# 基于代码仓库的合并冲突

赵涵[1]

[1](南京邮电大学 计算机学院、软件学院、网络空间安全学院,江苏 南京 210023)
通讯作者: 赵涵, E-mail: 1024041010@njupt.edu.cn

**摘 要**: 在现代协同开发中，合并冲突已成为影响软件开发效率与质量的重要因素。为了降低手动解决冲突的成本，研究者提出了多种自动化解决技术，其中基于深度学习的方法得到广泛关注。本文介绍了三种典型的基于差分测试的合并冲突解决方法:《DeepMerge: Learning to Merge Programs》,《MergeBERT: Program Merge Conflict Resolution via Neural Transformers》和《Merge Conflict Resolution: Classification or Generation?》。DeepMerge 通过指针网络，实现基于行级的代码重排；MergeBERT 使用 BERT 系列编码器，将冲突区域分类；MergeGen 则将合并解决规定为字符串生成问题，引入 CodeT5 生成模型，支持新内容生成。通过对该三种方法的结构设计，处理能力，分析性能进行系统总结，本文提炼了合并解决技术现有的优势与异同，并指出这三篇论文为促进开发人员协同工作提供了新思路和工具。

**关键词**: 合并冲突；自动化合并；深度学习
**中图法分类号**: TP311

## Papers Reading Report

Zhao han[1]

[1](School of Computer Science, Nanjing University of Posts and Telecommunications, Nanjing 210023, China)

**Abstract**: In modern collaborative software development, merge conflicts have become a significant factor affecting productivity and software quality. To reduce the cost of manual conflict resolution, researchers have proposed various automated approaches, particularly those based on deep learning. This paper introduces three representative differential testing-based merge conflict resolution methods: "DeepMerge: Learning to Merge Programs", "MergeBERT: Program Merge Conflict Resolution via Neural Transformers", and "Merge Conflict Resolution: Classification or Generation?". DeepMerge utilizes pointer networks to reorder code segments at the line level; MergeBERT employs a BERT-based encoder to classify conflict regions; and MergeGen formulates merge resolution as a sequence generation problem, leveraging the CodeT5 model to support flexible new content generation. Through a systematic analysis of their structural designs, processing capabilities, and performance characteristics, this paper summarizes the strengths and differences of current merge resolution techniques and points out that these three studies offer new ideas and tools to facilitate collaborative work among developers.

**Key words**: Merge conflict; Automated merging; Deep learning

在以 Git 为代表的版本控制系统支持下，现代软件开发日益依赖于代码仓库中分支的协作与合并操作。多个开发者在不同分支上并行开发功能，最后将其合入主干，这一机制显著提高了协同开发效率。然而，这种并行开发也引发了一个长期存在的问题——合并冲突。冲突通常发生在两个分支对同一文件的同一位置进行了不同修改，而版本控制系统无法自动决定保留哪一方的修改。

代码仓库中的合并冲突不仅延迟了开发进度，还可能造成构建失败，严重影响持续集成（CI）流程的顺利进行。据相关研究统计，大型软件项目中约 10% 到 20% 的合并操作会产生冲突，其中高达一半的合并冲突需要开发者手动干预，极大地影响了开发效率和软件质量。随着深度学习技术的进步，合并冲突解决方式从经典的结构化/半结构化算法，进化到基于分类和生成的深度学习模型。

本文第 1 节介绍了一种基于深度学习的合并方法，利用指针网络和编辑感知嵌入技术进行代码合并。第 2 节介绍了采用 BERT 系列编码器将合并冲突问题转化为分类任务，通过对不同版本的代码进行编码和分类，实现自动化冲突解决。第 3 节介绍了一种基于生成模型的方法，利用生成模型（如 CodeT5）自动生成合并结果，支持新内容的生成与更灵活的合并方式。最后，第 4 节对这三篇论文进行总结与对比，分析其优缺点，

并探讨未来研究的方向。

## 1 首次结合机器学习的冲突合并

论文《DeepMerge: Learning to Merge Programs》[1]开创性地将合并冲突处理建模为编辑感知的序列到序列学习任务，并通过指针网络成功捕捉了开发者的手动合并模式，极大拓展了数据驱动程序合并的研究空间，为后续生成式合并方法奠定了基础。

### 1.1 研究问题

传统的三向文本合并方法（如 diff3）在实际应用中存在较大局限性。由于缺乏对程序语义的理解，这类方法在面对复杂编辑冲突时，常常表现出较差的自动合并能力[4]，导致开发者需要频繁手动干预。即便是后续发展起来的结构化或半结构化合并技术，在处理动态类型语言（如 JavaScript）时，由于类型信息缺失，效果依然受限基于此背景[5]，DeepMerge 提出了一项新的研究问题：能否通过深度学习的方法，基于历史合并样本，自动学习开发者的冲突处理模式，从而高效、智能地解决合并冲突？具体而言，DeepMerge 关注的是，在不引入新代码的前提下，仅通过智能地选择和重排已有文本内容，完成高质量的冲突解决[6]。

为此，DeepMerge 采用了编辑感知编码方式，将冲突区域内的各版本变更显式建模，并结合指针网络捕捉潜在的重排模式。这种设计使得模型能够在理解冲突细节的基础上，合理决策合并输出，从而有效减少需要人工处理的冲突比例。

### 1.2 相关工作

在 DeepMerge 之前，程序合并领域已有若干重要技术路线。最早期的是完全基于文本的合并方法，如 diff3，它们简单高效，但无法理解代码的结构或语义，容易在存在复杂编辑的情况下引发冲突。随后出现了结构化合并（如 JDime），这类方法利用程序的抽象语法树（AST）进行对齐和合并，在一定程度上提升了准确性[7]。然而，在动态语言领域，由于缺乏静态类型信息，结构化合并技术往往难以有效应用。为了解决这一问题，研究者提出了半结构化合并（如 FSTMerge），结合了文本和结构信息，但其泛化性和扩展性仍受限制[8]。

此外，早期也有少量基于机器学习的工作，如使用浅层分类器或启发式规则对简单冲突模式进行预测。但这些方法通常针对特定场景设计，缺乏大规模验证和深层次建模能力[9]。

DeepMerge 的创新之处在于，它首次将程序合并冲突处理系统性地建模为序列到序列（Seq2Seq）学习问题，通过深度神经网络直接从数据中学习冲突解决的潜在模式，标志着程序合并技术由规则驱动向数据驱动的转变。

### 1.3 研究方法

DeepMerge 的研究方法包含了多个创新设计，包括编辑感知嵌入技术，指针网络变体解码器和数据集的构建，这些模块的巧妙设计有效提高了合并冲突框架的有效性。

在输入表示方面，论文提出了编辑感知嵌入技术。为了寻找最优的输入建模方式，作者对比了多种设计，包括直接拼接原始代码、简单线性化代码序列、结合原始代码和编辑序列（LTRE）、以及仅基于对齐后的编辑序列[10]。这些方法要么信息冗余、难以学习，要么缺乏结构化和线性化的编辑表达。综合分析后，论文最终采用了对齐线性化方法，通过对齐后的编辑序列，并结合线性化处理，既保留了充分的编辑语义，又有效简化了输入结构，提升了模型对冲突变化的感知与理解能力[11]。

在模型架构上，DeepMerge 引入了变种的指针网络作为解码器。不同于传统的序列生成，指针网络不直接生成新内容，而是从输入序列中选择已有行，按序排列，构造最终的合并输出。这种设计充分利用了合并冲突中“重排而非生成”的特点，避免了生成式模型在代码场景中容易出现的语义错误。

图 1 是 DeepMerge 的框架示意图，可以清晰地看出如何运用编辑感知对齐和指针网络的技术处理合并冲突。

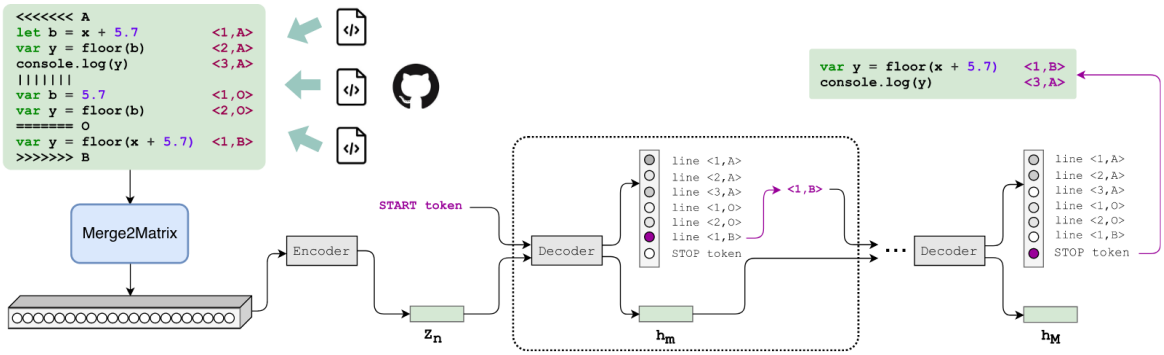


图 1　DeepMerge 的框架示意图

为了训练模型，作者从 GitHub 收集了大规模真实项目中的合并冲突案例，通过自动化算法筛选和提取高质量样本，最终构建了包含 8,719 个复杂冲突实例的数据集。每个样本包含三个输入文件（A、B、O）和一个手动合并后的正确输出（M），用于监督学习。

### 1.4 评估

DeepMerge 主要通过以下三个研究问题（RQ）进行评估：

RQ1: DeepMerge 在复杂合并冲突中的表现。为了评估 DeepMerge 在解决复杂合并冲突时的性能，实验重点考察了在没有人工干预的情况下，DeepMerge 能够自动解决多少非平凡的合并冲突。非平凡冲突指的是那些具有高度复杂性，无法通过简单的“选取”方式解决的冲突（例如：需要重排代码块或同时融合多个修改）。实验结果表明，DeepMerge 在这些复杂冲突的解决中，能够准确生成符合人工解决方案的合并结果。在 37% 的情况下，DeepMerge 的解决方案与人工合并的结果一致，表现明显优于现有的半结构化合并工具。

RQ2: DeepMerge 的编辑感知嵌入方式对合并效果的影响。为了验证编辑感知嵌入的有效性，实验通过对比不同的输入表示方式来评估该方法的贡献。具体来说，DeepMerge 提出了几种不同的编码方式，如 直接拼接、简单线性化、LTRE 等。最终选择的编辑感知嵌入方式表现出了最佳的效果。与其他方法相比，这种方法能够更有效地捕捉代码冲突区域的变化模式，尤其是在复杂冲突和不规则编辑中，展现出了更强的适应性和准确性。

RQ3: DeepMerge 与传统方法相比在效率和准确性上的差异。实验还比较了 DeepMerge 与传统的文本合并方法（如 diff3）以及一些半结构化合并工具（如 FSTMerge）的效率和准确性。结果表明，DeepMerge 在解决复杂冲突时，不仅能够提高合并的准确性，还能在处理时间上取得显著的提升。通过对比，DeepMerge 的方法在处理小规模冲突（3 行以内）的准确率达到 78%，远超传统方法的效果。

### 1.5 小结

DeepMerge 提出了一个基于深度学习的新型程序合并方法，通过结合指针网络和编辑感知嵌入技术，显著提升了合并冲突解决的效果。实验结果表明，DeepMerge 在解决复杂合并冲突方面表现出色，且其编辑感知嵌入方式为模型的成功提供了强有力的支持。

未来的研究方向包括将 DeepMerge 扩展到多语言支持，进一步增强模型的泛化能力，同时在语义层面进行更深层次的冲突分析。

## 2 基于分类的 Token 级冲突合并

论文《MergeBERT: Program Merge Conflict Resolution via Neural Transformer》[2]将合并冲突问题建模为一个分类任务，探索是否能通过预定义的九种基本合并模式提高模型的泛化能力和处理效率。具体而言，

MergeBERT 主要利用 BERT 这样的预训练模型，在不同的编程语言中处理并分类合并冲突，并且通过训练一个多语言合并分类器，在保证准确性的同时提高解决冲突的效率[12]。

### 2.1 研究问题

合并冲突的自动化解决一直面临几个主要问题。首先，许多传统的合并方法仅依赖于行级差异，无法捕捉到代码中的复杂语义关系，导致在复杂的合并场景下常常无法得到准确的合并结果。其次，现有的深度学习方法，如 DeepMerge，主要集中于基于行的合并处理，这种方法在面对需要细粒度理解的冲突时，往往表现得力不从心。此外，现有的合并方法大多数是为特定编程语言设计的，缺乏跨语言的泛化能力[13]。

针对这些问题，本文提出了 MergeBERT，旨在解决以下改进问题：如何通过 token 级别的差异表示，利用深度预训练模型（如 BERT）提高合并冲突解决的准确性与灵活性？以及如何设计一个多语言兼容的合并模型，解决跨语言冲突的合并问题？

### 2.2 相关工作

近年来，深度学习技术在软件工程领域得到了广泛应用，包括合并冲突。如 DeepMerge 主要集中在基于行级合并或指针网络的解决方案上[14]。该方法通过引入指针网络（Pointer Network），将冲突区域的代码重排问题转化为一个序列生成任务，重点关注行级的合并处理[15]。在这种方法中，每一行代码都被视为一个独立的单元，模型通过学习合并的历史模式来预测最佳的代码行重排顺序[16]。然而，尽管 DeepMerge 在一些简单冲突场景中表现良好，但它仅依赖于冲突区域的代码行重排，忽略了代码中更加细粒度的语法和语义信息。

MergeBERT 的创新之处在于，它通过将合并任务转化为一个标准的分类问题，利用 BERT 系列编码器对代码块进行 token 级别表示，并通过分类模型识别冲突解决方案。与 DeepMerge 的指针网络不同，MergeBERT 基于多分类任务，解决了行级合并仅适用于简单冲突的局限性。

### 2.3 研究方法

MergeBERT 的研究方法基于深度学习中的预训练模型 BERT，将合并冲突问题转化为一个分类任务，并利用 Token 级别的差异表示来捕捉冲突的细粒度信息。

论文改进了以往的输入表示形式，将基础版本（O）与两个修改分支（A 和 B）进行比较，计算每个 token 级别的差异。与传统的行级合并方法不同，它在处理合并冲突时关注代码中的每一个细小变化，从而能够捕捉到更丰富的上下文信息[17]。

对于每个冲突的区域，MergeBERT 预测一个合适的合并策略，论文设计了九种基本合并模式，能够覆盖绝大多数冲突合并的情况。为了训练这个模型，论文使用了来自不同编程语言的大量合并冲突样本，尤其是 Java、JavaScript 和 C# 等语言，通过多类分类器来训练模型。该分类器的输出是合并策略的类别标签。与传统的基于行级合并的模型相比，MergeBERT 的方法能够处理更为复杂的合并模式，并且能够跨不同的编程语言进行泛化。

此外，MergeBERT 的方法还依赖于预训练的 BERT 模型，它通过无监督的预训练阶段先学习大规模代码库中的通用语言知识，然后通过监督学习阶段，进一步针对合并冲突任务进行微调。这种两阶段训练流程不仅提升了模型的准确性，还增强了模型的适应性和泛化能力。

图 2 是 MergeBERT 的方法概览图，首先将给定冲突的程序执行标记级差分并对齐相应的序列。再提取对应编辑序列作为类型嵌入使用。最后，将编码的标记序列汇总为隐藏状态，作为分类层的输入得出目标解决方案并解码。

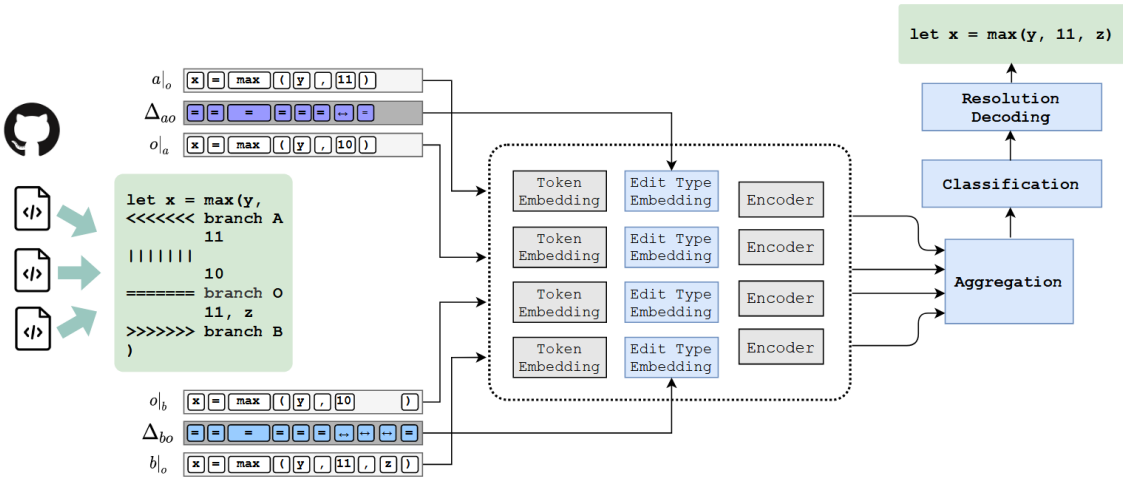


图 2　MergeBERT 的方法概览图

### 2.4 评估

MergeBERT 的评估主要基于以下几个研究问题（RQ）：

RQ1: MergeBERT 能在不同编程语言中解决合并冲突吗？实验结果表明，MergeBERT 在多种编程语言上表现出色。在这些语言上，MergeBERT 能通过 token 级别差异表示有效地进行合并分类，且准确率优于传统基于行的合并方法。

RQ2: MergeBERT 的多类分类方法能否有效提高合并冲突解决的精度？评估结果表明，MergeBERT 在九类合并策略的分类任务中，能够有效地处理不同类型的冲突，并且能够提供更高效的合并方案。与传统的深度学习模型（如 DeepMerge）相比，MergeBERT 在处理复杂冲突时，能够更灵活地应用预定义的合并策略，从而提高了整体合并精度。

RQ3: MergeBERT 与现有技术相比具有哪些优势？MergeBERT 相比于其他传统方法（如 FSTMerge）以及 DeepMerge 的指针网络方法，展现出了更高的准确率与更强的语言泛化能力。通过多语言预训练，MergeBERT 能够跨语言进行冲突解决，而不需要为每种编程语言设计特定模型。

### 2.5 小结

MergeBERT 提出了基于 BERT 编码器的程序合并方法，通过 token 级别分类模型解决了复杂的合并冲突。相比传统的行级重排方法，MergeBERT 能够通过细粒度的冲突表示，更加精准地处理多种编程语言中的合并问题。其多类分类策略显著提高了模型对复杂冲突的处理能力，也使得它能够更好地处理跨语言合并任务。

在未来的研究中，MergeBERT 可以进一步结合其他模型，提升合并灵活性与智能性，特别是在处理极为复杂和动态的合并冲突时。此外，如何进一步优化模型的计算效率和提升其在大规模项目中的应用性能，也是值得进一步研究的方向。

## 3　结合生成模型的冲突合并

论文《Merge Conflict Resolution: Classification or Generation?》[3]提出了一个基于生成模型的新型程序合并方法，通过将合并冲突视为一个字符串生成任务，能够自动生成新的合并代码。与传统的分类模型不同，MergeGen 通过生成模型能够处理更复杂的冲突，特别是那些需要创造新代码的情况。

### 3.1 研究问题

在合并冲突的自动化解决中，现有的方法大多集中在分类任务和基于现有文本的重排上。然而，传统的

分类方法（如 MergeBERT）和基于行级重排的方法（如 DeepMerge）存在一定的局限性，主要体现在它们只能基于已有的代码行进行合并，无法生成新的代码片段或者处理更复杂的代码变更。特别是在面对需要生成新内容的复杂冲突时，这些方法可能无法有效解决问题[18]。

因此，本文提出了 MergeGen，其核心问题是：能否通过生成模型（如 CodeT5），将合并冲突视为一个字符串生成问题，自动生成新的合并代码，而不是单纯地选择或重排已有的代码行？

### 3.2 相关工作

合并冲突的历史研究主要集中于分类方法和行级处理方法[19]。这些方法通常将合并任务视为一个分类问题或基于代码行的差异处理问题，旨在自动选择和重排不同版本中的代码行[20]。例如，DeepMerge 通过指针网络将合并冲突处理视为一个基于行级的重排问题，通过学习冲突区域的模式，重排已有代码行来生成最终的合并结果。同样，MergeBERT 通过使用预训练的 BERT 编码器，将合并冲突建模为分类任务，并将冲突区域的代码分类到九个基本的合并模式中。这些方法都取得了一定的效果，尤其在处理简单的、行级的冲突时，能够较好地自动化解决[21]。

然而，这些基于行级或分类的合并方法存在着显著的局限性[22]。首先，它们依赖于文本行的重排，无法深入理解代码的语法和语义结构。当冲突不仅仅是简单的代码行顺序问题时，行级方法往往无法提供有效的合并解决方案[23]。此外，现有的分类方法通常只能在给定的合并模式中选择，缺乏处理新内容生成和更加复杂合并场景的能力。

近年来，像 CodeT5 这样的预训练生成模型已被成功应用于代码生成、代码补全等任务，证明了其在程序生成方面的强大能力[24]。MergeGen 的创新之处在于将生成模型引入到程序合并领域，将合并冲突视为一个生成问题，直接通过模型生成合并后的代码，而不是简单地从现有代码片段中选择合适的部分[25]。有效解决了以往工作中无法生成新 Token 和组合模式受限的问题。

### 3.3 研究方法

MergeGen 的核心思想是将合并冲突任务转化为字符串生成问题，而不是传统的分类任务。具体来说，论文通过两个核心组件来解决合并冲突问题：结构化和细粒度冲突感知表示和编码器-解码器生成模型[26]。

MergeGen 将合并冲突的输入表示为更为细粒度的 token 级别表示，而不是简单的行级合并。同时与 MergeBert 不同的是，它还通过对冲突区域进行结构化的编码，使用特殊的 token（如<boc>、<soc>）将不同的冲突表示直接输入给模型，避免模型读取方面的难度和造成的误解[27]。

生成任务通过使用预训练的 CodeT5 生成模型作为架构的核心。编码器部分负责对输入的代码进行处理，通过自注意力机制，捕捉输入代码中的深层语法和语义信息，从而为后续生成提供充分的上下文支持。解码器部分则根据编码器提供的上下文信息，生成合并后的代码片段[28]。与传统的基于选择的合并方法不同，MergeGen 的解码器生成新的代码内容，而不仅仅是重排已有的行。这使得 MergeGen 能够处理那些包含新内容或复杂修改的合并冲突，提供更加灵活和准确的合并方案[29]。

具体的框架架构如图 3 所示。

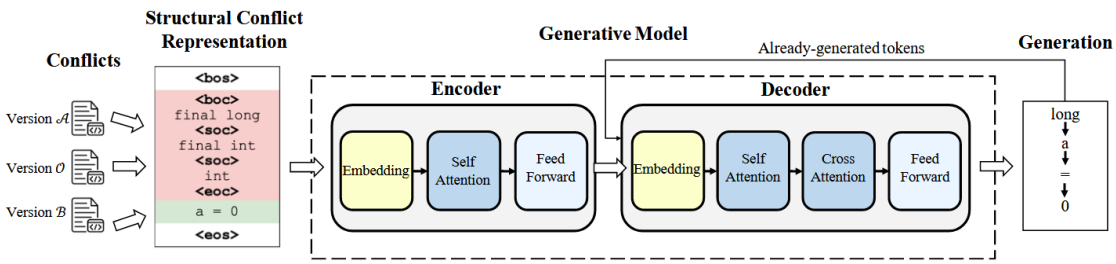


图 3　MergeGen 的框架架构图

通过这两个核心组件的结合，MergeGen 实现了对复杂合并冲突的智能解决，能够生成符合语法和语义的代码，而不仅仅是简单地选择或重排现有的代码行。结构化的细粒度冲突感知表示使得模型能够理解复杂的代码变更，而编码器-解码器生成模型则提供了生成新合并代码的能力。

### 3.4 评估

为了全面验证 MergeGen 在合并冲突自动解决中的有效性，研究从多个维度进行了系统性评估。首先，实验重点考察了 MergeGen 是否能够在面对复杂冲突时成功生成新的合并代码。评估结果表明，MergeGen 能够有效地理解冲突区域的上下文信息，并生成符合语法和语义的新代码片段。特别是在涉及变量声明变化、逻辑表达式重构、函数调用调整等需要创造性修改的冲突中，MergeGen 展现出了极高的准确率，与人工合并结果高度一致。

进一步的实验对比了 MergeGen 与传统分类模型（如 MergeBERT）在处理复杂冲突时的表现差异。结果显示，当冲突仅通过简单选择分支代码即可解决时，两者表现相近；但当冲突需要综合多个来源并引入新的逻辑时，MergeGen 明显优于 MergeBERT。MergeGen 通过生成新的合并代码，避免了分类模型在面对超出预设合并模式之外冲突时出现的僵化局限，极大地提升了解决复杂冲突的能力。

此外，为了验证 MergeGen 的跨语言适应性，研究在不同编程语言（如 Java、Python 和 C++）的合并冲突数据集上进行了测试。实验结果表明，MergeGen 不仅在单一语言中表现优异，在跨语言任务中也能保持稳定而优秀的生成效果。这表明 MergeGen 所采用的 CodeT5 生成模型能够较好地捕捉不同编程语言中的通用代码模式，具有良好的泛化能力。

整体来看，无论是在生成新内容的能力、在复杂冲突中的准确性，还是在不同编程语言环境下的一致性表现上，MergeGen 都明显优于传统的基于选择和分类的合并方法，为自动化合并冲突提供了一种更加灵活而强大的解决方案。

### 3.5 小结

MergeGen 提出了一种全新的合并冲突解决方法，将程序合并建模为一个端到端的生成任务。通过引入结构化和细粒度的冲突感知表示，MergeGen 能够全面理解冲突区域的复杂变化细节；同时，依托强大的编码器-解码器生成模型（CodeT5），它不仅能够重用已有代码，还能在需要时生成符合逻辑的新代码片段。与传统的基于行级重排或分类选择的方法不同，MergeGen 在面对复杂、变化多样的冲突场景时，展现出了卓越的灵活性和生成能力。

实验结果充分验证了 MergeGen 在多种环境下的有效性，尤其在处理需要引入新逻辑的冲突和跨编程语言合并任务中表现出色。相比传统方法，MergeGen 不再受限于预定义的合并策略或简单的文本选择，而是真正具备了“理解”代码冲突并创造性解决的能力。

未来的研究可以在多个方向进一步拓展。首先，可以探索更大规模、更丰富冲突类型的数据集，进一步提升模型在实际复杂项目中的应用效果。其次，将生成模型与分类模型结合，通过混合决策机制，结合选择与生成的优点，可能会带来更高效、更精确的合并解决方案。此外，引入代码执行验证机制，如生成后静态分析或单元测试验证，也可能进一步提升 MergeGen 生成合并结果的可靠性和安全性。这些方向都为程序合并领域的持续发展提供了广阔的空间和机遇。

## 4 总 结

随着协同开发规模的不断扩大，合并冲突已成为制约软件开发效率与质量的重要瓶颈。针对这一问题，近年来研究者们探索了多种基于深度学习的自动化解决方法。本文系统回顾了三种代表性技术路线，即 DeepMerge、MergeBERT 和 MergeGen，总结了它们在程序合并冲突解决中的研究思路、技术创新与实际效果。

DeepMerge 作为较早提出的基于深度学习的程序合并方法，首次将合并冲突建模为序列到序列问题，并

引入了编辑感知嵌入与指针网络，通过对冲突区域行级重排来实现自动化合并。它有效地提升了对非平凡冲突的处理能力，验证了深度学习方法在程序合并领域的应用潜力。但由于仅关注行级重排，DeepMerge 在处理涉及新内容生成或更复杂语义变化的冲突时仍存在一定局限。

在此基础上，MergeBERT 进一步深化了合并冲突的建模方法。通过引入 BERT 编码器并将合并问题形式化为多类分类任务，MergeBERT 将冲突区域细粒度建模到 token-level，捕捉了更丰富的语法和语义特征。其多语言支持和高效分类能力使得 MergeBERT 在复杂冲突的自动分类和合并上取得了较好效果。然而，作为分类模型，MergeBERT 仍然受限于预定义的合并策略，缺乏生成全新内容的能力，难以应对更复杂的合并场景。

图 4 是 DeepMerge 和 MergeBERT 的对比实验数据，通过这些指标我们可以更好地了解 MergeBERT 在包括有效性，召回率等多种指标上的结果。

| Approach | Precision | Recall | F-score | BLEU-4 |
|---|---|---|---|---|
| LM (line) | 3.6 | 3.1 | 3.3 | 5.2 |
| LM (token) | 49.7 | 48.1 | 48.9 | 55.3 |
| DeepMerge | 34.9 | 22.2 | 27.1 | 51.2 |
| MergeBERT | **69.1** | **68.2** | **68.7** | **78.6** |

图 4　DeepMerge 与 MergeBERT 的对比试验

针对现有方法的局限，MergeGen 提出了全新的生成式合并策略。通过引入结构化、细粒度的冲突感知表示，结合 CodeT5 预训练生成模型，MergeGen 将程序合并冲突转化为端到端的生成任务。它不仅能够在已有代码基础上重排内容，还可以根据冲突上下文自动生成新的、符合语法与语义要求的合并代码。实验证明，MergeGen 在复杂冲突处理、跨语言泛化能力及新内容生成方面均优于传统分类与重排方法，展现了极大的应用潜力。

图 5 是 MergeGen 的实验结果，可以帮助理解生成式模型在处理冲突合并时候的突出功能。

| Model | Precision | Accuracy |
|---|---|---|
| diff3 | **85.5** | 2.2 |
| JDIME [19] | 26.3 | 21.6 |
| MergeBERT [24] | 63.9 | 63.2 |
| MergeGen (our tool) | 69.2 | **67.7** |

图 5　MergeGen 的实验结果

综观三种方法的发展脉络，可以清晰看到程序合并冲突解决技术从行级重排到 token-level 细粒度建模，再到生成式合并的演进过程。每一次技术革新都推动了自动合并能力的边界，从简单冲突到复杂语义变化，从选择已有内容到生成新代码，深度学习在程序合并领域展现出越来越强大的应用潜力。

尽管如此，目前的研究仍面临诸多挑战。如何进一步提升生成模型在超大规模项目中的稳定性与效率，如何引入合并结果的静态分析或测试验证机制，如何在人机协同的框架下优化自动合并与开发者决策的结合，

都是未来值得深入探索的问题。同时，将分类与生成模型有效融合，动态选择合适策略，或许能够带来更加智能、灵活且安全的程序合并系统。

合并冲突自动化解决作为软件工程与人工智能交叉的重要方向，正在快速发展，未来也必将在更广泛的实际工程实践中发挥重要作用。

**References**:


[1] Elizabeth Dinella, Todd Mytkowicz, Alexey Svyatkovskiy, Christian Bird, Mayur Naik, and Shuvendu Lahiri. 2023. DeepMerge: Learning to Merge Programs. IEEE Transactions on Software Engineering, 49(4), 1599–1617.

[2] Alexey Svyatkovskiy, Todd Mytkowicz, Negar Ghorbani, Sarah Fakhoury, Elizabeth Dinella, Christian Bird, Neel Sundaresan, and Shuvendu Lahiri. 2021. MergeBERT: Program Merge Conflict Resolution via Neural Transformers. In Proceedings of the 36th IEEE/ACM International Conference on Automated Software Engineering (ASE '21), 722–734.

[3] Haotian Zhang, Xin Peng, and Zhendong Su. 2024. Merge Conflict Resolution: Classification or Generation? In Proceedings of the 46th International Conference on Software Engineering (ICSE '24).

[4] Sven Apel, Christian Kästner, and Christian Lengauer. 2009. Semistructured Merge: Rethinking Merge in Revision Control Systems. In Proceedings of the 27th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '09), 190–200.

[5] Udi Alon, Omer Levy, and Eran Yahav. 2019. code2seq: Generating Sequences from Structured Representations of Code. In Proceedings of the International Conference on Learning Representations (ICLR '19).

[6] Sven Apel, Olaf Leßenich, and Christian Lengauer. 2012. Structured Merge with Auto-Tuning: Balancing Precision and Performance. In Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering (ASE '12), 120–129.

[7] Gleiph Ghiotto, Leonardo Murta, Márcio Barros, and André van der Hoek. 2020. On the Nature of Merge Conflicts: A Study of 2,731 Open Source Java Projects Hosted by GitHub. IEEE Transactions on Software Engineering, 46(8), 892–915.

[8] Pieter Godefroid, Hila Peleg, and Rishabh Singh. 2017. Learn&Fuzz: Machine Learning for Input Fuzzing. In Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE '17), 50–59.

[9] Georgios Gousios, Margaret-Anne Storey, and Alberto Bacchelli. 2016. Work Practices and Challenges in Pull-Based Development: The Contributor's Perspective. In Proceedings of the 38th International Conference on Software Engineering (ICSE '16), 285–296.

[10] Jiang Gu, Zichao Li, and Vincent O. K. Li. 2016. Incorporating Copying Mechanism in Sequence-to-Sequence Learning. In Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL '16), 1631–1640.

[11] Xin Gu, Hongyu Zhang, and Sunghun Kim. 2018. Deep Code Search. In Proceedings of the 40th International Conference on Software Engineering (ICSE '18), 933–944.

[12] Morris Hollander and Douglas A. Wolfe. 1973. Nonparametric Statistical Methods. Wiley.

[13] Søren Holm Jensen, Anders Møller, and Peter Thiemann. 2009. Type Analysis for JavaScript. In Proceedings of the International Static Analysis Symposium (SAS '09), 238–255.

[14] Brett Johnson, Yuanyuan Song, Emerson Murphy-Hill, and Robert Bowdidge. 2013. Why Don't Software Developers Use Static Analysis Tools to Find Bugs? In Proceedings of the 35th International Conference on Software Engineering (ICSE '13), 672–681.

[15] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M. German, and Daniela Damian. 2014. The Promises and Perils of Mining GitHub. In Proceedings of the 11th Working Conference on Mining Software Repositories (MSR '14), 92–101.

[16] Rishabh Singh, Sumit Gulwani, and Armando Solar-Lezama. 2013. Automated Feedback Generation for Introductory Programming Assignments. In Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '13), 15–26.

[17] Martin Monperrus. 2018. Automatic Software Repair: A Bibliography. ACM Computing Surveys, 51(1), Article 17.

[18] Zhendong Su, Shuvendu K. Lahiri, and Michael D. Ernst. 2008. Dynamic Detection of Atomic-Set-Serializability Violations. In Proceedings of the 2008 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '08), 12–21.

[19] Yingfei Xiong, Lu Zhang, Gang Huang, and Hong Mei. 2009. Detecting Merge Conflicts in Collaborative Software Development. In Proceedings of the 31st International Conference on Software Engineering (ICSE '09), 168–178.

[20] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL '19), 4171–4186.

[21] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. In Findings of the Association for Computational Linguistics: EMNLP 2020, 1536–1547.

[22] Jean-Rémy Falleri, Floréal Morandat, Xavier Blanc, Matias Martinez, and Martin Monperrus. 2014. Fine-Grained and Accurate Source Code Differencing. In Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering (ASE '14), 313–324.

[23] Sven Apel, Christian Kästner, and Christian Lengauer. 2010. Semistructured Merge in Revision Control Systems. In Proceedings of the 4th International Workshop on Variability Modelling of Software-Intensive Systems (VaMoS '10), 13–19.

[24] Earl T. Barr, Yuriy Brun, Premkumar Devanbu, Mark Harman, and Federica Sarro. 2014. The Plastic Surgery Hypothesis. In Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE '14), 306–317.

[25] Yuriy Brun, Reid Holmes, Michael D. Ernst, and David Notkin. 2011. Proactive Detection of Collaboration Conflicts. In Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering (ESEC/FSE '11), 168–178.

[26] Suman Chakraborty, Yufan Ding, Miltiadis Allamanis, and Baishakhi Ray. 2022. CODIT: Code Editing with Tree-Based Neural Models. IEEE Transactions on Software Engineering, 48(4), 1385–1399.

[27] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning Phrase Representations Using RNN Encoder–Decoder for Statistical Machine Translation. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP '14), 1724–1734.

[28] Colin Clement, David Drain, John Timcheck, Alexey Svyatkovskiy, and Neel Sundaresan. 2020. PyMT5: Multi-Mode Translation of Natural Language and Python Code with Transformers. In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP '20), 9052–9065.

[29] Karlovs-Karlovskis, U. 2024. Generative Artificial Intelligence Use in Optimising Software Engineering Process. Applied Computer Systems, 29(1), 68–77.