代码神经模型中的干扰项研究

Md Rafiqul Islam Rabin*mrabin@uh.edu*休斯敦大学，休斯敦，德克萨斯州，美国

Aftab Hussain*ahussain27@uh.edu*休斯敦大学，休斯敦，德克萨斯州，美国

Sahil Suneja*suneja@us.ibm.com*IBM 研究院，约克镇高地，纽约州，美国

Mohammad Amin Alipour*maalipou@central.uh.edu*休斯敦大学，美国得克萨斯州休斯敦

arXiv:2303.01739v1 [cs.LG] 3 Mar 2023

摘要—在可解释人工智能领域，寻找影响神经模型预测的重要特征是一项活跃的研究方向。神经模型是不透明的，找出此类特征有助于更好地理解其预测。与此相反，本文从相反的视角探讨干扰特征：这类特征会影响模型对其预测的置信度，从而引发对预测结果的质疑。理解干扰项能为神经模型预测中特征相关性提供一种互补的视角。

本文采用一种基于归约的技术来寻找干扰项，并初步报告其影响和类型。我们在多种代码任务、模型和数据集上进行的实验表明，移除词元会显著影响模型对其预测的置信度，而词元类别也会对模型置信度发挥重要作用。本研究旨在通过强调那些会显著影响模型置信度的词元，提升模型的透明度。

索引词—**可解释人工智能、干扰项、代码模型**

I. 引言

深度学习越来越多地被应用于软件工具，以分析现有的编程数据并产生新的见解。深度学习的应用范围涵盖用于预测程序属性的判别模型 [1]，例如类型预测、错误检测和克隆检测，也涵盖更为复杂的生成式语言模型，这些模型输出字符字符串 [2]，例如代码生成、代码摘要和代码搜索。尽管近年来深度学习模型在各种软件工程任务中的应用取得了长足进展，但这些模型仍然不透明，人们对于它们为何做出某一预测以及哪些输入特征会影响它们仍知之甚少，从而阻碍了其可信度。

近期研究表明，代码智能神经模型高度依赖未经筛选、且有时存在重复的训练数据 [3, 4]，并且通常难以在已知训练场景之外的真实世界数据上实现良好的泛化性能 [5, 6]。为揭示模型的学习机制，一些研究采用特征选择 [7, 8]、代码扰动和变异 [9, 10, 11], 以及程序简化 [12, 13], 等技术，探讨代码智能模型的可解释性。这些工作旨在检测对模型预测有积极贡献的最重要输入特征子集。与之相反，本研究聚焦于逆向问题：哪些特征会使人对某项预测产生怀疑，或可能降低模型的置信度？更准确地说，在代码智能领域，我们将干扰项词元定义为：移除这些词元后，模型对其预测的置信度会发生显著变化。

图1说明了干扰项对 Code2Seq 模型置信度的影响 [14]，以此作为一个激励性示例。从输入程序中省略关键字令牌 throw 和 new，会显著降低模型预测的概率分数，表明模型在预测过程中的置信度出现下降。同样，省略变量名令牌 otherCast 和成员访问点运算符，会使预测概率分数上升。该示例表明，某些词元可能对模型的置信度产生巨大影响。为了更好地理解这种影响，有必要研究不同词元如何影响模型预测。本研究通过考察各种词元对模型预测的影响，在这一方向上迈出了第一步。通过了解哪些词元会对模型置信度产生积极或消极影响，我们可以调整数据预处理和归一化，帮助模型聚焦于与任务更相关的代码结构，从而提高模型的可靠性和可信度。

在本研究中，我们提出了一种方法，用于识别并评估代码智能模型中干扰项的影响程度。我们的方法基于面向代码智能的先进程序缩减框架（SIVAND [12, 15] 和 P2IM [13, 16]）构建。该方法使用差分调试 [17] 算法，通过迭代简化输入程序，以找出最相关的输入特征。它会分析每一步缩减的输出，并识别出其缩减会显著提高或降低模型对其预测的置信度的词元。

我们针对四项任务中的七种代码智能模型和四个数据集进行实验，结果表明，在方法命名任务中，Code2Vec 模型更容易受到干扰项的影响；在代码搜索任务中，CodeBERT 模型则更依赖单个词元。此外，我们的结果表明，某些类别的词元（例如控制结构）可能会显著影响模型的置信度。尽管我们关注的是现有可解释性研究中常见的研究对象，但我们的方法也

	Programs	***Prediction probability scores before and after reduction***	
(a)	`private void addMember(AMIdentityRepository repo, AMIdentity amIdentity, AMIdentity member) throws IdRepoException { if (isMember(amIdentity)) { if (isOperationSupported(repo,getType().EDIT)) { addMember(member); } else { throw new ForbiddenException(""); } } }`	0.8631	0.7516 (A decrease of 0.1115)
(b)	`boolean equals(Object other) { if (other == null) { LazyForwardingEquality<?> otherCast = (LazyForwardingEquality<?>) other(otherCast . toBytes(),toBytes()); } }`	0.3243	0.4421 (An increase of 0.1178)

图 1：移除词元对模型预测概率分数的影响。此处，我们给出两个程序，展示在方法命名任务中移除词元（红色高亮）前后 Code2Seq 对应的预测概率分数。验证模型对这些词元的依赖程度，并与真实情况进行比较，有助于揭示模型的学习质量。

适用于其他模型架构、下游任务和目标数据集。本文的贡献如下。

- 我们提出了代码神经模型中干扰项的概念。
- 我们评估了干扰项在代码的若干任务、模型和数据集中的影响程度。
- 我们讨论了这些发现对代码神经模型调试工作的意义。

II. 方法论

本节介绍我们从神经代码智能模型中提取干扰词元的方法。为了解单个词元在预测过程中对模型置信度的影响，我们使用程序缩减算法系统地从程序中移除词元，同时保留模型的预测结果。接下来，我们研究约简程序中模型置信度的变化，以观察被移除词元的影响。

差分调试。Zeller 等人 [17] 提出了用于缩减故障诱发输入程序规模的差分调试（*ddmin*）算法。该算法的目标是找出一个能够保留输入程序相同行为的最小程序。例如，给定一个故障诱发输入程序，*ddmin* 算法会在保留故障的同时，系统地从输入程序中移除词元。它主要遵循以下步骤：

- 首先将输入拆分为一系列更小的块，称为差分块。接着，它通过从输入中移除差分块，迭代地创建并评估候选项。
- 随后，该算法检查任一生成的候选项是否保留了输入的相同行为，例如失败。若是，则将该候选项用作新的基准，以进一步缩减。否则，它会提高差分块的粒度，直到确定该输入无法再进一步缩减。
- 最终缩减后的候选项是*1*-最小的，即任何进一步缩减它的尝试都会改变输入的行为。

Rabin等人 [12, 15] 和 Suneja等人 [13, 16] 采用差分调试，通过找出与模型预测最相关的特征，来提高神经代码智能模型的透明度。在他们的方法中，他们使用 *ddmin* 算法，在保留模型预测的同时缩减输入程序的规模。假设，$y = M(p_o)$表示模型 M 对输入程序 p_o的输出为 y。他们使用*ddmin* 将 p_o 约简为 p_r，其中 p_r的规模显著小于 p_o，即 $|p_r| < |p_o|$，且 p_r 是 1-最小。在他们的方法中，他们对比原始输入程序 p_o 及其对应的 1-最小约简程序p_r ，以识别 p_o 中对模型预测贡献最大的相关特征，即 $M(p_o)$。

干扰词元。尽管几乎所有先前研究都主要关注理解与模型预测最相关的关键输入特征，但本研究关注词元对模型预测置信度的影响。预测可能保持不变，但移除某些词元可能会改变模型在预测时的置信度。为此，在模型预测保持不变的连续步骤中，我们检查：（1）是否移除了任何词元；以及（2）模型的置信度是否受到显著影响。在本研究中，如果模型预测的概率分数变化至少达到 0。1，即 10%，我们就认为影响显著。

使用*ddmin*从输入程序中移除词元的每个步骤之后，我们都会检查约简程序，观察移除操作是否对模型预测产生了明显影响。如果预测概率分数上升或下降，而目标预测标签保持不变，我们就将其视为剧烈概率变化。随后，我们会收集这些被移除的词元，作为目标预测的潜在干扰项词元。

我们对*ddmin* 的约简步骤作如下调整：

• 给定一个输入程序 p_o 和一个代码智能模型M，我们记录该模型 M 在输入程序 p_o上给出的预测标签 y_o 和概率分数s_o 。

$$y_o, s_o = M(p_o)$$

• 接着，我们使用 *ddmin*，通过从输入程序 p_o 中移除一些词元 t_r 来生成一个约简程序 p_r ，同时保留相同的预测标签 y_o。然后，我们计算模型 M 在约简程序 p_r上的预测标签 y_r 和概率分数 s_r 。

$$p_r, t_r = ddmin(p_o)$$
$$y_r, s_r = M(p_r)$$

• 最后，我们比较输入程序的概率分数 s_o 与约简程序的概率分数 s_r ，以了解词元的影响 t_r。如果移除词元会显著改变模型的概率分数（$\Delta s = |s_r - s_0|$），那么这些词元就被认定为干扰词元。

$$\text{distractor tokens}, t_r = \begin{cases} s_r \gg s_0 : +\Delta s \\ s_r \ll s_0 : -\Delta s \end{cases}$$

当*ddmin* 算法将原始输入程序（p_o）约简为终结的1-最小程序（p_r）时，它会生成若干保留原始预测（M（p_o））的中间约简程序。假设对于某个样本，需要经过 n 个这样的中间约简程序才能得到最终约简程序。随后，对于每个输入程序，我们计算平均概率变化为 $\frac{1}{n}\sum$（$\Delta s_1 + ... + \Delta s_i + ... \mp \Delta s_n$），最大概率变化为 max($\Delta s_1$, ... Δs_i, ... Δs_n)，其中 $\Delta s_i = |s_i - s_{i-1}|$ 和 s_i 是模型在第 i^{th} 步的约简程序上给出的预测概率分数。

III. 研究对象

在本节中，我们简要介绍本研究所使用的不同任务、模型和数据集的设计与实现方面。我们针对三种编程语言（C、Java 和 Python ）的四个数据集（ s-bAbI 、 Java-Large 、 Py150-Great 和 CSN-Java），研究了四个知名任务（软件漏洞检测、方法名预测、变量误用定位与修复，以及自然语言代码搜索），并涵盖七个模型（循环神经网络、卷积神经网络、Code2Vec、Code2Seq、循环神经网络、Transformer 和 CodeBERT）。

软件漏洞检测 [13, 16], 方法名称预测 [7, 18], 变量误用定位与修复 [19, 20],以及自然语言代码搜索 [2, 21] 任务，已被广泛研究，用于评估神经代码智能模型的透明度 [4, 5, 6, 8, 10, 12, 13, 15, 16, 22, 23,24, 25, 26]。对于这些任务，我们选择了训练产物公开可用的流行模型和数据集[13, 14, 21, 27]。我们采用作者在论文或代码仓库中提供的默认基础配置，仅进行少量修改，例如更改批量大小。

A. 软件漏洞检测（VulDetection）

任务： 在软件漏洞检测任务中，目标是检测源代码中的潜在漏洞（也称为错误或缺陷）。模型通过样本进行训练，这些样本表明给定代码片段是否包含漏洞，随后模型用于预测新的代码片段是否可能包含漏洞。此任务已被用作不同应用的下游任务，例如探测 [13], 理解 [28],并增强 [16] 模型的信号感知能力和可靠性问题。

数据： 我们使用 s-bAbI [29]合成数据集执行 VULDETECTION 任务。s-bAbI 数据集包含语法有效、具有非平凡控制流和缓冲区溢出漏洞的 C 程序。该数据集由 Suneja 等人 [13]使用 s-bAbI 生成器生成（并进行了规范化），该生成器生成了约 475K 个函数。带有“UNSAFE”标签（含有合成漏洞）的样本标记为“1”，而带有“SAFE”标签（不含漏洞）的样本标记为“0”。

模型： 我们为漏洞检测任务使用两种模型：循环神经网络 [30]和卷积神经网络 [31]。这些模型采用不同的源代码表示和架构 [13]。第一个模型将源代码视为词元序列，并使用双向 GRU 构成的 2 层网络；第二个模型则使用二维卷积层将源代码编码为图像。

B. 方法名称预测（方法名预测）

任务： 在方法名称预测任务中，模型尝试根据方法体预测方法的名称。这项任务已被用作下游任务，用于评估多个最先进的代码智能模型 [14, 32]。

数据： 我们在方法名称预测任务中使用 Java-Large 数据集 [14]，其中包含来自 9500 个星标数最高的 GitHub 仓库的约 1600 万个 Java 方法。该数据集将 9000 个 Java 项目划分为训练集、250 个 Java 项目划分为验证集，并将 300 个 Java 项目划分为测试集。

模型：我们为方法命名任务使用两个模型：Code2Vec [32], 和 Code2Seq [14]。这些模型通过从方法的抽象语法树（ AST ）中提取连接一个终结符或词元与另一个终结符或词元的“路径”来完成任务。Code2Seq 使用长短期记忆网络对路径进行编码，并逐个解码目标，而不是像 Code2Vec 那样使用未分词嵌入。

C. 变量误用定位与修复（VarMisuse）

任务： 变量误用是软件开发中一种常见的错误，当程序中使用了与预期不同的变量时便会发生 [19]。在变量误用定位与修复任务中，模型会尝试定位误用错误，并提出修复方案，即给出应使用的正确标识符 [20, 27]。

数据： 我们使用 Hellendoorn 等人基于 ETH Py150 数据集 [33] 构建的 Py150-Great 合成数据集 [27]。该数据集包含来自总计 15 万个 Python 文件的函数，其中 9 万个文件作为训练集，1 万个文件作为验证集，5 万个文件作为测试集。每个函数都以无缺陷样本的形式收录，同时还被加入最多三个合成缺陷，因此总计约有 200 万个样本。

表I： 样本的平均概率变化。

Task	数据集	模型	概率增加（PI）			概率减少（PD）			样本（%）		
			min	max	mean	min	max	mean	PI	PD	PI ∪ PD
漏洞检测	s-bAbI	RNN	0.10	0.48	0.20	-0.10	-0.48	-0.19	42.10	43.50	46.10
		CNN	0.10	0.49	0.23	-0.10	-0.50	-0.24	32.20	54.00	54.70
方法命名	Java-Large	Code2Vec	0.10	**0.72**	**0.25**	-0.10	**-0.74**	**-0.27**	46.20	64.10	65.20
		Code2Seq	0.10	0.55	0.18	-0.10	-0.69	-0.22	44.90	72.70	74.80
变量误用	Py150-Great	RNN	0.10	0.45	0.22	-0.10	-0.64	-0.24	85.00	93.50	94.40
		Transformer	0.10	0.59	0.22	-0.10	-0.60	-0.23	79.80	95.30	95.60
代码搜索	CSN-Java	CodeBERT	0.10	0.46	0.19	-0.10	-0.47	-0.21	28.50	34.40	46.90

模型： 我们为 VARMISUSE任务使用两种模型：循环神经网络（RNN）[30] 和 Transformer [34]。第一种模型是一种简单的基于双向循环结构的架构，使用门控循环单元作为循环单元；Transformer 模型则是一种基于注意力的架构，通过全到全通信迭代地细化词元的表示。我们使用具有 512 个隐藏维度的 2 层循环神经网络，以及具有 8 个头和 512 个注意力维度的 6 层 Transformer。

D. 自然语言代码搜索（CODESEARCH）

任务： 在此任务中，给定一个自然语言查询，目标是从一组候选项中找出语义最相关的源代码。该任务被建模为一个二元分类问题：给定一对查询和代码，模型旨在判断代码在语义上是否与查询相关 [21]。该任务已在许多软件开发实践中得到积极研究和应用 [2, 35]。**数据：** 我们使用 Feng 等人[21] 基于原始 CodeSearchNet（CSN）数据集 [35],处理得到的数据集，其中每个样本都包含一个代码片段及与之配对的自然语言查询。该数据集包含数量相等的正样本和负样本。正样本是代码与查询相关的样本，标记为“1”。相反，负样本包含随机替换的不相关代码或查询，标记为“0”。我们选用 Java 语言数据（CSN-Java），其中包含 908K 多个训练样本、30K 个验证样本和 155 K 个测试样本。

模型： CodeBERT 是一种面向编程语言（PL）和自然语言（NL）的双模态预训练模型。它捕捉 NL 与 PL 之间的语义联系，并生成可广泛支持各种下游 NL-PL 任务的通用表示 [21]。它遵循 BERT [36] 和 RoBERTa [37], 的架构构建，而后者本身基于 Transformer [34], Transformer 被大多数大型预训练模型所采用。

IV. 结果与讨论

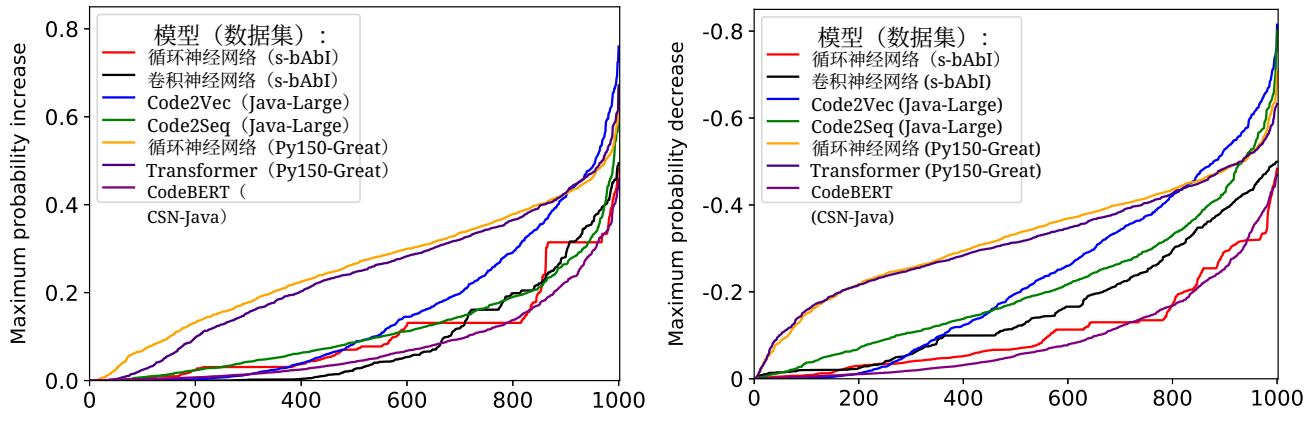
我们采用了文献中的四项任务的模型和数据集 [2, 12, 13] ，并从测试集中随机选取了 1，000 个样本输入程序进行分析。我们研究了对模型预测产生正面和负面影响的中间约简程序。通过发现并理解影响模型置信度的词元，我们希望获得有助于调试模型预测的洞见。在本节中，我们量化了干扰项对不同任务、模型和数据集中的模型置信度的影响。我们总结了干扰项引起的概率变化，并按类别突出显示排名靠前的干扰词元。这类分析是引导模型提高透明度的重要一步。

A. 研究问题*1*：干扰项的影响

为观察干扰项对任务和模型的影响，我们研究了产生显著概率变化（至少 10%）的样本比例。我们计算了这种变化的平均值，以及在约简步骤中两个方向（增加和减少）的最小值和最大值，所有结果汇总于表I。

从表I中的“样本（%）”可以看出，概率增加（PI）出现在最多 42% 的 s-bAbI 样本（由循环神经网络模型报告）、46% 的 Java-Large 样本（由 Code2Vec 模型报告）、85% 的 Py150-Great 样本（由循环神经网络模型报告）以及 28% 的 CSN-Java 样本（由 CodeBERT 模型报告）中。另一方面，概率下降（PD）出现在最多 54% 的 s-bAbI 样本（由卷积神经网络模型报告）、72% 的 Java-Large 样本（由 Code2Seq 模型报告）、95% 的 Py150-Great 样本（由 Transformer 模型报告）以及 34% 的 CSN-Java 样本（由 CodeBERT 模型报告）中。这些结果表明，**变量误用任务（Py150-Great 数据集上的循环神经网络和 Transformer 模型）在测试样本中表现出更多干扰项情况**，其比例分别比方法命名、漏洞检测和代码搜索任务高出近 20%、40% 和 50%。请注意，VARMISUSE任务中的最大可能缩减幅度有限，因为它需要保留所有变量出现位置，作为潜在的错误和修复目标 [12], ；因此，移除词元可能会导致超过 90% 的样本出现显著概率变化。

比较表I中的模型 PI 和 PD 值后，我们还观察到，与其他模型相比，Code2Vec 模型的“最大值”（PI=0.72，PD=-0.74）和“平均值”（PI=0.25，PD=- 0.27）概率变化值最高。其他模型的 PI 和 PD 值则更难以定论——例如，Code2Seq 模型在 Java-Large 样本上的 PI 平均值相对较低（=0.18），而循环神经网络模型在 s-bAbI 样本上的 PD 平均值相对较低（=-0.19）。这些发现可能表明，**平均而言，与其他模型相比，Code2Vec 模型更容易受到干扰项的影响**。



(a) 最大概率增幅 (MPI) (b) 最大概率降幅 (MPD)

图 2：样本的最大概率变化。

B. 研究问题*2*：概率变化的幅度

为了了解模型置信度显著变化时干扰项的影响程度，本节研究模型预测概率分数的最大增幅和最大降幅。图2展示了样本缩减过程中模型在任意步骤出现的最大概率变化。图2(a)突出显示最大概率增幅（MPI），图2(b)则显示最大概率降幅（MPD）。

根据图 2，对于 VARMISUSE 任务的大多数样本，RNN（Py150-Great）和 Transformer（Py150-Great）模型的 MPI 和 MPD 值高于其他任务模型报告的值。CODESEARCH 任务的 CodeBERT（CSN-Java）模型报告的 MPI 和 MPD 值恰好最低。对于 VULDETECTION 任务，RNN（s-bAbI）模型的 MPD 较低，而卷积神经网络（s-bAbI）模型的 MPI 较低。METHOD-NAMING 任务的模型（Code2Vec 和 Code2Seq）在 MPI 和 MPD 方面均处于中间水平。这些结果可能表明，在我们的实验中，**用于代码搜索任务的 CodeBERT 模型对单个词元的依赖程度相较于其他模型更高**。

C. RQ3：顶部干扰词元

接下来，我们研究约简步骤中从输入程序移除的干扰词元类别，尤其是在概率分数发生显著变化时。表 II 展示了从输入程序中移除、且对模型预测的概率分数至少造成 10% 变化的最常见词元。我们发现，综合考虑所有模型，在归一化 s-bAbI 数据集中共有 16 个不同的干扰词元。因此，我们仅列出不同任务和数据集中各模型共有的前 16 个干扰词元。

从表 II 中，我们可以观察到干扰词元属于以下类别：控制流（例如，“if”、“else”，

表 II： 模型预测概率分数变化至少 10% 的顶级干扰词元。

Task（数据集）	顶级干扰词元	
	类别	词元列表
漏洞检测（s-bAbI）	控制流	if、else、for、while、return
	数据类型	整数、字符
	数字	数字，0，1
	标识符	变量，随机
	运算符	=, <, ++
	其他符号	;
变量误用（Py150-Great）	控制流	if, for, 返回
	数据类型	None
	数字	0, 1
	标识符	断言相等
	运算符	., =, ==, :, 在, %
	其他符号	换行符 ，缩进 ，取消缩进
方法命名（Java-Large）	控制流	if、return
	数据类型	字符串
	数字	0, 1
	标识符	限定名称
	运算符	. 、 =、 ！ 、 :
	其他符号	; 、@、Override、super、new、this
代码搜索（CSN-Java）	控制流	if、return、try
	数据类型	字符串，整数
	修饰符	公有，终结，静态
	运算符	., =, <, >
	其他符号	;, @, Override, new

‘for’、‘while’、‘return’、‘try’）、数据类型（例如‘int’、‘char’、‘String’）、数字（例如‘0’、‘1’、‘numbers’）、运算符（例如‘.’、‘=’、‘<’、‘>’、‘==’、‘++’、‘%’）、语句结束语法（例如‘;’、‘NEWLINE’），以及其他标识符和关键字。有趣的是，我们可以看到，来自这些类别的词元是四个预测任务中最常见的干扰词元之一。因此，所有任务的模型似乎都受到了相似词元类别的影响（预测概率得分至少受到10%的影响）（请注意，没有数据类型

用于VARMISUSE任务，该任务处理Python代码）。这些结果可能表明，**来自各种类别的词元能够在模型置信度中发挥重要作用**。这一发现值得进一步分析相关程序，因为这些类别也可能构成实际负责预测的重要词元。我们期待在未来的工作中提供一项比较研究。

V. 相关工作

在计算机视觉、文本和自然语言处理领域，已有大量关于透明或可解释机器学习的研究，重点在于理解神经模型背后的推理过程 [38, 39]。可解释或透明机器学习对代码智能有诸多益处，包括使预测具有可解释性[10, 12], 、识别错误预测背后的推理过程 [11, 13],、理解关键特征 [9, 15], 、利用已学习的模型产生新的洞见 [4, 6], ，以及提升模型本身的质量 [16, 40, 41]。

A. 学习代码元素

模型往往会学习特定特征、简单捷径，甚至噪声，以达到目标性能。Compton等人[24] 表明，Code2Vec 的嵌入高度依赖变量名，并研究了混淆对改进 Code2Vec 嵌入的影响，以使其更好地保留代码语义。受词嵌入泛化能力研究的启发，Kang等人 [5] 评估了代码嵌入在各种软件工程任务中的泛化能力，并证明 Code2Vec 学到的嵌入并不总能泛化到其训练所用示例任务之外的其他任务。Rabin等人 [6, 23], 、Rabin和Alipour [42] 以及Yefet等人 [25]表明，代码模型通常缺乏鲁棒性或泛化能力，并容易受到对抗样本的影响。Suneja 等人 [13, 16] 通过检查原始代码中的漏洞在精简后的最小代码片段中是否消失，揭示了模型对错误信号的依赖。Rabin等人 [12, 15] 证明，模型通常只利用少数简单的句法捷径来进行预测。Allamanis [3] 发现，代码重复等因素很容易误导深度学习器，这种重复会虚假地提高其性能。Rabin等人 [4] 随后表明，模型凭借过大的参数容量能够拟合含噪训练数据，从而导致泛化性能下降。由于模型往往会学习噪声或无关特征来获得较高的预测性能，因此，若不了解哪些输入特征会影响模型的预测，就会损害正确分类的可信度。在漏洞检测或自动化缺陷修复等关键应用中，这种不透明性会造成更严重的问题。因此，我们尝试突出显示干扰词元，以提升透明度。

B. 提取关键特征

已有多种研究致力于为源代码模型寻找相关输入特征。Allamanis等人 [7]使用程序中的一组特征，并表明，提取能够捕获全局上下文的相关特征，对于学习有效的代码上下文至关重要。Rabin等人 [8]尝试通过人工检查某个标签下的一些输入程序，来寻找该标签的关键输入特征。Bui等人 [9]尝试通过扰动程序中的语句，并结合相应的注意力和置信度分数，识别相关代码元素。Cito等人 [11] 通过对代码进行最小幅度的修改，寻找模型改变预测结果时现实且合理的反事实解释。Wang等人 [10] 提出了一种变异—约简方法，在考虑程序有效性的前提下，寻找代码摘要模型中的关键特征。Rabin等人 [12, 15] 和 Suneja等人 [13, 16] 使用输入程序约简技术，寻找能够保留模型预测结果的最小输入，进而找出与预测相关的程序关键词元。作者通过从输入程序中删除与预测无关的部分，旨在更好地理解模型推理中的关键特征。我们以差分调试算法为基础，研究从输入程序中删除词元后的影响。

VI. 结论

本文介绍了代码智能模型中干扰特征的概念。我们提出了一种基于差分调试的技术，用于研究影响模型预测置信度的特征。我们在多个代码任务、模型和数据集上评估了该方法。我们观察到，Code2Vec 模型更容易受到干扰项的影响，而 CodeBERT 模型对单个词元的依赖相对更高。未来，我们计划针对不同训练和测试方面，详细研究各种大型生成模型中干扰项的影响及其范围。

参考文献

[1] M. Allamanis、E. T. Barr、P. Devanbu 和 C. Sutton，《面向大规模代码和自然性的机器学习综述》，发表于 *ACM Computing Surveys*，2018 年。[2] S. Lu、D. Guo、S. Ren 等，《CodeXGLUE：用于代码理解和生成的机器学习基准数据集》，2021 年。[3] M. Allamanis，《代码重复对代码机器学习模型的不利影响》，发表于*Onward!*，2019 年。[4] M. R. I. Rabin、A. Hussain、M. A. Alipour 和 V. J. Hellendoorn，《神经代码智能模型中的记忆与泛化》，*IST*，2023 年。[5] H. J. Kang、T. F. Bissyand´e 和 D. Lo，《评估 code2vec 令牌嵌入的泛化能力》，*ASE'19*。[6] M. R. I. Rabin、N. D. Bui、K. Wang、Y. Yu、L. Jiang 和 M. A. Alipour，《关于神经程序模型针对保持语义的程序变换的泛化能力》，发表于*IST*，2021 年。[7] M. Allamanis、E. Barr、C. Bird 和 C. Sutton，《提出准确的方法名和类名》，*ESEC/FSE'15*。[8] M. R. I. Rabin、A. Mukherjee、O. Gnawali 和 M. A. Alipour，《迈向揭示源代码嵌入的维度》，发表于*RL+SE&PL*，2020 年。

[9] N. D. Q. Bui、Y. Yu 和 L. Jiang，《Autofocus：通过代码扰动解释基于注意力的神经网络》，发表于 *ASE*，2019 年。[10] Y. Wang、K. Wang 和 L. Wang，《WheaCha：一种解释代码模型预测的方法》，2022 年。[11] J. Cito、I. Dillig、V. Murali 和 S. Chandra，《代码模型的反事实解释》，*ICSE-SEIP*，2022 年。[12] M. R. I. Rabin、V. J. Hellendoorn 和 M. A. Alipour，《通过程序简化理解神经代码智能》，发表于*ESEC/FSE*，2021 年。[13] S. Suneja、Y. Zheng、Y. Zhuang等，《通过保持预测的输入最小化探测模型的信号感知能力》，发表于 *ESEC/FSE*，2021 年。[14] U. Alon、O. Levy 和 E. Yahav，《code2seq：从代码的结构化表示生成序列》，发表于 *ICLR*，2019 年。[15] M. R. I. Rabin、A. Hussain 和 M. A. Alipour，《用于理解神经代码智能模型的语法引导程序约简》，发表于*MAPS*，2022 年。[16]S. Suneja、Y. Zhuang、Y. Zheng、J. Laredo 和 A. Morari，《数据驱动且由软件工程辅助的人工智能模型信号感知能力增强与自省》，2021 年。[17] A. Zeller 和 R. Hildebrandt，《简化并隔离故障诱发输入》，发表于*TSE*，2002 年。[18] M. Allamanis、H. Peng 和 C. A. Sutton，《用于源代码极限摘要的卷积注意力网络》，发表于*ICML*，2016 年。[19]M. Allamanis、M. Brockschmidt 和 M. Khademi，《学习用图表示程序》，*ICLR'18*。[20]M. Vasic、A. Kanade、P. Maniatis、D. Bieber 和 R. singh，《通过联合学习定位和修复实现神经程序修复》，发表于*ICLR*，2019 年。[21] Z. Feng、D. Guo 等，《CodeBERT：面向编程语言和自然语言的预训练模型》，2020 年。[22] K. Wang 和 M. Christodorescu，《COSET：用于评估神经程序嵌入的基准》，2019 年。[23] M. R. I. Rabin、K. Wang 和 M. A. Alipour，《测试神经程序分析器》，*ASE-LBR*，2019 年。[24] R. Compton、E. Frank、P. Patros 和 A. Koay，《使用 code2vec 嵌入 Java 类：变量混淆带来的改进》，发表于 *MSR*，2020 年。[25] N. Yefet、U. Alon 和 E. Yahav，《代码模型的对抗样本》，发表于 *PACMPL-OOPSLA*，2020 年。[26]M. R. I. Rabin 和 M. A. Alipour，《Code2Snapshot：“使用代码快照学习源代码表示”，*ICMLA*，2022年。[27] V. J. Hellendoorn、C. Sutton、R. Singh 等，“源代码的全局关系模型”，发表于*ICLR*，2020年。[28] S. Suneja、Y. Zheng、Y. Zhuang 等，“迈向可靠的源代码理解人工智能”，*SoCC*，2021年。[29] C. D. Sestili、W. S. Snavely 和 N. M. VanHoudnos，“借助人工智能进行安全缺陷预测”，2018年。[30] K. Cho、B. van Merrienboer、C. Gulcehre 等，“使用 RNN 编码器-解码器学习短语表示，用于统计机器翻译”，发表于*EMNLP*，2014年。[31]Y. Kim，“用于句子分类的卷积神经网络”，发表于*EMNLP*，2014年。[32]U. Alon、M. Zilberstein、O. Levy 和 E. Ya- hav，“Code2vec：学习代码的分布式表示”，发表于 *PACMPL-POPL*，2019年。[33]V. Raychev、P. Bielik 和 M. Vechev，“基于决策树的代码概率模型”，发表于 *OOPSLA*，2016年。[34]A. Vaswani、N. Shazeer、N. Parmar、J. Uszkoreit、L. Jones、A. N. Gomez、Ł. Kaiser 和 I. Polosukhin，“注意力就是一切”，发表于 *NeurIPS*，2017年。[35]H. Husain、H.-H. Wu、T. Gazit、M. Allamanis 和 M. Brockschmidt，“CodeSearchNet 挑战赛：评估语义代码搜索的现状”，2019年。[36]J. Devlin、M.-W. Chang、K. Lee 和 K. Toutanova，“BERT：用于语言理解的深度双向 Transformer 预训练”，发表于 *NAACL-HLT*，2019年。[37] Y. Liu、M. Ott、N. Goyal、J. Du、M. Joshi、D. Chen、O. Levy 等，“RoBERTa：一种经过稳健优化的 BERT 预训练方法”，*ICLR*，2020年。[38] W. Samek、G. Montavon、S. Lapuschkin 等，“深度神经网络及其他模型的解释：方法与应用综述”，*IEEE*，2020年。[39] T. R¨auker、A. Ho、S. Casper 和 D. Hadfield-Menell，“迈向透明人工智能：解读深度神经网络内部结构的调查”，2022年。[40] Z. Zhang、H. Zhang、B. Shen 和 X. Gu，“精简代码有益健康：简化代码预训练模型的程序”，*ESEC/FSE*，2022年。[41] D. Wang、Z. Jia、S. Li、Y. Yu、Y. Xiong、W. Dong 和 X. Liao，“连接代码预训练模型与源代码理解下游任务”，*ICSE*，2022年。[42] M.R. I.Rabin 和 M. A.Alipour，“语义保持变换下神经程序分析器泛化能力的评估”，2020年。