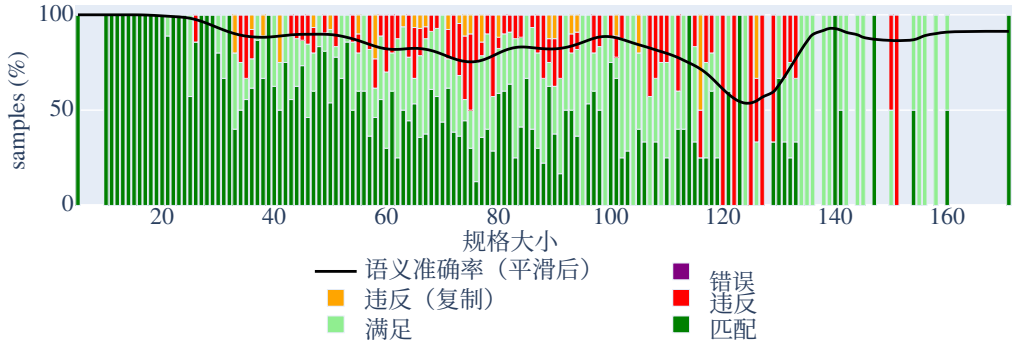


图10：按规格抽象语法树的大小细分的准确率和样本状态。

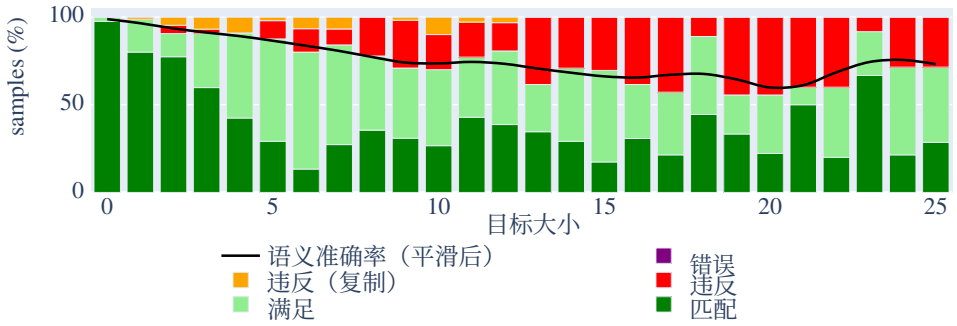


图11：按目标电路大小（与门 +锁存器）细分的准确率和样本状态。

F 改进措施

图 12 显示了故障输入电路与预测之间的莱文斯坦距离差（$lev(C_p,\ C_t) - lev(C_f,\ C_t)$）。值小于零表示朝目标电路方向的语法改进。图 13 显示了满足的子规约数量。电路满足的子规约越多，其语义上就越接近正确电路。

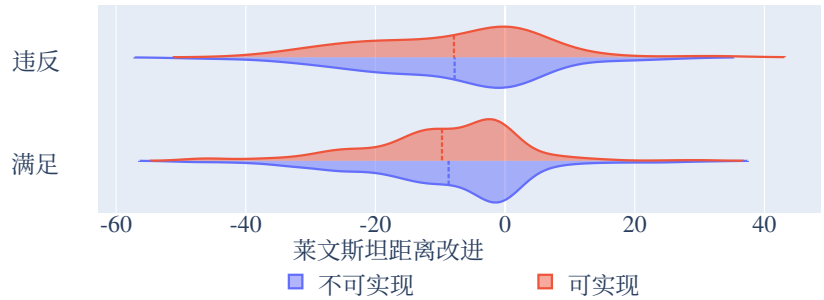


图 12：修复电路和预测相对于目标的莱文斯坦距离改进的小提琴图。虚线表示所示分布的均值。

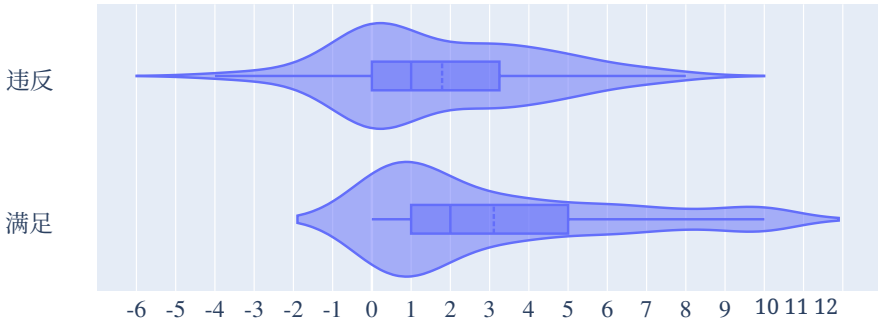
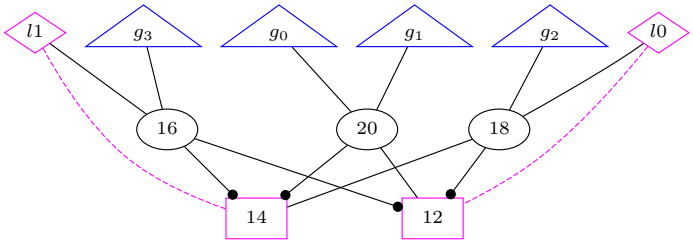


图 13：故障输入电路与预测电路所满足的子规格数量之差的小提琴图。数量越大，改进越大。小提琴图内嵌有箱线图，虚线表示所示分布的均值。仅包含可实现的样本。

G 仲裁器

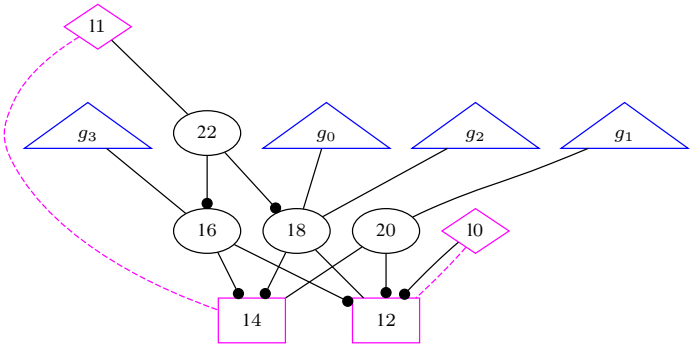
这里，我们再次展示图 1 中的仲裁器，每个图示左侧的所有电路均采用 AIGER 格式。

```
aag 10 5 2 5 3
2          input 0 (i0)
4          input 1 (i1)
6          输入 2 (i2)
8          输入 3 (i3)
10         输入 4 (i4)
12 18      锁存器 0 (l0)
14 16      锁存器 1 (l1)
16         输出 0 (g3) -
18         输出 1 (g2)
20         输出 2 (g0) -
20         输出 3 (g1) -
0          输出 4 (o4)
16 15 13   和 -门
18 14 13   以及 -gates
20 15 12   以及 -gates
```



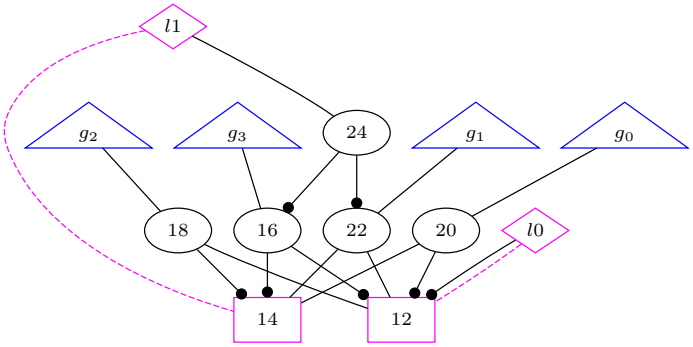
（a）故障电路。在基模型中预测（第0次迭代）。

```
aag 11 5 2 5 4
2          input 0 (i0)
4          输入 1 (i1)
6          输入 2 (i2) -
8          输入 3 (i3)
10         输入 4 (i4)
12 13      锁存器 0 (l0)
14 22      锁存器 1 (l1)
16         输出 0 (g3) -
18         输出 1 (g2)
18         输出 2 (g0) -
20         输出 3 (g1) -
0          输出 4 (o4)
16 15 13   和 -gates
18 15 12   和 -gates
20 14 13   和 -gates
22 19 17   和 -与门
```



(b) 故障电路。由修复模型在第1次迭代中预测。

```
aag 12 5 2 5 5
2
输入 0 (i0)
4          输入 1 (i1)
6          输入 2 (i2) -
8          输入 3 (i3)
10         输入 4 (i4)
12 13      锁存器 0 (l0)
14 24      锁存器 1 (l1)
16         输出 0 (g3) -
18         输出 1 (g2)
20         输出 2 (g0) -
22         输出 3 (g1) -
0          输出 4 (o4)
16 15 13   及 -gates
18 15 12   及 -gates
20 14 13   及 -gates
22 14 12   与 -门
24 23 17   与 -门
```



(c) 正确电路。修复模型在第 2 次迭代中预测得出。

图14：仲裁器合成失败后的尝试及成功修复。

H 缩放参数

在这项实验中，我们扩展了综合模型（Schmitt 等人，2021b），使其参数数量达到或超过我们的模型。这表明，分离式分层 Transformer 参数数量的增加并非整体性能提升的原因。详细结果见表 2。

表 2：综合模型不同配置与我们的模型在模型规模和语义准确率方面的比较。

模型	参数	语义准确率，束宽为 16
综合模型（基线）	14786372	77。1%
综合模型：8 个局部层	17945412（+21.4%）	46.3%（−30.8）
综合模型：8 个全局层	17945412（+21.4%）	46.5%（−30.6）
综合模型：6 个编码器层	17945412 (+21.4%)	58.2% (−18.9)
综合模型：网络规模为 2048（局部层）	16887620 (+14.2%)	77。4%（+0.3）
综合模型：网络规模为 2048（全局层）	16887620 (+14.2%)	77。2% (+0.1)
综合模型：网络规模为 2048（编码器）	18988868 (+28.4%)	77.3% (+0.2)
Repair 模型（我们的方法）	17962820 (+21.5%)	83.9% (+6.8)

I 超参数研究

我们在所展示的数据集（scpa-repair-gen-96）上训练了模型的多个版本，作为表3所示的超参数研究。

J 消融实验

图15展示了使用束宽 16 对流水线进行评估所得的语义准确率。如果某个样本已被正确解决，那么在之后的所有迭代中都会将其计为正确。我们展示了本文所介绍的模型 exp-repair-gen-96-0（蓝色）以及另外两个模型的结果。模型 exp-repair-alter-19-0（绿色）展示了一个仅基于第 3.2 节数据集训练的模型。模型 exp-repair-gen-108-0 和 exp-repair-gen-96-0（红色 和 蓝色）是表现最好的两个模型，使用第 3.2 节和第 3.2 节的数据混合训练。有关数据集的更多信息，请参见图9。

表 3：超参数研究。每列代表一个模型。第一列显示最终的超参数。灰色的超参数值与第一列相同。结果中的粗体值表示该行中的最佳值。

参数									
	嵌入维度	256	256	256	256	256	256	128	256	256
	网络规模	1024	1024	1024	1024	256	512	1024	1024	256
	编码器									
	全局注意力头	4	4	8	4	4	4	4	8	8
	特定注意力头	4	4	4	4	4	4	4	4	4
	电路注意力头	4	4	4	4	4	4	4	4	4
	全局层	4	4	4	4	4	4	4	4	8
	特定层	4	4	4	4	4	4	4	4	4
	电路层	4	6	4	4	4	4	4	8	8
	解码器									
	注意力头	4	4	4	4	4	4	4	4	4
	层	8	6	8	10	8	8	8	8	8
结果										
	训练集									
	loss	**0.03**	0.05	0.03	0.04	0.04	0.04	0.04	0.06	0.08
	准确率	**0.97**	0.95	0.97	0.97	0.96	0.96	0.96	0.94	0.92
	（每个序列）	**0.47**	0.40	0.44	0.43	0.39	0.42	0.42	0.32	0.24
	验证集									
	loss	**0.06**	0.13	0.07	0.06	0.07	0.06	0.07	0.14	0.14
	准确率	**0.95**	0.91	0.95	0.95	0.95	0.95	0.95	0.89	0.88
	（每个序列）	**0.32**	0.27	0.30	0.30	0.29	0.30	0.30	0.22	0.19
	束宽为 1	**0.59**	0.54	0.58	0.58	0.53	0.57	0.53	0.47	0.42
	束宽为 16	**0.82**	0.77	0.81	0.82	0.80	0.82	0.81	0.73	0.69
	测试集									
	束宽为 1	0.58	0.52	0.59	0.57	0.56	0.58	**0.59**	0.44	0.41
	束宽为 16	0.85	0.77	**0.85**	0.82	0.83	0.82	0.81	0.72	0.70

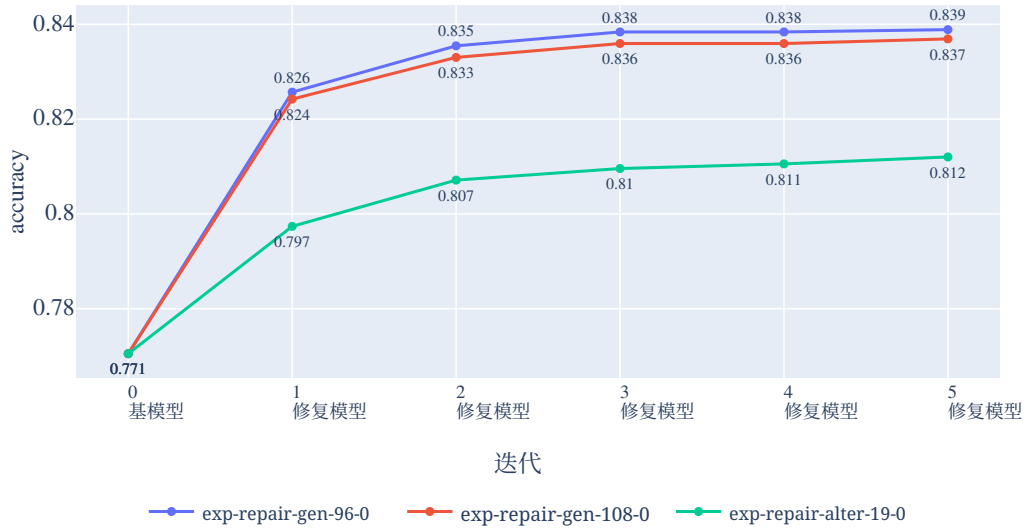
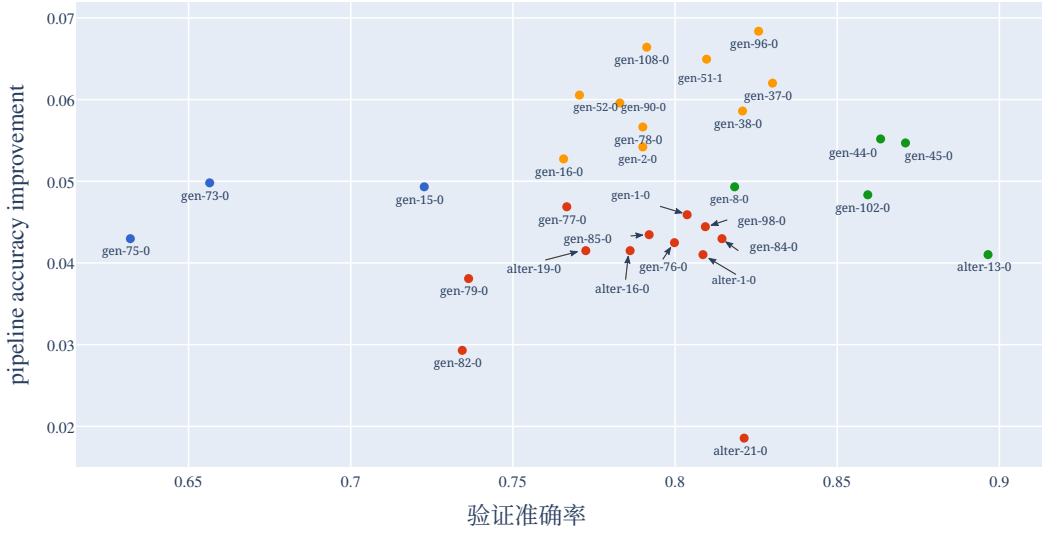


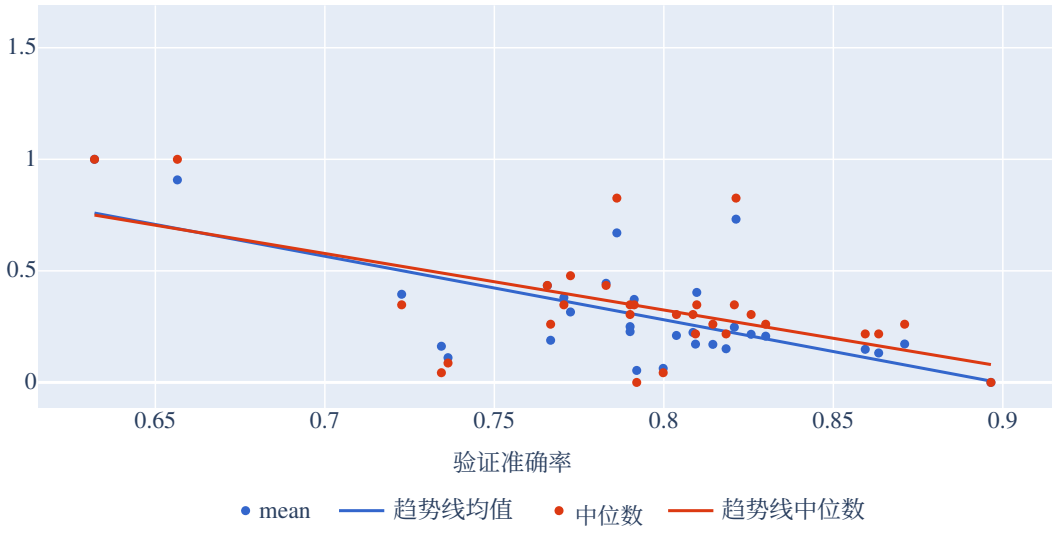
图15：流水线每次迭代后的准确率（汇总）。

在图16中，我们展示了所有使用不同数据集训练的模型。所有训练都使用相同的超参数。我们绘制了流水线中的准确率提升，因此 y 轴表示模型在反应式综合任务上的表现。x 轴表示验证准确率，即模型在电路修复问题上的表现。此外，所有模型的流水线准确率提升均基于相同的分布，而验证准确率则基于训练模型时所用的相应数据集。我们可以看到，流水线准确率较高的模型，其训练所用的数据集包含评估结果（第 3.2 节），而非修改后的电路（第 3.2 节）。这并不令人意外，因为这些数据集更接近流水线准确率所依据的分布。我们可以辨认出几个模型簇，其中一个簇（黄色）具有相对较高的验证准确率和非常显著的流水线准确率提升。该簇中的所有模型都将综合准确率提升了 5 个百分点以上，其中模型*exp-repair-gen-96-0*的提升幅度最高，达到 6.8 个百分点。

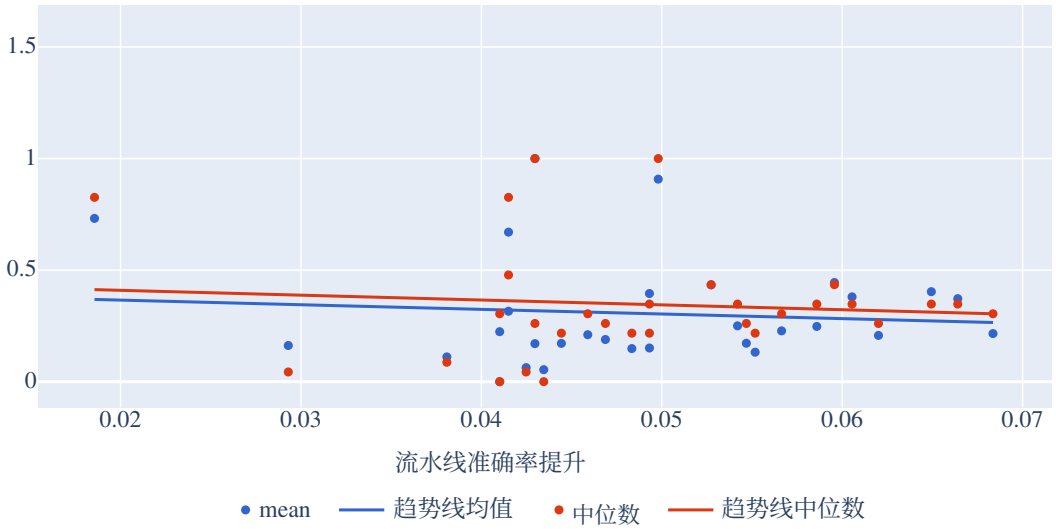


Fig图16：相较于验证准确率的流水线准确率提升（百分点）　　f
所有
训
练模型。颜色基于包含4个簇的k均值聚类。

在图 17 中，我们绘制了所有已训练模型中，故障电路与目标电路之间莱文斯坦距离的均值和中位数。图 17a 中的图表取决于验证准确率（修复问题的准确率），而图 17b 中的图表取决于流水线准确率提升（综合任务上的表现）。



(a) 验证准确率 ($bs = 16$)



(b) 流水线准确率提升

图 17：所有训练模型的验证准确率和流水线准确率提升，与故障电路和目标电路之间的莱文斯坦距离的比较。均值和中位数按最小值和最大值缩放至 0 和 1。仅显示所选模型的标签。

K 缩放点积注意力

对于打包到矩阵 Q（查询）、K（键）和 V（值）中的一组查询、键和值，缩放点积注意力（Vaswani 等人，2017年）定义如下：

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

L 从部分规范修复电路

为展示我们方法的广泛适用性，我们进行了另一项实验。借助经典综合工具，我们根据相应规格生成了一组可能存在故障的电路测试集：与原始规格相比，我们移除了最后一项或两项保证。这种方法确保这些电路中只有 7.8% 仍满足原始规格。我们在这些分布外电路上评估了（而非训练）我们的模型，经过最多 5 次迭代后，语义准确率达到 71.4%（见表 4）。

我们进一步按照（原始）规范大小和目标大小对样本的状态进行了细分分析（见图 18 和图 19）。尽管总体结果表明，我们的模型成功应用于这一问题，但值得注意的是，该模型会产生更多语法错误，尤其是在较大的电路和规范中。与图 10 和图 11 相比尤其如此，在后者中，模型没有生成任何存在语法错误的电路。这很可能是因为本次测试中的有缺陷的电路属于分布外数据。

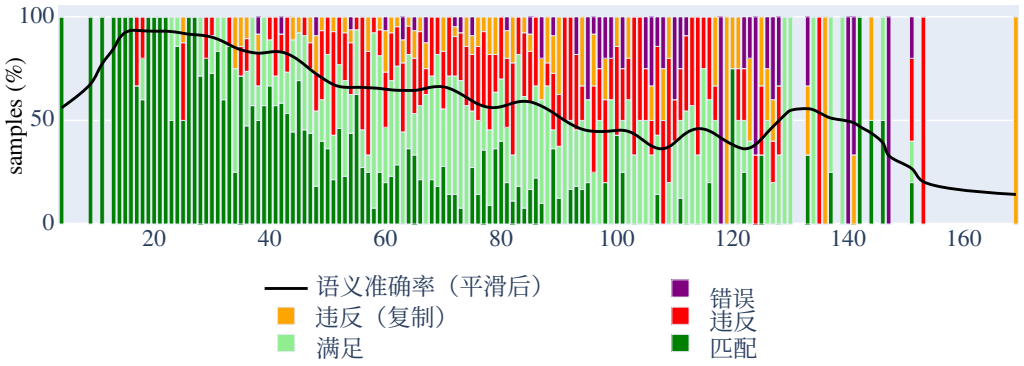


图 18：按规格抽象语法树的大小细分的准确率和样本状态。对部分规范中的故障电路进行评估。

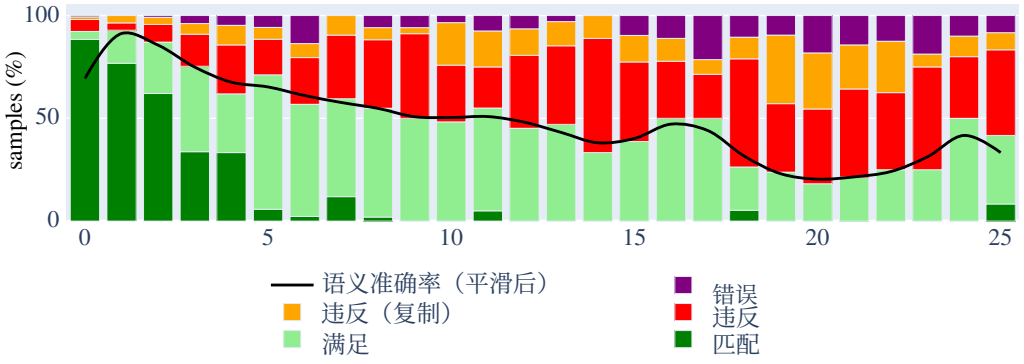


图 19：按目标电路规模细分的准确率和样本状态。对由部分规格生成的故障电路进行评估。

表 4：使用部分规格生成的故障电路的修复结果。（表 1 的扩展）

	综合模型	第一次迭代后	最多经过 n 次迭代后	n
部分	-	64。2%	71.4% (+7.2)	5