T EN S U RE：稀疏张量编译器模糊测试（注册报告）

Kabilan Mahathevan、Yining Zhang、Muhammad Ali Gulzar、Kirshanthan Sundararajah 计算机科学系 Virginia Tech，Blacksburg，VA 24061，美国 电子邮件：{kabilan, yiningz, gulzar, kirshanthans}@vt.edu

arXiv:2603.18372v1 [cs.PL] 19 Mar 2026

摘要——**稀疏张量编译器（STCs）已成为优化高维数据分析和机器学习工作负载的关键基础设施。STC 必须根据高层声明式规范，直接为各种压缩存储格式合成复杂且不规则的控制流，因此极易出现细微的正确性缺陷。现有测试框架依赖于变异计算图，而计算图仅限于标准算子词汇表，无法充分测试这些编译器任意合成循环的能力。此外，由于多个张量间索引复用受到严格规则约束，通用的基于语法的模糊测试器难以生成有效输入。**

本文介绍 TENSURE，这是首个专为测试 STC 设计的可扩展黑盒模糊测试框架。TENSURE 利用爱因斯坦求和（Einsum）表示法作为通用输入抽象，从而能够生成复杂、非传统的张量缩并，揭示 STC 代码生成阶段的边界情况。我们提出一种新颖的基于约束的生成算法，可确保合成内核的语义有效率达到 100%，显著优于基线语法模糊测试器的 ∼3.3% 有效率。为在没有可信参考结果的情况下进行蜕变测试，我们引入了一组语义保持变异算子，利用代数交换性以及存储格式的异质性。我们对两个最先进的系统 TACO 和 Finch 进行评估，发现它们普遍脆弱，尤其是 TACO：TENSURE 在大多数生成的测试用例中发现了崩溃或静默错误编译。这些发现凸显了稀疏编译生态系统亟需专用测试工具。

I. 引言

过去十年间，高效执行科学计算和深度学习工作负载的需求激增，使张量编译器成为现代系统的核心。TVM [1], 、TensorRT [2],、Tri- ton [3], 和 XLA [4] 等编译器框架，通过将高级程序降级为硬件特定代码，大力优化张量计算。与此同时，张量规模迅速扩大，而硬件扩展速度却趋于平缓 [5]。这种差距使得直接在压缩张量存储格式上执行计算成为必要。

然而，针对各种压缩张量数据存储格式上的复合运算，构建优化的张量计算（即内核）并非易事。与稠密数组不同，压缩格式不支持高效的随机访问；因此，生成的内核需要复杂的循环嵌套，以协调多个稀疏操作数之间的访问。因此，基于编译器的方法被广泛认为适用于稀疏张量运算。这一需求推动了稀疏张量编译器（STC）这一专用工具类别的兴起，用于在压缩存储格式上执行计算 [6]–[11]。

遗憾的是，稀疏遍历调度的复杂度远远高于稠密对应项 [6], [9],[10], [12]。支持多种压缩数据结构的需求带来了大量潜藏细微正确性缺陷的可能，其中包括迭代故障和内存安全违规；在串联多个张量运算时，这一问题尤为突出。

近期现实中的故障表明，即使是在更简单的稠密场景中，编译器缺陷也可能造成严重损害。例如，Anthropic 追查发现，Claude 的响应质量下降是由 XLA 错误编译导致的 [13]。诊断此类问题向来十分困难，因为模型输出几乎无法揭示底层编译后的程序，而开发者往往默认信任编译器。稀疏张量流水线进一步加剧了这一挑战：其控制流复杂得多，因此缺陷也更难定位。

本文介绍了一种可扩展的自动化测试框架，专门用于验证稀疏张量编译器（STC）的功能正确性。现有的差分测试方法 [14], [15] 虽可用于验证通用编译器，但由于不同执行环境中的数值精度存在差异，会产生数量不可忽视的误报 [16]，因此从根本上不适用于张量编译器。因此，TENSURE 采用了类似于 POLYJUICE 的方法，利用相同的执行环境进行蜕变测试。

传统蜕变测试中使用的保持语义的变异技术，例如死代码注入、 [17],指令填充 [18], 和控制流重构 [14], [19]——依赖于对命令式控制结构进行变异（例如，if 代码块、循环和语句排序）。然而，STC 通常处理高级声明式规范，例如 *Einsum* （爱因斯坦求和）表示法 [20],，其中不存在显式控制流。因此，这些标准的命令式

模糊测试研讨会（FUZZING）2026年 2026年2月27日，美国加利福尼亚州圣迭戈 ISBN 978-1-970672-04-6
https://dx.doi.org/10.14722/fuzzing.2026.23006
www.ndss-symposium.org

变异方法并不适用，因为控制流并未在源代码中定义，而是在编译过程中合成，从而没有可供传统代码转换操作的目标。

此外，稀疏张量编译器的发展格局存在显著的语法和语义碎片化问题。TACO [6], Finch [7], 和 MLIR 稀疏方言 [9]等主流框架各自提供了不同的声明式领域特定语言，并支持不同范围的 einsum 标准。这种分化还因底层基础设施各异而加剧——从 C++ 模板库到基于 LLVM 的降级流水线，不一而足。因此，构建统一的测试框架并非易事；对一种工具有效的测试用例，在语法上可能与另一种工具不兼容。要弥合这一互操作性差距，就需要一种中间表示，以抽象稀疏张量编译器程序的核心语义特征，使测试生成与特定实现的语法相分离。

此前针对稠密张量编译器的测试工作（例如 NNSmith [21] 和 PolyJuice [22], ）依赖于构建计算图——这类有向无环结构中的节点表示固定的高级算子（例如，矩阵乘法、二维卷积），并通过数据依赖相连。这种方法本质上受限于标准算子库中固定的词汇表；它可以测试编译器优化已知模式的能力，却无法充分检验其为任意且非常规的张量运算合成循环嵌套的能力。更关键的是，这些框架也无法表示压缩存储格式，而这正是稀疏张量编译器的基础，因此遗漏了稀疏编译所需的复杂遍历约束。

据我们所知，目前没有专门针对 STC 功能正确性的测试框架。因此，张量编译器生态系统中相当一部分实际上未经测试。为弥补这一空白，我们开发了一个模糊测试器，它能够生成有效的张量内核，将其转换为目标稀疏张量领域特定语言，并创建语义等价的程序变体以进行蜕变测试。与通用模糊测试不同，后者可通过大量重写规则生成许多等价变体，而单内核张量程序的变异机会有限。

我们在两个最先进的系统上评估了 TENSURE：基于 C++ 的 TACO [6]和基于 Julia 的 Finch [7]。实验发现，TACO 存在严重的健壮性问题：模糊测试器生成的测试用例中，超过 60% 会导致程序崩溃或编译错误。此外，成功集成 Finch 也验证了该框架可扩展至不同编译器架构。总体而言，这些初步结果凸显了当前稀疏张量编译基础设施普遍存在的脆弱性，并展示了 TENSURE 检测潜在缺陷的有效性。

本文的贡献如下：

- **可扩展的 STC 模糊测试器**：据我们所知，我们提出了 TENSURE，这是首个与语言无关、专为验证 STC 而设计的黑盒模糊测试框架。
- **领域特定变异算子**：我们定义了一组利用存储格式的变异算子异质性和代数交换性来生成语义等价的 STC 程序。
- **基于约束的生成算法**：我们对一种生成算法进行了形式化并予以实现，该算法能够求解上下文相关的维度约束，以合成 einsum 表达式。标准的基于语法的模糊测试器的有效率仅为 ∼3.3%，而我们的方法可确保 100% 的语义有效性，从而支持对深层编译阶段进行高吞吐量测试。

本文其余部分的安排如下：第二节介绍稀疏编译的背景，第三节阐述开展专门模糊测试的必要性。第四节详细介绍 TENSURE 的设计。第五节对该工具进行初步评估。第六节和第七节分别讨论其局限性和相关工作，最后在第八节总结全文。

II. 背景

A. 张量与 *Einsum*

张量是多维数组，其计算可以用 einsum 表示法简洁地表达 [23]。einsum 表示法通过隐含地对张量间的共享索引求和，推广了张量运算，无需显式使用求和符号。

$A(j) = B(i,j) * C(i)$ 是一个简单的内核，其中张量缩并以 einsum 表示法表示。根据求和$A_j = \sum_i B_{ij}C_i$ 的形式化定义，该运算通过沿共享索引 i 累加B_{ij} 与 C_i 的乘积，定义输出张量 A_j （或 $A(j)$）。需要注意的是，einsum 规范是声明式的——它定义的是数据依赖，而非执行流程——但它隐含了沿 i 维度进行归约。换言之，该运算会沿 i 维度进行归约，同时累加来自 B_{ij} 、由 C_i 加权的贡献。作为参考，$A(i,j) = B(i,k) * C(k,j)$ 使用声明式 einsum 表示法给出了标准的通用矩阵乘法（GeMM），而$A_{ij} = \sum_k B_{ik}C_{kj}$ 则展示了以求和形式表达的相应代数定义。

B. 张量编译器与循环降级

张量编译器，例如 TVM [1]，XLA [4]，和 MLIR [24]，充当高级数学表示法与高效机器码之间的桥梁。为了使科学家免受底层实现细节的困扰，这些框架提供了声明式领域特定语言。虽然 einsum 表示法是定义任意缩并的强大通用抽象，但现代编译器也支持对无处不在的张量运算进行直接的声明式定义，例如矩阵乘法和二维卷积。

这些编译器的基本任务是降级——将这些声明式规范自动转换为经过优化的命令式循环嵌套。算法 1 展示了内核 $A(j) = B(i, j) * C(i)$ 的降级输出的伪代码表示。即使对于稠密张量而言，这种转换也并非易事：简洁的数学表达式会扩展为一系列复杂的嵌套循环、显式的

内存寻址和边界检查。因此，生成代码的圈复杂度会随着张量的维数和操作链的深度迅速增加，从而在降级过程中产生出现故障的重大可能性。

C. 压缩数据结构

与几乎所有元素都非零的稠密张量不同，大多数真实世界中的大规模张量都非常稀疏。例如，Amazon Reviews 张量尤其庞大，包含 1.5 x 10^{19} 个分量，相当于 107 艾字节的数据（假设每个分量占用 8 字节），但只有 1.7×10^9 的分量（13 吉字节）为非零值 [6], [25]–[27]。稀疏张量通过采用坐标格式（COO）、压缩稀疏行（CSR）、压缩稀疏列（CSC） [28],或更专门的混合布局等压缩格式来表示数据，从而解决这一不平衡问题。这些格式只存储非零值及其索引元数据，使编译器能够跳过冗余迭代，避免不必要的乘法和加法运算。这对于在大规模张量工作负载上实现高效计算至关重要。

D. 稀疏张量编译器

张量代数编译器（TACO） [6] 率先实现了稀疏张量内核的自动生成，也是我们评估的主要对象。沿着这一先例，包括 MLIR [9]、TVM [8]、XLA [29], 和 PyTorch [11], [30] 在内的主要基础设施框架也引入了稀疏扩展。然而，尽管业界对此广泛关注，这些系统对稀疏运算的支持在很大程度上仍处于实验阶段。稀疏编译固有的复杂性阻碍了完全稳健且可用于生产环境的实现的开发，导致目前的测试版功能普遍缺乏正确性保证。这种普遍存在的不稳定性凸显了对专用测试框架的迫切需求，以验证不断发展的这些稀疏张量编译器。

E. 自动化测试技术

编译器验证通常依赖于强大的测试预言机与结构化输入生成之间的协同作用。差分测试和蜕变测试已成为验证通用编译器和稠密张量编译器的黄金标准 [14], [15], [17], [19], [21], [22]。从根本上说，这种方法是黑盒式的，并依赖这样一个不变量：语义等价输入——或由不同编译器实现执行的同一输入——必须产生相同的执行结果。这种简单性使其能够检测细微的功能正确性错误，而无需对编译器内部逻辑做出形式化规范。

为了开展这些差分测试和蜕变测试活动，编译器需要满足严格语法规则的高度结构化输入。基于语法的模糊测试通过遵循形式语言规范来生成语法有效的测试用例，从而解决这一问题。形式语言规范通常使用标准巴科斯-诺尔范式（BNF）或特定工具的扩展巴科斯-诺尔范式（EBNF）方言（如 ANTLR）定义。与对种子输入随机翻转比特的非结构化变异模糊测试器不同，Grammarinator [31], Lang-Fuzz [32], 和 Nautilus [33]等基于语法的工具会通过在语法推导树上随机遍历来构造输入。这种技术在测试语言处理器时尤其有效，因为它能确保生成的输入成功通过解析器，并进一步测试编译器更深层的语义分析和降级阶段。

III. 动机

```
1 整数主函数() {
2   循环(整数iC  = C1_pos[0];  iC  < C1_位置[1];iC++) {
3     整数i  = C1_crd[iC];
4     对于(整数 j  = 0; j  < B2_维度;j++) {
5       int jB  = i *B2_维度 + j;
6       A_值[j] =A_值[j] +B_值[jB] * C  _值[iC];
7     }
8   }
9 }
```

代码清单 3：稠密情形下正确的 TACO 生成程序。实现内核 $A(j) = B(i,j) \cdot C(i)$，其中 A 是稠密向量， B 是 CSR矩阵， C 是稀疏向量。

$$\underbrace{\begin{bmatrix}9\\16\\0\end{bmatrix}}_{\mathbf{A}} = \underbrace{\begin{bmatrix}0&4&0\\2&8&0\\1&0&0\end{bmatrix}}_{\mathbf{B}} \times \underbrace{\begin{bmatrix}0\\2\\5\end{bmatrix}}_{\mathbf{C}} = \begin{bmatrix}0\cdot 0+2\cdot 2+1\cdot 5\\4\cdot 0+8\cdot 2+0\cdot 5\\0\cdot 0+0\cdot 2+0\cdot 5\end{bmatrix} \quad (1)$$

为说明稀疏张量编译所面临的挑战，考虑以下内核：$A(j) = B(i,\ j) \cdot C(j)$。这里，张量 B 的每一列都与向量 C 中的元素相乘并进行归约，以计算 A中的元素。算法 1 描述了稠密张量的标准降级过程，其中元素通过直接寻址进行访问。相比之下，算法 2 展示了当 B 以 CSR 格式存储时对应的伪代码。转为稀疏存储后，编译器必须将仿射循环嵌套替换为复杂的、数据依赖的不规则迭代，以遍历压缩索引数组。无论底层格式如何，该计算都应产生方程 1 中所示的确定性结果。

为展示输出存储格式对代码生成的影响，代码清单 1 和代码清单 3 展示了 TACO 针对该内核生成的代码，其中输出张量采用了不同的存储格式 A。稠密输出变体能够正确执行，而稀疏输出变体会产生错误结果。TENSURE 揭示的这一缺陷源于循环初始化中的错误编译。如代码清单 1（第 2 行和第 7 行）所示，编译器未能在循环内正确重置 coordinatetracker j A。代码清单 2 展示了修正后的实现：它将初始化操作移入循环体。这一示例凸显了稀疏张量编译器固有的脆弱性：即使是基本计算内核，也可能触发人工检查难以发现的错误。

```
整数主函数() {
  整数 j_A = 0;
  对于 (整数i_C=C_列_指针[0]; i_C < C_列_指针[1]; i_C++) {
    整数i = C_行_索引[iC];
    对于(整数j = 0; j < B2_维度; j++) {
      //B2_维度->B 的列轴的维度
      A_值[j_A] = 0.0;

      int j_B = i * B2_维度 + j;
      A_值[j_A] += B_值[j_B] * C_值[i_C];

      j_A++;
    }
  }
}
```

代码清单 1：TACO 生成的程序（有错误 ）

```
int main() {
  循环(整数i_C=C_col_ptr[0]; i_C < C_col_ptr[1]; i_C++) {
    int i = C_行_索引[i_C];
    整数j_A = 0;
    对于(整数 j = 0; j < B2_维度; j++) {
    //B2_维度->B 的列轴的维度
      整数 j_B = i * B2_维度 + j;
      A_值[j_A] += B_值[j_B] * C_值[i_C];

      j_A++;
    }
  }
}
```

代码清单 2：手动更正的程序

图 1：TENSURE 为内核 $A(j) = B(i, j) \cdot C(j)$ 检测到的关键错误编译。代码清单 1（左）展示了 TACO 生成的原始代码，其中稀疏迭代循环包含初始化错误。代码清单 2（右）展示了手动更正的实现，并突出显示了正确执行所需的具体控制流逻辑。

算法 1： 稠密张量计算 $A(j) = B(i, j) \cdot C(j)$

输入： 稠密张量 $B \in \mathbb{R}^{m \times n}$，向量 $C \in \mathbb{R}^n$

输出： 满足 $A_j = B_{ij} \cdot C_j$ 的向量 $A \in \mathbb{R}^n$

对于 $j \leftarrow 0$ **到** $n-1$ **执行**
  $A[j] \leftarrow 0$ **对于** $i \leftarrow 0$ **到** $m-1$ **执行**
    $A[j] \leftarrow A[j] + B[i][j] \cdot C[j]$

返回 A

算法 2： 稀疏张量计算（CSR） $A(j) = B(i, j) \cdot C(j)$

输入： CSR 格式的稀疏张量 B，包含数组：

- B.值：非零值 ，
- B.列索引：列索引 ，
- B.行指针：行指针；

向量 $C \in \mathbb{R}^n$

输出： 向量 $A \in \mathbb{R}^n$ 使得 $A_j = \sum_i B_{ij} \cdot C_j$

初始化 $A[j] \leftarrow 0$ 对于所有 $j = 0, \ldots n-1$
当 $i \leftarrow 0$ **小于** $m-1$ **时执行**
  当 $k \leftarrow B$.行指针 **小于** B.行指针**时执行**
    $j \leftarrow B$.列索引$[k]$
    $A[j] \leftarrow A[j] + B$.值$[k] \cdot C[j]$

返回 A

要解决这些失败案例，需要自动化的蜕变测试。在稠密张量编译器领域，先前的工作采用了基于计算图的模糊测试 [21], [22]。这种方法会构造大量张量运算序列，并对图拓扑和单个运算进行变异，以生成语义等价项。然而，出于三个原因，这种方法并不适用于 STC。首先，图管理的开销会导致难以承受的延迟，通常将吞吐量限制在每秒约四次变异体执行 [22],，从而限制测试覆盖率。其次，图级变异会产生臃肿的失败案例，需要耗费大量计算资源进行最小化，才能隔离根因。最后，这些工具受限于固定的高级算子库，因此难以对通用 einsum 表示法定义的任意迭代模式进行压力测试。

此外，更广泛的一类 STC 将张量运算视为彼此独立的降级目标。编译过程类似于宏展开：每个 einsum 表达式都会被转译为独立的循环嵌套，而不考虑更广泛的数据流上下文。这一设计特点使基于图的生成方法不适用于蜕变测试，因为对序列中某个运算的修改不会影响其相邻运算的降级策略。因此，错误的搜索空间实际上按运算划分。由此，模糊测试器若生成由相互独立的运算组成的长链，就只会反复测试相同的独立展开机制，从而验证了在这一领域进行图级变异是冗余的。

一种潜在的替代方案是使用通用的基于语法的模糊测试器，例如 Grammarinator [31], ，直接合成 einsum 表达式。然而，einsum 表示法的有效性受上下文相关约束的制约。标准的基于语法的模糊测试器通常基于上下文无关文法运行，缺乏强制执行这些交叉引用约束所需的语义感知能力。因此，它们会产生大量无效内核，这些内核会被编译器前端拒绝，无法进入关键的循环降低阶段。

这些局限性凸显了专用测试框架的必要性。对 STC 进行有效验证，需要一个能够（1）生成有效的 einsum 表达式，（2）系统地变异这些表达式以生成语义等价的程序，以及（3）利用变形关系，在没有参考编译器的情况下检测语义偏差的系统。

算法 3： 随机爱因斯坦求和表达式生成

输入： N：输入张量的数量，
$R_{\max}$：最大张量秩，
I：指标标签池

输出： 随机 einsum 表达式
$A = B_1 * B_2 * \cdots * B_N$

步骤： 1：为输入张量分配指标

对 $k \leftarrow 1$ **到** N **执行**
  $r_k \leftarrow$ 均匀随机$(1, R_{\max})$
  B_k.索引 $\leftarrow$ 随机选取 r_k 个不同的索引从 $\mathcal{I}$
  更新所选索引的使用次数

步骤： 2：确定输出张量索引

$\mathcal{O} \leftarrow$ 使用次数非零的索引随机子集

步骤： 3：确保有效的归约索引

对于每个索引 $i \in \mathcal{I} \setminus \mathcal{O}$，其使用次数为 $= 1$ **执行**
  将 i添加到另一个输入张量中，以确保其出现至少两次

步骤： 4：分配张量形状

对每个索引 $i \in \mathcal{I}$ **执行**
  赋予随机维度 d_i

对每个张量 B_k **执行**
  对每个 j 属于 B_k.索引 **执行**
    B_k.形状$[j] \leftarrow d_{B_k\text{.索引}[j]}$

步骤： 5：构造 einsum 表达式

$A(\mathcal{O}) = \prod_{k=1}^{N} B_k(B_k\text{.索引})$

返回 A 和 $\{B_1, \ldots, B_N\}$

IV. 设计与实现

不同于稠密张量编译器通过构建全局计算图来管理一系列运算中的降级和优化，稀疏张量编译器采取了不同的方法。稀疏张量编译器并不维护涵盖整个运算序列的类图结构，而是专注于为单个 einsum 表达式独立构建一个迭代图。因此，稀疏张量编译器的复杂性在于为各个运算合成循环嵌套，而不是在多节点序列上执行图级重写。在这一背景下，串联多个 einsum 运算序列对测试的收益递减，因为复杂性在于迭代的合成，而不在于图拓扑。因此，我们的模糊测试器有别于基于图的生成；相反，它合成单个 einsum 表示法来表示复杂的张量运算，详见算法 3。

A. 随机内核生成

虽然 einsum 方程在语法上看似简单，但其有效性依赖于超出上下文无关文法表达能力的上下文相关约束。一条基本的有效性规则是，输出索引集合 $\mathcal{O}$ 必须是所有输入索引的并集 $\bigcup \mathcal{I}_{\text{in}}$ 的子集，即 $\mathcal{O} \subseteq \bigcup \mathcal{I}_{\text{in}}$。因此，这意味着输出符号是否有效取决于输入序列中先前消耗的特定符号。这种依赖关系违背了上下文无关语言的泵引理 [34]，因此有效的 einsum 表达式应归类为上下文敏感语言。

为合成结构有效的 einsum 表达式，我们将生成过程视为一个涉及索引连通性和维度一致性的约束满足问题。首先，我们从全局索引池 $\mathcal{I}$中采样，为所有输入张量（B_1，B_2，……，B_N）填充索引集合，并严格限制每个张量的最大秩为 $R_{\max}$。随后，我们将已使用的索引划分为两组：输出索引 $\mathcal{O}$（保留维度）和收缩索引 $S = \mathcal{I} \setminus \mathcal{O}$。Einsum 语义中的一项关键有效性约束是，收缩索引通常连接跨张量的维度。如果某个索引 $s \in S$ 出现于少于两个输入张量中，则该收缩的定义不明确。为解决这一问题，我们强制确保连通性：遍历 S，并将任何代表不足的收缩索引注入额外随机选择的输入张量中。一旦符号结构得到验证，我们便将抽象索引映射为具体的运行时维度。通过为I 中的每个唯一索引分配不同的整数大小，并将这些大小传播到（B_1，B_2，……，B_N）的相应轴上，我们保证所生成的张量具有与指定收缩兼容的形状。

B. 变异算子

然而，将搜索空间限制在单个 einsum 运算上，便无法使用传统的图级重写规则作为变异算子。为此，我们引入两种基于不变性的变异策略：代数交换性和 存储格式异质性。

代数交换性建立在张量代数之上：假设张量元素属于交换环（例如，实数域 $\mathbb{R}$），则乘法运算满足交换律（即，$\forall a,\ b \in \mathbb{R},\ a \cdot b = b \cdot a$）。这一性质也适用于 einsum 运算，其中操作数的顺序不会改变语义结果。例如，表达式$A(i,\ j)\ =\ B(i,\ k)\ *\ C(k,\ j)$ 在语义上等价于$A(i,\ j)\ = C(k,\ j) * B(i,\ k)$。通过对操作数进行置换，我们迫使编译器为同一数学运算生成不同的迭代调度。

第二种变异算子，存储格式异质性，旨在应对不同压缩存储格式下稀疏迭代方案的复杂性。在稀疏张量编译中，内核的数学定义与其数据的物理布局相互独立。张量 $B(i,\ j)$ 无论以 COO、CSR、CSC 还是其他压缩存储格式存储，其非零值及对应坐标都完全相同。然而，如图 1 所示，格式的选择会显著改变代码生成路径。我们利用这种解耦，在生成的 einsum 表达式中随机为每个输入张量和输出张量分配不同的存储格式。这会迫使编译器针对每种格式组合生成独特的迭代图和循环嵌套。由于计算语义保持不变，因此不同格式排列下的输出若出现任何差异，就表明编译器在处理特定访问模式或压缩数据格式时存在故障。

通过组合这两种变异算子——调整操作数顺序和改变存储格式——我们可以从单个 einsum 表达式生成丰富多样且语义等价的测试程序。

C. 语言无关架构

尽管我们的变异算子能够生成丰富的抽象测试程序，但要执行这些程序，就必须应对 STC 生态中的语法碎片化问题。由于大多数 STC 都是在高级宿主语言中运行的宏展开器（例如，TACO 使用 C++ ，Finch 使用 Julia），直接生成代码的方法并不天然具备可移植性。为确保广泛兼容，我们设计了一个通用的 JSON 抽象层，用于描述合成内核的所有语义细节，包括随机化存储格式配置。此设计将集成工作从模糊测试器转移到一个轻量级抽象层：编译器开发者只需实现一个轻量级转换器，解析此 JSON 架构并生成相应的领域专用稀疏张量程序，即可集成其工具。

图 2 展示了我们模糊测试器解耦的执行工作流。核心模糊测试引擎与目标 STC 隔离，仅通过抽象层进行交互。运行时，模糊测试器首先通过转换器将随机选取的参考程序分派给目标 STC。执行成功后，它会生成语义等价的变异体——应用交换律算子和存储格式异质性算子——并在同一后端上执行这些变异体。随后，模糊测试器充当蜕变测试预言机，将变异体的张量输出与参考结果进行比较。任何差异都表明 STC 的迭代方案存在正确性违规。此基于插件的架构确保系统能够扩展，以适应未来的研究和工业编译器。

D. 程序执行与错误检测

执行张量内核不仅需要代码，还需要有效的输入数据。在生成程序的同时，TENSURE 会自动合成小型输入张量，并确保其严格遵循 einsum 规范的 维度一致性 。参考程序成功完成执行后，其输出会被记录为真实值。随后执行变异后的程序，并将其结果与该基准进行比较。如果检测到差异，TENSURE 会将输入张量、两个程序变体及各自的输出汇总成一份详细的错误报告。

V. 评估

为了通过实证量化我们生成策略的优势，我们采用 *Grammarinator* [31],建立了一个基线方法。Grammarinator 是一种先进的基于语法的模糊测试器，配置了用于 einsum 表示法的标准上下文无关文法。我们开展了一项大规模对比研究，使用基线方法和我们定制的生成器合成了一个包含一百万个（10^6）内核的语料库。

表 I：TENSURE 在为期 6 小时的模糊测试活动中发现的错误。

STCs	#迭代	STC 不可接受输入	C 类错误	错误代码漏洞
TACO	267k	236k	14.4k	5,758
Finch	1,619	7	57	0

注： STC 不可接受输入：STC 的不可接受输入；C 类错误：崩溃漏洞数量；错误代码漏洞：错误代码漏洞（静默错误）数量。“#迭代次数”表示模糊测试迭代总数。

结果显示，生成效率存在显著差异。我们的基于约束的算法通过构造保证有效率达到 100%，而基于语法的基线方法仅在 ∼3.3% 的尝试中生成了有效内核。由于维度不一致，绝大多数由语法生成的输入都遭到拒绝。这一结果证实，通用语法模糊测试器在张量编译器测试中效率低下，因为它们将超过 ∼96% 的模糊测试预算浪费在语法有效、语义却不正确的输入上。

我们将基于约束的 einsum 生成器集成到 TENSURE 框架中，作为核心输入合成引擎。为了评估其有效性，我们开展了为期六小时的模糊测试活动，针对两款最先进的稀疏张量编译器：TACO 和 Finch。结果见表 I。

在模糊测试生命周期中，TENSURE 会将执行异常归为三种诊断确定性各不相同的故障场景。第一种场景是初始参考内核执行失败；由于许多 STC 仅支持 einsum 标准的严格子集，此类拒绝通常源于不支持的特性，而非真正的缺陷。第二种场景是参考内核成功执行，但语义等价的变异体触发了编译错误或运行时崩溃。虽然这通常表明存在有效的崩溃漏洞，但也可能是编译器对特定迭代方案支持不足所致。第三种，也是最关键的场景，是参考程序和变异程序均成功执行，却产生不一致的输出。这构成了功能正确性违规，也就是静默错误编译。与通常能被运行时断言捕获的崩溃不同，这种输出差异表明编译器为有效的数学运算生成了错误的稀疏迭代方案。

A. TACO 评估结果

在编译器前端接受的有效 einsum 运算中，TENSURE 在约 ∼65.2% 的案例中发现了缺陷。关键的是，这些故障中有 ∼18.6% 被归类为严重错误编译（错误代码缺陷），即编译器在不发出提示的情况下生成了错误代码。

我们将如此大量的缺陷归因于两个主要因素。首先，报告中的数字反映的是故障总数；若要确定不同的根本原因，就必须进行全面的人工检查，而鉴于故障规模之大，这并不可行。其次，TACO 主要被设计为一个基础研究原型，用于展示稀疏编译概念，

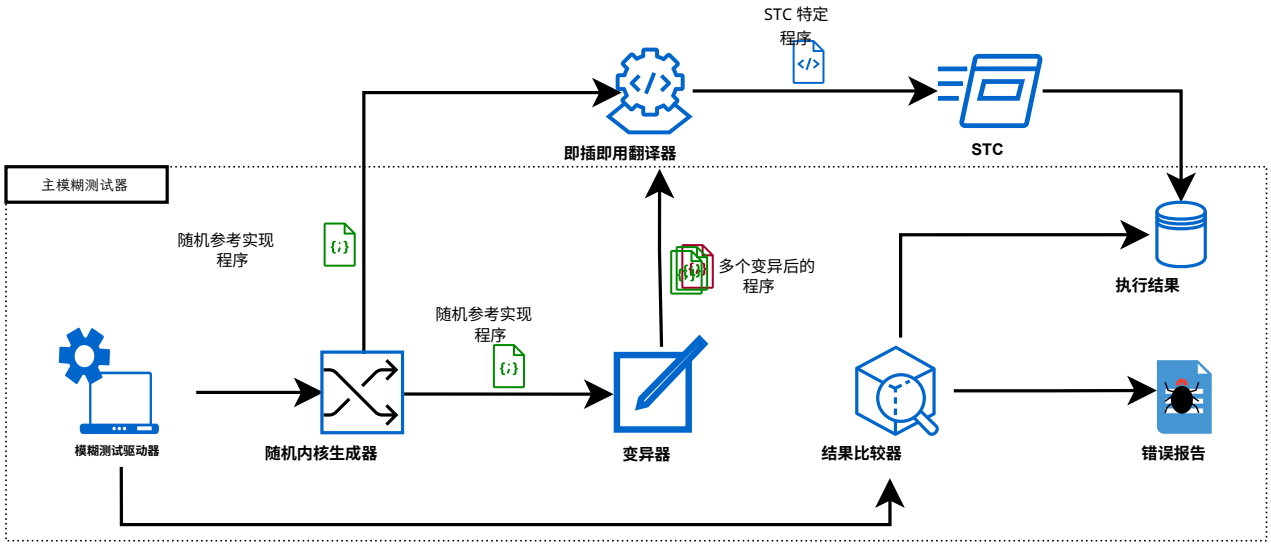

图 2：主要的模糊测试循环。系统会生成一个随机张量程序及几个变异变体。两个版本都会经过编译并执行，然后比较输出，以识别表明存在编译器缺陷的不匹配。

而非生产级稳健系统。由于该项目已不再积极维护，也不再接收错误报告，我们将分析重点放在总体故障率上，以展示模糊测试器的有效性，而没有为报告目的进行细粒度的去重。由于 TENSURE 具有黑盒特性，可靠地去重本质上十分困难，因为同一个底层编译器缺陷可能由多个语法形式不同的 einsum 表达式触发。

B. Finch 评估结果

在针对 TACO 的模糊测试活动中，我们评估了超过 267,000 个内核，并发现了数千个缺陷；而对 Finch 的评估受到显著编译延迟的限制，在同样的六小时窗口内仅完成了 1,619 次迭代。尽管吞吐量有所降低，TENSURE 仍成功在 Finch 编译器中发现了 57 个崩溃缺陷。在这一有限样本中未检测到静默错误编译，不过，能够导致崩溃的输入证实了模糊测试器具备对基于 Julia 的基础设施进行压力测试的能力。

两个因素可以解释这种差异。首先，也是最重要的一点，Finch 内核的编译时间是主要瓶颈。如图 3 所示，TACO 处理单个内核约需 64 毫秒，而 Finch 每个内核需要超过 181 秒。稀疏张量复杂的数据布局给初始化和计算编译阶段带来了显著开销，而且每次迭代都会重复产生这项开销。其次，本次评估仅限于初步的六小时窗口；更大规模的纵向模糊测试活动将留待未来开展。

提高吞吐量的一种方法是降低测试变体的粒度。通过将更改限制在特定表达式，我们可以让 Julia 编译器只重新编译

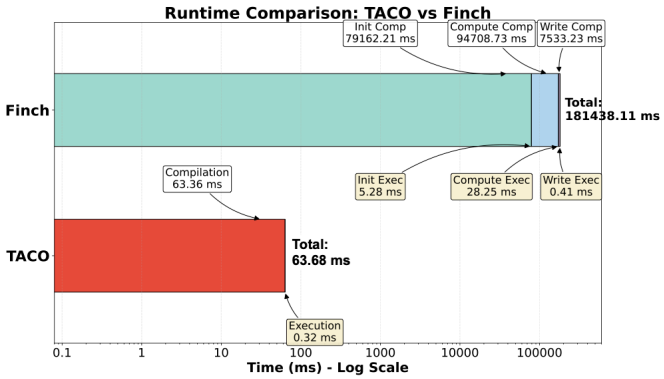

图 3：运行时间比较：TACO 与 Finch。时间轴采用对数刻度，以展现编译时间的差异。

被变异的部分，而不是整个程序。不过，这需要对模糊测试器进行结构性更改和专门化。

VI. 讨论与未来工作

验证数值软件时，一个长期存在的挑战是处理浮点确定性。尽管我们的变异算子利用了 Einsum 运算的交换性，但这种高层次的重排序往往会迫使编译器改变底层的归约调度。由于 IEEE-754 浮点运算不满足结合律，这些调度变化会引入轻微的数值偏差。目前，TENSURE 采用一种基于 ϵ 的宽松比较器来容忍这种噪声；然而，这也带来了权衡：过宽的容差可能会掩盖细微的正确性缺陷。未来，我们将通过引入纯整数测试模式来缓解这一问题

以严格验证合成的控制流。这种方法消除了数值噪声，从而能够明确区分有效的算法重排序与真正的错误编译。

除了应对精度方面的挑战外，系统性地利用张量转置也是增强模糊测试器的一种有前景的方法。目前，我们的策略依赖于操作数交换律和存储格式异质性。然而，利用张量缩并在索引置换下的代数不变性，我们可以引入一类新的转置变异算子。这些算子将通过置换输入张量的维度来转换输入张量（实际上计算 A^T），同时调整缩并索引以保持语义等价。这种有针对性的方法将严格检验编译器处理置换访问模式的能力，从而暴露出简单的操作数重排序无法触及的索引映射生成逻辑缺陷。

此外，我们没有针对冗余错误报告（第 V-A 节）采用高级去重策略，也没有实施特定编译器优化来加快 Finch 等目标的编译时间（第 V-B 节）。应对这些挑战需要检查内部编译器状态，或针对特定后端定制生成器；这些修改将从根本上改变 TENSURE 的性质，使其从通用黑盒模糊测试器变为专用的灰盒或白盒工具。

在未来的工作中，我们计划通过将 TENSURE 与更广泛的先进基线方法进行基准测试 [33], [35], [36]来扩展评估范围，从而进一步验证其有效性。最后，我们旨在通过集成 MLIR 稀疏方言和 PyTorch Sparse 等新兴平台，展示我们框架的可扩展性。由于 PyTorch Sparse 原生支持 einsum，集成工作较为直接；而针对 MLIR 稀疏方言则存在显著的语义差距，因为该方言无法直接处理高层 einsum 声明。因此，这种集成需要增强我们的抽象层，将基于 JSON 的中间表示转换为 MLIR 稀疏方言，从而实现对不同稀疏编译器基础设施的稳健比较分析。

VII. 相关工作

自动化编译器验证是一个发展成熟的研究领域。Csmith 的奠基性工作 [14]展示了随机差分测试的有效性：通过从头生成有效的 C 程序，在 GCC 和 LLVM 中发现了数百个漏洞。在此基础上，Orion [17] 和 YARPGen [15] 等后续框架引入了复杂的基于变异的策略。这些工具通过注入死代码或简化算术表达式，创建语义等价的测试用例。

深度学习编译器的蓬勃发展推动了专用测试框架的开发。NNSmith [21] 开创了基于约束的有效计算图生成方法，擅长验证多样的图拓扑结构和算子组合。在此基础上，PolyJuice [22] 采用等式饱和来合成——语义等价的图变体，专门针对图级优化器中的缺陷。与此同时，HiraGen [37] 则通过系统地变异计算图，着重测试高级优化遍。

近年来，大语言模型（LLM）已被应用于输入生成。TitanFuzz [38]以黑盒方式利用大语言模型合成算子序列，无需编译器知识。相比之下，WhiteFox [39]引入了一种白盒方法，利用大语言模型分析编译器源代码和优化过程，以生成有针对性的测试输入。然而，所有这些框架主要针对由标准算子构成的稠密计算图，而稀疏张量编译和降级阶段所特有的挑战在很大程度上仍未得到解决。

稀疏编译领域正在快速发展，其驱动力在于需要将算法规范与数据表示解耦。TACO [6] 率先提出了格式无关编译的概念，引入基于格的理论，为任意稀疏张量格式合成代码。这项奠基性工作随后被现代框架所采用，包括 MLIR 的稀疏张量方言 [9], Finch [7], 和 TVM 的 SparseTIR [8],, 以及 XLA [29] 和 PyTorch [11], [30]的工业扩展。随着这些系统从研究原型转变为关键的生产基础设施，对稳健且专业化的测试框架的需求也日益迫切。

基于语法的模糊测试是测试语言处理器的一种标准技术。Grammarinator [31],LangFuzz [32], 和 Nautilus [33]通过遍历形式文法生成输入，在多种语言的语法测试中展现出极高的有效性。然而，通用的基于语法的模糊测试器通常基于上下文无关文法运行，缺乏内在机制来强制执行 einsum 表示法中的稀疏张量程序所需的语义约束。

VIII. 结论

本文提出了 TENSURE，这是首个专为稀疏张量编译器（STC）的自动化测试而设计的可扩展黑盒模糊测试框架。与受限于固定算子库 [21], [22], 的先前框架不同，TENSURE 将 einsum 表示法作为完全通用的输入抽象，可用于合成任意张量缩并。

我们通过针对两个不同的稀疏张量编译器（STC）——TACO 和 Finch——展示了 TENSURE 的实用性。我们对 TACO 的评估揭示了严重的稳健性问题：模糊测试器在超过 ∼60% 的生成测试用例中发现了崩溃或错误编译。这些发现，以及成功适配 Finch 生态系统的结果，共同凸显了当前稀疏编译领域的脆弱性，并强调了 TENSURE 等专用工具对于确保下一代基础设施可靠性的必要性。

参考文献

[1] T. Chen、T. Moreau、Z. Jiang、L. Zheng、E. Yan、M. Cowan、H. Shen、L. Wang、Y. Hu、L. Ceze、C. Guestrin 和 A. Krishnamurthy，《TVM：用于深度学习的自动化端到端优化编译器》，arXiv，2018 年 2 月。[2]《NVIDIA TensorRT》，11 月。 2025, [在线；访问日期：11 月 17 日。 2025]。[在线]。可用地址：https://developer.nvidia.com/tensorrt [3] P. Tillet、H. T. Kung 和 D. Cox，《Triton：用于分块神经网络计算的中间语言和编译器》，MAPL 2019 系列。美国纽约州纽约市：计算机协会，2019 年，第 10–19 页。在线。可用地址：https://doi.org/10.1145/3315508.3329973。[4] 2024, [17 2025] [在线]。可用地址：https://doi.org/。/。《XLA》，12 月。在线；访问日期：11 月。在线]。可用地址：https://openxla.org/xla [5] L. B. Kish，《摩尔定律的终结：微纳电子学中热噪声导致的集成死亡》，《物理快报 A》，第 305 卷，第 3–4 期，第 144–149 页，2002 年。[6] F. Kjolstad、S. Kamil、S. Chou、D. Lugato 和 S. Amarasinghe，《张量代数编译器》，ACM 程序设计语言期刊，第 1 卷，OOPSLA 期，2017 年 10 月。 [在线]。可用地址：https://doi.org/10.1145/3133901[7] W. Ahrens、T. F. Collin、R. Patel、K. Deeds、C. Hong 和 S. Amaras- inghe，《Finch：带控制流的稀疏结构化张量编程》，ACM《编程语言会议论文集》，第 9 卷，OOPSLA1 期，第 1042–1072 页，2025 年。[8] Z. Ye、R. Lai、J. Shao、T. Chen 和 L. Ceze，《SparseTIR：深度学习中稀疏编译的可组合抽象》，ASPLOS 2023 系列。美国纽约州纽约市：计算机协会，2023 年，第 660–678 页。在线。可用地址：https://doi.org/。/。A. Bik、P. Koanantakool、T. Shpeisman、N. Vasilache、B. Zheng 和 F. Kjolstad，《MLIR 中对稀疏张量计算的编译器支持》，ACM《体系结构、代码优化与编译器学报》，第 19 卷，第 4 期，第 1–25 页，2022 年 9 月。[10] A. J. C. Bik 和 H. A. G. Wijshoff，《稀疏矩阵计算的编译技术》，载于第 7 届国际超级计算会议论文集，ICS '93 系列。美国纽约州纽约市：计算机协会，1993 年，第 416–424 页。 [在线]。可用地址：https://doi.org/10.1145/165939.166023。/。《参考 API - PyTorch Sparse》，11 月。在线；访问日期：11 月 20 日。 2025, [2025]。 [在线]。可用地址：https://docs.pytorch.org/docs/stable/ sparse.html [12] N. Sedaghati、T. Mu、L.-N. Pouchet、S. Parthasarathy 和 P. Sadayappan，《在 GPU 上自动选择稀疏矩阵表示》，载于第 29 届 ACM 国际超级计算会议论文集，ICS '15 系列。美国纽约州纽约市：计算机协会，2015 年，第 99–108 页。[在线]。可用地址：https://doi.org/10.1145/2751205.2751244。。《近期三个问题的事后分析》，9 月。 2025, [在线；访问日期：11 月 18 日。 2025]。 [在线]。可用地址：https://www.anthropic.com/engineering/a-postmortem-of-three-recent-issues [14] X. Yang、Y. Chen、E. Eide 和 J. Regehr，《发现并理解 C 编译器中的错误》，载于第 32 届 ACM SIGPLAN 编程语言设计与实现会议论文集，PLDI '11 系列。美国纽约州纽约市：计算机协会，2011 年，第 283–294 页。 [在线]。可用地址：https://doi.org/10.1145/1993498.1993532[15] V. Livinskii、D. Babokin 和 J. Regehr，《使用 YARPGen 对 C 和 C++ 编译器进行随机测试》，ACM 程序设计语言期刊，第 4 卷，OOPSLA 期，2020 年 11 月。 在线]。可用地址：https://doi.org/10.1145/3428264。/ D. Xiao、Z. LIU、Y. Yuan、Q. Pang 和 S. Wang，《深度学习编译器的蜕变测试》，ACM《测量、分析与计算系统》，第 6 卷，第 1 期，2022 年 2 月。 [在线]。可用地址：https://doi.org/10.1145/3508035。/ V. Le、M. Afshari 和 Z. Su，《通过输入模等价进行编译器验证》，SIGPLAN Not.，第 49 卷，第 6 期，第 216–226 页，2014 年 6 月。 [ ] 10.1145/2666356.2594334 在线。可用地址：https://doi.org/。/。A. F. Donaldson、H. Evrard、A. Lascu 和 P. Thomson，《图形着色器编译器的自动化测试》，ACM 程序设计语言期刊，第 1 卷，OOPSLA 期，2017 年 10 月。 [在线]。可用地址：https://doi.org/10.1145/3133917[19]《CLSmith》，11 月。 2025, [在线；访问日期：11 月 20 日。 2025]。 [在线]。可用地址：https://github.com/ChrisLidbury/CLSmith [20] A. Einstein，《广义相对论的基础》，1916年。

[21] J. Liu、J. Lin、F. Ruffy、C. Tan、J. Li、A. Panda 和 L. Zhang，"NNSmith：为深度学习编译器生成多样且有效的测试用例"，载于《第 28 届 ACM 编程语言与操作系统架构支持国际会议论文集》第 2 卷，丛书 ASPLOS 2023。美国纽约州纽约市：计算机协会，2023 年，第 530–543 页。 [在线]。网址：https://doi.org/10.1145/3575693.3575707。/。[22] C. Zhou、B. Qian、G. Go、Q. Zhang、S. Li 和 Y. Jiang，"PolyJuice：通过基于等式饱和的重写检测张量编译器中的错误编译缺陷"，ACM 程序设计语言期刊，第 8 卷，第 OOPSLA2 期，2024 年 10 月。 [在线]。网址：https://doi.org/10.1145/3689757。/ G. Ricci 和 T. Levi-Civita，"绝对微分计[23]算方法及其应用"，数学年刊，第 54 卷，第 1–2 期，第 125–201, 1901 [ ] 157997 – 页。在线。[24] 网址：http://eudml.org/doc/ C. Lattner、M. Amini、U. Bondhugula、A. Cohen、A. Davis、J. Pienaar、R. Riddle、T. Shpeisman、N. Vasilache 和 O. Zinenko，"MLIR：扩展领域特定计算的编译器基础设施"，载于《2021 IEEE/ACM 代码生成与优化国际研讨会（CGO）》，2021 年，第 2–14 页。[25] R. M. Bell 和 Y. Koren，"从 Netflix 奖挑战赛中汲取的经验"，《ACM SIGKDD 探索通讯》，第 9 卷，第 2 期，第 75–79 页，2007 年。[26] B. Viswanath、A. Mislove、M. Cha 和 K. P. Gummadi，"Facebook 用户交互的演变"，载于《第 2 届 ACM 在线社交网络研讨会论文集》，2009 年，第 37–42 页。[27] J. McAuley 和 J. Leskovec，"隐因子与隐主题：利用评论文本理解评分维度"，载于《第 7 届 ACM 推荐系统会议论文集》，2013 年，第 165–172 页。[28] I. S. Duff、A. M. Erisman 和 J. K. Reid，《稀疏矩阵的直接方法》。牛津大学出版社，2017 年。[29] "使用稀疏张量"，2024 年 10 月。 2024, [在线；访问日期：11 月 20 日。2025]。 [在线]。网址：https://www.tensorflow.org/guide/sparse tensor[30] B. Yan、A. J. Root、T. Gale、D. Broman 和 F. Kjolstad，"Scorch：稀疏深度学习库"，arXiv，2024 年 5 月。[31] R. Hodov´an、A. Kiss 和 T. Gyim´othy，"Grammarinator：基于语法的开源模糊测试器"，载于《第 9 届 ACM SIGSOFT 自动化测试用例设计、选择与评估国际研讨会论文集》，丛书 A-TEST 2018。美国纽约州纽约市：计算机协会，2018 年，第 45–48 页。 [在线]。网址：https://doi.org/10.1145/3278186.3278193。/ . C. Holler、K. Herzig 和 A. Zeller，"使用代码片段进行模糊测试"，载于《第 21 届 USENIX 安全研讨会会议论文集》，丛书 Security'12。美国：USENIX 协会，/2012 年，第 38 页。[33] C. Aschermann、T. Frassetto、T. Holz、P. Jauernig、A.-R. Sadeghi 和 D. Teuchert，"Nautilus：利用语法捕捉深层缺陷"，载于 NDSS，第 19 卷，2019 年，第 337 页。[34] H.-J. Kreowski，"上下文无关图语言的泵引理"，载于《图语法及其在计算机科学与生物学中的应用》，V. Claus、H. Ehrig 和 G. Rozenberg 编。柏林、海德堡：Springer Berlin Heidelberg，1979 年，第 270–283 页。[35] D. Steinh¨ofel 和 A. Zeller，"输入不变量"，载于《第 30 届 ACM 联合欧洲软件工程会议暨软件工程基础研讨会论文集》，丛书 ESEC/FSE 2022。美国纽约州纽约市：计算机协会，2022 年，第 583–594 页。在线。可用地址：https://doi.org/10.1145/3540250.3549139。[36] 4 在线。网址：https://doi.org/。/ . C. S. Xia、M. Paltenghi、J. Le Tian、M. Pradel 和 L. Zhang，"模糊测试一切：使用大语言模型进行通用模糊测试"，载于《IEEE/ACM 第 46 届软件工程国际会议论文集》，丛书 ICSE '24。美国纽约州纽约市：计算机协会，2024 [ ] 10.1145/3597503.3639121。在线。网[37] 址：https://doi.org/。/ . H. Ma、Q. Shen、Y. Tian、J. Chen 和 S.-C. Cheung，"使用 Hirgen 对深度学习编译器进行模糊测试"，载于《第 32 届 ACM SIGSOFT 软件测试与分析国际研讨会论文集》，丛书 ISSTA 2023。美国纽约州纽约市：计算机协会，2023 年，第 248–260 页。 [在线]。网址：https://doi.org/10.1145/3597926.3598053。/ . Y. Deng、C. S. Xia、H. Peng、C. Yang 和 L. Zhang，"大语言模型是零样本模糊测试器：利用大语言模型对深度学习库进行模糊测试"，载于《第 32 届 ACM SIGSOFT 软件测试与分析国际研讨会论文集》，2023 年，第 423–435 页。[39] C. Yang、Y. Deng、R. Lu、J. Yao、J. Liu、R. Jabbarvand 和 L. Zhang，"WhiteFox：由大语言模型赋能的白盒编译器模糊测试

模型”，*ACM* 程序设计语言期刊，第 8 卷，OOPSLA2 期，2024年10月。[在线]。获取地址：https://doi.org/10.1145/3689736