DesignEvo：跨提交恢复和跟踪仓库级已实现设计系统

MINGYUE YUAN，澳大利亚新南威尔士大学JINBO YANG，
中国北京理工大学JIESHAN CHEN，澳大利亚 CSIROAORUI
YANG，中国浙江大学AARON QUIGLEY，澳大利亚国立大学
GELAREH MOHAMMADI，澳大利亚新南威尔士大学
ZHENCHANG XING，澳大利亚 CSIROBOXI WU，中国浙江大学
DEHAI ZHAO*，中国浙江大学

arXiv:2611.32020v1 [cs.SE] 24 Nov 2026

现代前端应用程序依赖设计系统来保持页面和发布版本之间的一致性。然而，文档和组件库无法完整捕捉界面模式在实践中如何实现、复用和调整。现有的用户界面分析技术主要关注视觉缺陷、界面结构或元素对应关系，对于在代码仓库级别自动恢复和跟踪已实现设计系统的支持有限。为弥补这一空白，我们提出了 DesignEvo。该方法结合屏幕截图、DOM 结构、元素几何信息和样式值来恢复功能性 UI 组。DesignEvo 将视觉语言模型的判断限制在由程序生成的候选区域内，把恢复出的组和样式汇总为仓库级记录，并推断相邻提交之间的生命周期状态。我们在 10 个开源代码仓库上评估了 DesignEvo，这些仓库包含 317 个页面版本实例、3,478 个标注的功能性 UI 组以及 2,435 个生命周期状态标签。DesignEvo 在组恢复任务上取得了 0.85 的 F1 分数，总体生命周期状态准确率达到 0.84，而最强的生命周期基线为 0.71。消融实验展示了候选约束、功能类别和样式证据各自的贡献。最终生成的记录为反复出现的 UI 模式及其演化提供了结构化、可追溯的证据，从而支持前端维护和变更审计。

附加关键词和短语：设计系统、前端代码仓库、用户界面演化、用户界面分析

ACM 参考格式：
Mingyue Yuan、Jinbo Yang、Jieshan Chen、Aorui Yang、Aaron Quigley、Gelareh Mohammadi、Zhenchang Xing、Boxi Wu 和 Dehai Zhao。2026 年。DesignEvo：跨提交恢复和追踪仓库级已实现设计系统。1，1（2026年11月），31 页。https://doi.org/10.1145/nnnnnnn.nnnnnnn

*通讯作者。

作者联系方式：Mingyue Yuan，新南威尔士大学，澳大利亚，mingyue.yuan@unsw.edu.au；Jinbo Yang，北京理工大学，中国，jinboy@bit.edu.cn；Jieshan Chen，CSIRO，澳大利亚，jieshan.chen@csiro. au；Aorui Yang，浙江大学，中国，22551167@zju.edu.cn；Aaron Quigley，澳大利亚国立大学，澳大利亚，aaron.quigley@anu.edu.au；Gelareh Mohammadi，新南威尔士大学，澳大利亚，g.mohammadi@ unsw.edu.au；Zhenchang Xing，CSIRO，澳大利亚，zhenchang.xing@data61.csiro.au；Boxi Wu，浙江大学，中国，wuboxi@zju.edu.cn；Dehai Zhao，浙江大学，中国，dehai.zhao@zju.edu.cn。

允许出于个人或课堂使用目的，免费制作本作品的全部或部分纸质或数字副本，但前提是不得出于营利或商业利益制作或分发副本，并且副本必须附有本声明及第一页上的完整引文。对于本作品中由作者以外的其他方拥有版权的组件，必须予以尊重。注明出处后进行摘要是允许的。除此之外，如需复制、再版、发布到服务器或重新分发到列表，必须事先获得具体许可和/或支付费用。请通过 permissions@acm.org 申请许可。© 2026 版权所有归所有者/作者所有。出版权已授权给 ACM。ACM
XXXX-XXXX/2026/11-ARThttps://doi.org/10.1145/nnnnnnn.nnnnnnn

1 引言

现代软件用户界面（UI）通常构建于内部**设计系统**之上：包括可复用组件库、样式值和交互规范，它们编码了产品用户界面应遵循的视觉、结构和行为规则 [20]。从软件工程的角度来看，设计系统充当用户界面层的**质量契约**：这是一份不断演进的规范，数百个页面和数十名工程师都应共同遵循。它的存在会影响下游的用户界面质量保证、可访问性检查、跨团队治理以及长期前端维护 [11, 17, 35, 36, 66, 67]。然而，文档中记录的规范并不总能反映界面模式在实践中的实现和调整方式。随着前端应用程序不断演进，经过局部修改的组件、不一致的样式以及过时的模式可能会在各个页面中不断累积，使得理解和维护用户界面的一致性变得愈发困难。

设计系统仅被**部分**记录。其权威形式存在于前端实现中——即实际呈现给用户的组件、类层次结构、样式令牌和布局模式。文档化的设计系统与已实现的设计系统是两个必须保持一致的工件，但在实践中它们经常出现偏离。工程师引入了一个在本地看来合理的卡片变体，却从未将其回馈到共享库中；某种颜色被硬编码，而不是引用共享令牌；设计系统已弃用某种模式后，该模式仍在三个页面中持续存在；代码中形成了文档尚未认可的事实约定。我们将所有此类差距称为**设计系统—实现漂移**。有些漂移是应予纠正的非预期偏差；另一些则是合理的演化信号，设计系统本身应当吸收这些信号 [41]。区分二者是一项软件质量管理决策，需要设计系统层面的证据。

基于基础模型的用户界面生成 [3, 4, 19, 55] 加剧了这一质量管理难题。代理驱动的变更可以在单个步骤中波及数十个文件、多个页面和多个组件系列，其到来的速度快于人工审查，也可能早于下游代理使用这些变更。无论变更由人还是模型编写，审查者面对的核心问题都相同——“这项变更会如何推动已实现设计系统发生变化，团队是否应该接受这种变化？”

以往研究提出了多种解决方案，但并未直接恢复此类仓库级结构化设计系统记录。视觉用户界面质量保证技术能够检测预定义的显示故障、可访问性违规，或偏离明确设计指南的情况 [1, 11, 15, 16, 30, 50, 51,65, 66]。用户界面结构恢复、GUI 分组和截图到代码技术能够识别单个屏幕中的元素或感知区域 [6, 9, 31, 40, 57, 59, 62, 63, 69]。用户界面演化和测试修复技术能够建立跨版本的元素对应关系 [5, 22, 38, 47–49]。然而，这些研究通常假定存在预定义参考，在屏幕或元素粒度上运行，或侧重于测试维护，而非将功能组、变体和样式规范恢复为仓库级记录。

恢复这些关系带来了三个挑战：第一，合适的分析粒度介于低层实现产物与整页屏幕截图之间：新增元素可能是新的组件族、现有组件族的变体，也可能仅仅是既有项目规范的另一处出现。第二，设计系统级一致性并不等同于视觉标识。两个区域的外观可能不同，却遵循相同的项目规范；两个区域看起来相似，却可能依赖不同的结构或样式值。第三，恢复出的功能性 UI 组和族记录会随着提交和页面不断演变，其中还包括没有先前同页面对应版本的新页面；因此，生命周期推断需要跨页面和版本，结合样式值证据来匹配功能组与变体，而不能只对齐新旧元素。

我们提出了 DesignEvo，这是一种跨工件方法，用于恢复仓库级已实现设计系统记录，并跟踪其功能性 UI 组和族在各提交中的生命周期状态演变。给定选定的仓库版本后，DesignEvo 会渲染目标页面，并从屏幕截图、DOM 层级结构、UI 元素框和样式记录中收集运行时证据。它从交互容器、共享导航目标和 DOM 容器中生成候选区域，并约束分类法引导的视觉语言模型仅从这些固定候选中进行选择、移除或合并，以形成功能性 UI 组。最后，DesignEvo 会跨页面聚合恢复出的组和样式值，形成仓库级 UI 组族及其变体，以及仓库共享样式值。对于每个恢复出的族和功能性 UI 组记录，DesignEvo 还会推断其在相邻提交之间的生命周期状态——复用、变更、新增、移除或歧义（表 1）——从而为人工审查人员和下游代理提供每项变更的设计系统级视图，并为决定将其吸收到设计系统中还是针对它进行回滚提供依据。

我们在收集的包含 10 个开源前端代码仓库的数据集上评估了 DesignEvo。该数据集涵盖 37 条页面时间线、317 个页面版本实例、280 个相邻版本转换、3,478 个标注的功能性 UI 组，以及 2,435 个转换级生命周期状态。DesignEvo 在功能性 UI 分组恢复任务中的 F1 值达到 0.85，优于直接 VLM 框生成和 UI 分组基线。在生命周期推断方面，DesignEvo 的总体生命周期状态准确率达到 0.84，而在我们的设置中，最强基线的准确率为 0.71。消融实验结果表明，固定候选框、功能类别证据、渲染样式变化证据和仓库级样式聚合均有助于提升最终性能。我们的实现和数据集可在 https://github.com/Mingyue-eva/Designevo 获取。本文的贡献如下：

- 我们定义了一种仓库级已实现设计系统表示，将功能性 UI 组、重复出现的族、变体和共享样式值与其页面出现位置及跨版本对应关系相关联，从而支持对用户界面演化进行可追溯分析。
- 我们提出了 DesignEvo，这是一种跨工件方法，将程序派生的候选区域与分类法引导的视觉语言模型判断相结合，以恢复功能性 UI 组；随后通过代码仓库级聚合和生命周期状态推断，跟踪它们跨提交的演化。
- 我们构建了一个包含 10 个开源前端仓库的标注基准数据集，并通过基线比较、消融实验和模型敏感性分析，评估组恢复和生命周期状态推断，展示了受约束定位、功能类别和样式证据的作用。

2 相关工作

2.1 自动化 UI 质量保证

自动化 UI 质量保证越来越多地利用渲染界面来检测用户可见的问题。OwlEye 可检测并定位显示故障 [51]。dVermin 通过比较默认显示比例和放大显示比例下的界面来识别缩放故障 [50], ，而 Chiou 等人[11] 则检测响应式网页中的重排可访问性问题。AccessiText 会审核文本呈现要求[1],， MotorEase 则检查运动障碍无障碍触控目标和间距 [30]。这些方法展现了渲染证据的价值，但其判断依赖预定义的缺陷类别、变换或无障碍规则。其他研究则利用更广泛的设计知识来评估界面。Yang 等人[65] 和 Seenomaly[70] 将 Material Design 指南转化为可操作的方法，以检测视觉设计异味和问题动画效果。Duan 等人[16]根据书面设计启发式规则生成反馈，而 UICrit[15] 则从专家批评和质量评分中学习。UIClip [60] 学习屏幕截图的通用质量和相关性。DesignRepair [66] 利用组件级和系统级 Material Design 知识来审核和修订前端实现，而 GUIRepair [26]

则利用视觉问题观察来理解故障，并通过重新渲染的输出验证补丁。这些方法有助于评估或修复，但通常依赖明确的参考依据，例如指南、带标签的评述、错误报告、质量标签或持续维护的设计系统。

DesignEvo 解决了进行此类审计的一项先决问题。在许多代码仓库中，项目特定的设计知识隐含在反复出现的功能区域、可复用变体、共享样式值、页面、文件和提交中。DesignEvo 不会判断某项更改是否可接受或存在缺陷，而是恢复这些已实现的仓库级设计系统记录，以供后续比较。

2.2 设计系统恢复的逆向工程

我们将设计系统恢复定义为恢复仓库级已实现设计系统记录：功能性 UI 组、UI 组族、UI 组变体、仓库共享样式值和生命周期状态。这一目标与逆向工程有关；逆向工程通过系统组件及其关系，重建更高层次的表示 [10]。以往关于 UI 和代码仓库的研究恢复了交互式实现、应用程序故事板、架构、文档和可追溯性 [8, 25, 29, 34, 68, 71]。DesignEvo 延续并拓展了这方面的工作，将代码、DOM、渲染后的 UI、样式和仓库历史中的证据提升为已实现的仓库级设计系统。源代码级复用提供结构性证据，而恢复出的族和变体则描述了反复出现的功能模式在不同页面上的实际呈现方式；它们不必与文档化组件库的边界一致。已有研究对 HTML、通用 Web 应用程序和 React 组件的重构进行了归类 [17, 23, 39]。其他技术则通过静态分析和动态分析识别未使用的选择器、重复声明、冗余规则和被覆盖的样式 [21, 35, 36, 44]。这些分析揭示了共享样式值，但主要用于实现维护，而非设计系统恢复。

DesignEvo 还与 UI 结构恢复和前端维护相关。元素检测方法用于识别 UI 元素并合并碎片化的视觉图层 [6, 9, 31, 62]，，而感知分组 [63] 和 GUI 分组 [69] 则根据视觉、空间、语义或视图层级线索推断区域 [7, 58, 61]。截图转代码系统采用相关的分解方法来生成界面 [40, 57, 59]。Web 和 CSS 分析会整理 HTML、Web 和 React 重构，或检测未使用的选择器、重复的声明、冗余规则以及被覆盖的样式 [17, 21, 23, 35, 36, 39, 44]。这些技术能够揭示结构或样式证据，但主要针对单个屏幕、代码维护、重复或重构。DesignEvo 将这些证据类型结合起来，以恢复 UI 组族和共享样式值，并通过功能组、变体和使用上下文来解释重复出现的值。

2.3 用户界面演化分析

软件制品会持续经历提交和发布版本的演化。分析这类演化需要在不同版本的软件制品之间建立对应关系，即使其中存在新增、删除、重命名、布局变更和重构等情况。在 UI 领域，GCAT 总结移动应用版本之间的 GUI变更 [38], ，而 GuiEvo 支持移动GUI演化分析 [47]。WebEvo 是与 DesignEvo 最密切相关的工作；它检测网页中的语义结构变化，并映射对应的元素 [48]。UI测试修复技术也会在界面演化过程中建立元素对应关系。WATER [12] 和 VISTA [49] 分别利用属性信息和视觉信息匹配元素。SFTM [5] 执行灵活树匹配。UITESTFIX [32] 提取元素信息，而 EDIT DIS [64] 基于这些信息，仅依据 XPath相似度执行匹配。这些方法为匹配演化中的UI元素提供了具有代表性的策略，但其主要目标是维护测试定位器和测试稳定性。更通用的软件演化技术会跟踪历史记录、重构、迁移以及跨制品一致性 [24, 29, 33, 42, 45, 52–54], ，但通常聚焦于代码

实体、重构实例、迁移规则、文档链接或 DOM 变更。DesignEvo 以跨版本对应关系为基础，但将分析目标从单个元素、页面结构和测试定位器提升至已实现的仓库级设计系统记录的生命周期。其混合分析结合确定性的运行时证据与受约束的语义判断，将页面出现位置与整个代码仓库中的功能族、变体和共享约定联系起来。这种已实现的表示提供了仅凭视觉差异或文档化组件目录无法解决的证据。

3 背景

3.1 前端实现中的设计系统

现代 Web 应用程序使用设计系统来协调不同产品、团队和发布版本中的界面决策。设计系统通常由设计令牌、可复用组件和界面模式构成，并由实现库、文档和使用指南提供支持。设计令牌编码了共享的视觉决策，例如颜色、排版、间距和阴影 [13]。可复用组件封装按钮、输入框和卡片等界面元素，而界面模式则将这些元素组织成更大的功能区域，例如表单、导航面板和对话框 [18, 43]。Carbon、Ant Design、Material Design 和 Fluent 等成熟设计系统展示了这些元素如何通过文档和组件库进行组织和呈现 [2, 20, 28, 37]。

在前端实现中，这些设计决策会以不同形式呈现。组件可能通过共享库实例化、在本地扩展，也可能使用相似的结构独立实现。同样，样式值可能通过命名令牌、CSS 变量、工具类或硬编码声明来表示。因此，相同的功能模式可能具有不同的源代码结构，而共享组件则可能根据其属性、内容和上下文呈现不同的渲染变体。总的来说，应用程序中体现的重复功能区域、组件变体和共享视觉规范，共同构成其已实现设计系统。

随着应用程序不断演化，这些实现会通过组件修改、本地适配、样式更新，以及界面模式的引入或移除而发生变化。例如，卡片可能会增加控件，表单可能会采用不同的布局，或者共享颜色可能会被本地定义的值取代。这些变化可能是有意的改进，也可能导致预期规范与其实现之间，或预期遵循相同规范的页面之间出现不一致。我们将这些不一致称为设计系统实现漂移。业界讨论将漂移分为令牌、组件变体、模式、文档和行为漂移 [41]。这些类别反映了设计系统演化的不同方面，但仅仅识别出不一致，并不能据此确定底层变化究竟存在缺陷，还是有其合理性。

3.2 恢复已实现设计系统的证据

理解已实现设计系统，需要识别反复出现的功能模式、这些模式的渲染实例及其样式规范，以及这些关系如何随版本变化。现有设计工件提供了互补证据，但它们所捕捉的内容以及对当前实现的直接呈现程度各不相同。

设计文件和文档。设计文件描述预期的布局、组件和视觉属性，而设计系统文档则规定可复用元素、支持的变体和使用规则。例如，Carbon 会记录按钮变体及其预期用途 [27], ，而 Design Tokens Format Module 则规范了命名设计值的表示方式

设计值 [13]。然而，这些工件不一定能确定某种模式在特定应用程序中的实例化位置，也不一定能说明它在本地进行了怎样的调整。因此，它们能否用于实现分析，取决于其可用性、与所分析版本的同步情况，以及与实际页面和渲染状态的对应关系。

源代码级组件与样式。前端源代码会呈现组件声明、导入、属性和样式定义。这些信息可以揭示明确的组件复用和共享样式依赖。然而，功能等价性并不总是对应源代码级同一性：独立实现的组件可能发挥相同作用，而共享组件也可能根据其属性和子元素渲染出不同的变体 [46]。同样，等价的视觉值可能源自命名令牌、计算样式或硬编码声明。因此，仅凭源代码级关系，无法完整描述用户实际体验到的重复出现的模式。

渲染界面与视觉变化。屏幕截图和渲染后的页面结构呈现了已实现界面的外观、构成和空间组织。比较不同版本可以揭示新增、移除和视觉修改。然而，仅凭像素级差异无法确定变更区域是否发挥相同功能，或是否属于跨页面重复出现的模式。UI演化技术可以建立不同版本间的元素对应关系 [48]，但这些对应关系并不一定能够识别仓库级模式族或其共享样式规范。

总而言之，这些工件提供了刻画已实现设计系统所需的不同部分证据。要恢复仓库级视图，就需要将功能区域与其渲染实例关联起来，识别跨页面重复出现的模式和共享样式，并跟踪它们在不同版本中的变化。当现有文档或工具维护了此类对应关系时，便可以直接支持这项分析；否则，就必须从实现中恢复这些关系。这些恢复出的关系刻画了可观测的设计系统演化，但不会独立判断某项变更是否符合预期规范。

4 方法论

DesignEvo 恢复代码仓库中隐含的、项目特定的已实现设计系统，并跟踪其跨提交的演化。关键挑战在于，这类设计知识分散在多个工件中：源代码包含样式声明并复用用户界面结构，DOM 树呈现渲染后的 UI元素层级，而渲染页面则揭示用户所看到的视觉布局。单个工件都不足以完成这项任务，因为代码层面的相似性未必意味着视觉或功能相似性，而视觉上相似的 UI 组可能依赖不同的实现结构。DesignEvo 通过对源代码、DOM 树和渲染页面开展跨工件分析来应对这一挑战。其核心设计选择是将候选框生成与基于VLM的语义判断分离：DesignEvo 根据渲染后元素的位置和尺寸生成 UI元素框，并要求 VLM 从这些固定候选框中识别功能性 UI 组，而不是直接绘制边界框。随后，这些恢复出的组、样式值和跨提交对应关系会被汇总为仓库级已实现设计系统记录。

4.1 概述

如图 1所示，DesignEvo 以仓库历史、一组目标页面以及待比较的相邻提交对作为输入。对于每个渲染后的页面版本，它会恢复功能性 UI 组及其对应的样式值。对于每个仓库版本，它会将页面级记录汇总为仓库级已实现设计系统记录：UI 组族、UI 组变体和仓库共享样式值。对于每个相邻提交对，它会输出生命周期状态，指明这些恢复出的记录是新增、复用、变更、移除还是存在歧义。该

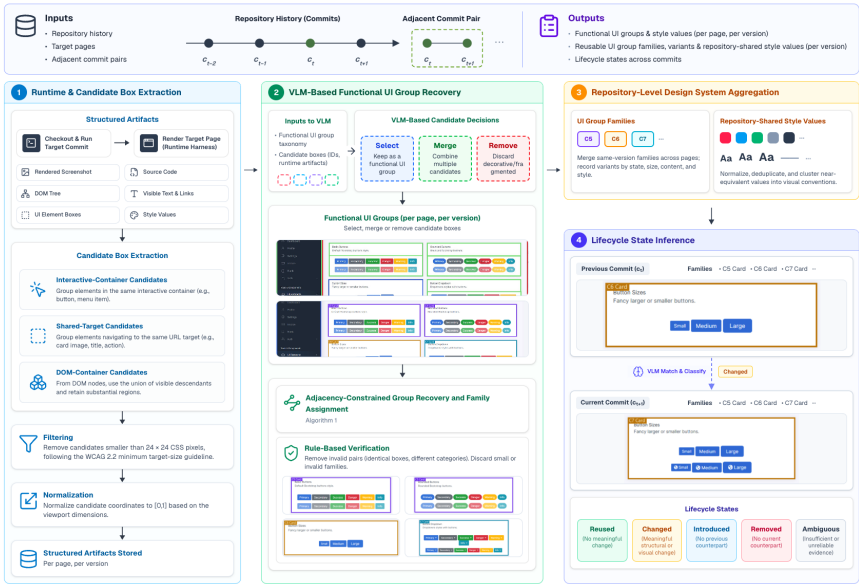


图 1。**DesignEvo 概览**。DesignEvo 以仓库历史、目标页面和相邻提交对作为输入。对于每个页面版本，它会渲染目标提交、收集跨工件运行时证据，并从交互容器、共享 URL 目标和 DOM 容器中提取固定候选框。随后，分类法引导的视觉语言模型会选择、合并或移除这些候选项，以恢复功能性 UI 组和同版本组族。DesignEvo 将恢复出的组和样式值汇总为仓库级设计系统记录，并比较相邻提交中的记录，以推断出复用、变更、新增、移除和歧义等生命周期状态。

流水线包含四个阶段。首先，在运行时和候选框提取阶段，运行时工具套件会渲染页面，收集屏幕截图、源代码和样式值，提取 DOM 树、UI 元素框、可见文本和链接，并存储这些结构化产物。随后，DesignEvo 从三个来源提取候选框。其次，视觉语言模型接收功能性 UI 组分类法和候选框，并判断哪些候选项对应功能性 UI 组，以及应如何解读这些 UI 组。第三，DesignEvo 将页面级功能性 UI 组和样式值汇总为仓库级 UI 组族和仓库共享样式值。第四，DesignEvo 比较相邻提交中的恢复记录，生成跨提交匹配决策和生命周期状态。

4.2 功能性用户界面分类体系的构建

DesignEvo 依靠功能性 UI 组分类法来指导基于视觉语言模型的组恢复。功能性 UI 组的概念**功能性 UI 组**源自 Atomic Design [18]：它最接近 organism 层级，即多个 UI 元素和小部件组合成一个连贯的界面区域，以实现可辨识的功能目的。为使这一单元适用于自动化恢复，我们构建了功能性 UI 组分类法。该分类法并非旨在引入新的 UI 组类型，也不打算罗列所有低层级 UI 元素和实现组件。相反，它概括并合并了设计系统组件中已有的常见功能区域，为视觉语言模型提供明确的页面区域分类法。

分类法构建。 我们通过审阅 designsystems.surf 列出的九个广泛使用的设计系统的公开组件目录和使用指南来构建分类法 [14]，包括苹果公司、Atlassian、蚂蚁集团、亚马逊、IBM、微软、Tailwind、Adobe 和谷歌。组件目录提供了一个切实可行的起点，因为它们描述了反复出现的界面单元及其交互用途。由于不同设计系统常常对功能相似的区域使用不同名称，当源名称符合以下两个标准时，我们会将其合并：它们服务于相同的主要面向用户的用途，并且通常被视为或编辑为一个连贯的页面区域。例如，次级标签页眉合并子 Header、应用栏和导航栏，因为这三者都指代一个顶层区域，用于集合品牌塑造、全局导航和页面级控件。

分类体系结构。 最终的分类体系包含九个主要类别：全局元素、导航、操作、输入、数据展示、状态、叠加层、图像类型和第三方嵌入内容。这些主要类别包含二级功能性 UI 组标签，包括页眉、页脚、面包屑导航、搜索、侧边栏、操作栏、输入、选择器/下拉选择器、表单、卡片、统计面板、可视化、时间线、网格、代码片段、数据表、列表、主视觉区、内容块、消息、空状态、加载器、进度、对话框/模态框、徽标/广告区块和嵌入式小部件。我们还报告了经审查的设计系统组件名称与这些分类标签之间的映射关系。我们在随本文提交的补充材料中提供了完整的功能性 UI 组分类法。

映射与筛选标准。只有当源名称主要表示基础概念、实现工具或低层级 UI 元素，而非功能性 UI 组时，才不对其进行映射。我们的标准是：该项通常是否构成一个独立的功能区域，其角色可以在不同页面和提交之间进行比较。例如，按钮、图标、标签、链接、徽章、滑块和开关都包含在功能性 UI 组中，对于分析 UI 元素组成和样式值仍然有用，但不会被提升为功能性 UI 组标签。单个按钮或图标的样式值可能发生变化，但其含义通常由周围的组决定，例如搜索、操作栏、表单、卡片或对话框/模态框。这一映射与筛选流程涵盖了常见的设计系统组件，同时保持标签空间紧凑且稳定，以便进行 UI 组族恢复和生命周期推断。

4.3 运行时与候选框提取

对于每个目标提交，DesignEvo 都会检出代码仓库，在配置好的运行时工具套件下构建或运行应用程序，并打开目标页面。运行时工具套件会记录源代码、渲染页面截图、DOM 树、可见文本和链接、UI元素框及样式值。此运行时视图将代码级变更与渲染页面联系起来：代码差异可能显示类名或样式值发生了变化，但渲染结果能够确定受该变化影响的面向用户的区域。候选框是从渲染页面中自动提取的区域，可能对应一个功能性 UI 组或其一部分。DesignEvo 会在下一步调用视觉语言模型之前提取候选框。候选集合有意设置得较为宽泛，因此可能包含重叠或嵌套的区域，因为下一阶段会要求视觉语言模型从中选择、移除或合并候选项，以形成多个功能性 UI 组。

DesignEvo 从三个来源提取候选框。**交互容器候选框**将属于同一交互容器的可见 UI 元素归为一组，例如按钮、链接、菜单项或可点击卡片。**共享目标候选框** 将导航至同一 URL 目标的相邻可见 UI 元素归为一组，例如卡片图片、标题和操作，它们都会打开同一个详情页。**DOM 容器候选框** 可恢复未被共享交互容器或共享 URL 目标捕获的功能性 UI 组。对于每个可见 DOM 节点，DesignEvo 根据其可见后代 UI 元素框的并集计算外接矩形，并将该节点保留

为候选框，前提是矩形包围的视觉区域具有一定规模，且面积充足、包含可见内容或交互目标。此规则适用于表单、搜索区域、表格区域、图表面板、页眉、侧边栏和内容卡片等区域；这些区域中的元素可能共同承担某项功能，但不一定共享同一个交互容器或导航目标。此外，DesignEvo 会筛除尺寸小于 24×24 CSS 像素的候选框，并遵循 WCAG 2.2 针对指针输入的最小目标尺寸指南 [56], ，同时将其坐标归一化至 $[0,1]$ 视口尺寸对应的范围。

4.4 基于视觉语言模型的功能性 UI 组恢复

DesignEvo 将固定候选框提取与 **基于视觉语言模型的候选决策**相结合。这种设计避免要求视觉语言模型进行开放式视觉定位，因为这可能会产生不稳定或无法复现的坐标。相反，DesignEvo 会先从渲染后的页面中提取固定候选框，然后由视觉语言模型决定是否将每个候选框选择为功能性 UI 组、将其移除，或与附近的候选框合并。这种方式将坐标生成与功能解释分离开来，并使每个恢复出的组都能追溯到具体的运行时记录。

给定候选框和运行时工具套件记录，视觉语言模型会根据第 4.2 节定义的分类体系恢复功能性 UI 组。对于每项决策，视觉语言模型都会返回所选候选 ID、功能类别和简短理由。它只能引用现有候选 ID。如果多个候选框共同构成一个可见且功能连贯的区域，视觉语言模型会输出一个包含所有相应候选 ID 的功能性 UI 组。如果候选框属于装饰元素、与其他框重复，或只是更大区域的不完整片段，视觉语言模型会将其标记为 移除。随后，程序会将所选候选框的并集作为每个功能性 UI 组的最终边界框。视觉语言模型不会创建新坐标，也不会编辑框的值。

邻接约束组恢复与族分配。算法 1 首先，利用邻接图从候选框中恢复功能性 UI 组。邻接图会连接彼此重叠、接触或之间没有可见间隙的候选框。对于每个连通分量，视觉语言模型会决定保留候选框、合并多个候选框，还是移除装饰性、重复或不完整的片段。DesignEvo 使用保留或合并后的候选框之并集，将其具体化为功能性 UI 组。随后，它会将同一仓库版本中的组分配到 UI 组族中。DesignEvo 不会按功能类别对所有组进行全局分组，而是针对类别兼容且空间相邻的组构建邻居图，并使用广度优先搜索扩展各个族。对于每一对相邻组，视觉语言模型会根据其功能类别、布局、UI 元素组成、样式值、文本和周围上下文，判断它们是否属于同一族的 UI 组变体。这种局部策略既保留了可能的合并关系，也避免了比较彼此无关的页面区域。

基于规则的验证。DesignEvo 会应用基于规则的验证，以减少视觉语言模型产生的幻觉。它会移除无效的成对决策：当两个组的边界框完全相同但功能类别不同时，通常说明它们是对同一区域的重复解释，而非同一 UI 组族的变体。如果移除这类配对后导致某个暂定族断开，或其中剩余的组少于两个，DesignEvo 就会舍弃该族分配。

4.5 仓库级设计系统聚合

DesignEvo 将页面级恢复结果汇总为每个仓库版本的仓库级已实现设计系统记录。这些记录由两个互补部分组成：UI 组族和仓库共享样式值。二者共同近似呈现了代码仓库中体现的已实现设计系统，因为它们既记录了哪些功能性 UI 组在各页面间复用，也记录了用于呈现其视觉外观的样式值。

算法 1 邻接约束的功能性 UI 组恢复与族分配

要求： 渲染页面截图 I、候选框 B、运行时记录 R、功能性 UI 组分类法 T
确保： 功能性 UI 组 G 和同版本 UI 组族分配 F

$G \leftarrow \emptyset$
在 B 中的候选框上构建邻接图 A ▷ 连接边连接相互重叠、接触或之间没有视觉间隙的框。
对 A 中的每个连通分量 C **执行**
  $D_C \leftarrow$ 对 C 中候选框的视觉语言模型决策 ▷ 每个决策为选择、合并或移除。
  对每个决策 $d \in D_C$ **执行**
    如果 d 是选择或合并**则**
      $S_d \leftarrow$ 由 d选择的候选 ID
      从 S_d实例化一个功能性 UI 组 g_d
      将 g_d的边界框设置为 S_d中各框的并集
      为 g_d 分配视觉语言模型返回的功能类别
      将 g_d 添加到 G
    结束 if
  结束 for
结束 for
构建邻居图 Nover G ▷ 边连接具有兼容功能类别的空间相邻组。
$F \leftarrow \emptyset$
$Assigned \leftarrow \emptyset$
对于每个组 $g \in G$ **执行**
  如果 $g \notin Assigned$ **则**
    初始化暂定族 $f \leftarrow \{g\}$ 和已接受边 E_f $\leftarrow \emptyset$
    初始化一个BFS队列 $Q \leftarrow [g]$
    当队列不为空时 **执行**
      从 Q中弹出一个组 u
      对于 v 中的每个邻居 u，在 N中，其中 $v \notin f$且 $v \notin Assigned$ **执行**
        询问视觉语言模型： u 和 v 是否是同一 UI组族的 UI 组变体
        如果视觉语言模型接受该对**则**
          将 v添加到 f，并将边 (u, v) 添加到 E_f
          将 v 推入 Q
        结束 if
      结束 for
    结束 while
  结束 if
结束 for
返回 G ,F

UI 组族聚合。 DesignEvo 使用视觉语言模型，将目标页面中同一版本的族分配结果合并为 UI 组族。当组具有兼容的功能类别、相同的主要面向用户的用途，以及可辨识的共享结构或视觉规范时，就会将它们合并。状态、尺寸、内容或视觉呈现方面的差异，则记录为 UI 组变体，而不是单独的族。

仓库共享样式值聚合。 DesignEvo 通过三个步骤将组级样式值聚合为仓库级样式记录。首先，它应用频率加权，识别在恢复出的功能性 UI 组中反复出现的值，并降低偶发的页面专属样式的影响。其次，对于同一样式属性下的值，它会进行近似等价分组，例如感知相似的颜色、相邻的间距值，或通过不同字面值或变量表示的等价圆角值。第三，它应用基于分类体系的命名，将反复出现的值组映射到共享视觉规范，例如调色板、间距尺度、排版尺度和圆角尺度。此聚合既保留规范化的仓库级记录，也保留底层实现记录，包括 CSS 类名、可解析时的声明值、渲染视口中的计算值，以及相关的功能性 UI 组。

4.6 生命周期状态推断

在恢复每个版本的仓库级设计系统记录后，DesignEvo 推断生命周期状态，使这些记录可用于 UI 变更审计。仓库级记录在各次提交之间被新增、复用、变更、移除或标记为歧义，能够为判断

表 1。DesignEvo 使用的生命周期状态。

状态	定义
复用	相同的仓库级记录，其视觉、结构或样式值没有发生有意义的变化。
变更	具有有意义的布局、尺寸、样式值、内容排布、UI 元素组成或惯例级变更的相同仓库级记录 .....、UI 元素组成或惯例级变更。
新增	在当前版本中出现，但没有先前对应项。
移除	存在于先前版本中，但当前版本没有对应项。
歧义	对应关系存在噪声、不可靠或缺乏充分支持。

表 2. DesignEvo 数据集的代码仓库级构成。

代码仓库	主要技术栈	时间线	实例	状态转移	功能性 UI 组	生命周期状态 标签
BootstrapDash/PurpleAdmin-Free-Admin-模板	Bootstrap	3	21	18	221	147
ColorlibHQ/AdminLTE（v3 分支）	Bootstrap/jQuery	5	38	33	500	364
StartBootstrap/startbootstrap-sb-admin-2	Bootstrap	4	39	35	292	249
YafetGetu/Admin-Dashbored	Django/HTML	1	11	10	75	76
adminkit/adminkit	Bootstrap	5	40	35	406	261
creativetimofficial/soft-ui-dashboard	Bootstrap	2	16	14	144	104
design-sparx/antd-multipurpose-dashboard	React/Ant Design	5	20	15	361	209
puikinsh/CoolAdmin	Bootstrap	4	59	55	554	458
themeselection/sneat-bootstrap-html-admin-template-free	Bootstrap	4	31	27	675	399
zuramai/mazer	Bootstrap	4	42	38	250	168
总计	–	**37**	**317**	**280**	**3,478**	**2,435**

UI 变更是否遵循或偏离项目不断演变的视觉规范提供依据。因此，DesignEvo 会比较相邻提交中的仓库级记录，并为其分配表1中的一种生命周期状态。这些记录包括 UI 组族和仓库共享样式值。随后，视觉语言模型会将每个候选匹配分类为复用、变更或歧义。DesignEvo 将未匹配的当前记录标记为新增，并将未匹配的先前记录标记为移除。对于保留仓库级记录身份，但改变其渲染结构、UI 元素组成、样式值或共享视觉规范的情况，我们使用变更。如果多个先前的 UI 组族对应一个当前组族，或一个先前组族对应多个当前组族，只要角色和主要面向用户的用途保持不变，DesignEvo 仍使用变更，并在原因中记录合并或拆分。这样可以保持生命周期状态空间的紧凑性，避免引入会增加分类复杂度的罕见状态。

5 实验

5.1 研究对象

代码仓库与版本收集。 我们从 10 个开源前端代码仓库构建了 **DesignEvo数据集** ，这些代码仓库涵盖不同的应用领域、项目规模和具有代表性的前端技术栈，包括 Bootstrap、React、Ant Design、Tailwind CSS 和 Django。所选代码仓库包含组件丰富的多页面界面和反复出现的用户界面结构，适合用于研究仓库级功能性用户界面组复用及其演化。

对于每个代码仓库，我们获取其提交历史，并保留包含前端 UI 文件实质性修改的提交。我们排除了仅涉及文档、配置、依赖项、注释、格式或文本内容的更改，并过滤掉 UI 差异可忽略的近重复提交。随后，我们检出每个保留的版本，并尝试构建和渲染其中可访问的应用路由。无法以可复现方式完成安装、启动或渲染的版本将被排除。在某次提交中成功渲染的每个路由构成一个页面版本实例；同一路由按时间顺序排列的实例构成页面时间线，相邻的有效实例则定义了我们分析中使用的版本转换 。我们仅保留至少包含两个有效渲染版本且存在可观察用户界面变化的路由。对于每个页面版本实例，我们收集其屏幕截图、DOM 结构、计算样式、元素几何信息、路由和提交元数据。对于相邻版本，我们还生成了视觉和结构比较，以支持标注和基线构建。这些自动生成的工件仅作为证据使用；所有最终标注均经过人工审查。

标注人员结合相应的 DOM 和变更证据，对变更前后的渲染结果进行了审阅。他们识别功能性 UI 组，分配功能类别，并将每个组的生命周期状态标记为复用、变更、新增、移除；当现有证据不足时，则标记为歧义。在页面时间线开始处首次观察到的组被标记为新增。两名标注人员独立审阅生成的比较实例，并对其中 20% 的实例进行了重新标注，以进行质量控制。他们的一致率超过 97%。不一致之处通过讨论和第三方裁定解决；仍然定义不充分的实例则从最终数据集中排除。

数据集统计。 最终数据集包含 10 个代码仓库和 37 条页面时间线，包括 317 个页面版本实例和 280 次版本转换。在这些实例和转换中，我们标注了 3,478 个功能性 UI 组和 2,435 个转换级生命周期状态标签。表 2总结了每个代码仓库的贡献。各个代码仓库贡献的功能性 UI 组数量介于 75 至 675 之间，生命周期状态标签数量介于 76 至 458 之间，这反映了所收集页面时间线在数量、长度和复杂性方面的差异。具体而言，复用 标签占所有生命周期状态标注的 71.01%。其余 28.99% 反映了可观察到的用户界面演化，包括 变更（15.56%）、新增（7.06%）、移除 （6.24%）和歧义 （0.12%）标签，具体定义见表 1。因此，该数据集同时包含持久化的用户界面结构和跨版本变更，可用于评估功能性 UI 组匹配和生命周期状态推断。主要和次要功能性 UI 组类型的详细分布见补充材料。

5.2 研究问题

我们通过三个研究问题评估 DesignEvo。每个研究问题都针对方法论中的一项具体设计选择，然后报告相应的实验设置和结果。

- **RQ1：功能性 UI 分组恢复**。DesignEvo 从一次代码仓库提交中恢复功能性 UI 组的准确度如何？与基线方法相比又如何？
- **RQ2：生命周期状态推断**。对于跨相邻提交恢复的功能性 UI 组和族记录，DesignEvo 推断其生命周期状态标签的准确度如何？与基线方法相比又如何？
- **RQ3：核心证据与约束的消融**。DesignEvo 的核心证据来源和流水线约束在多大程度上有助于功能性 UI 分组恢复、仓库级设计系统记录构建和生命周期状态推断？

5.3 共享实验方案

视觉语言模型配置。主要实验使用实验日志中记录的 GPT-5.5 视觉语言模型，因为它支持图像理解、结构化 JSON 输出以及候选框语义判断。我们将 temperature 设为 0，使用网关默认的 top-p，并为 JSON 输出设置特定于任务的令牌上限。我们保留提示词和原始响应，以供审计。由于 DesignEvo 依赖视觉语言模型判断来恢复功能性组、进行族分配和推断生命周期状态，我们还使用 Qwen2.5-VL 对 RQ1 和 RQ2 开展模型敏感性研究，并使用相同的输入、提示词、候选集合和评分脚本。

评估指标。对于 RQ1，我们按交并比从高到低贪心匹配预测框和参考框。只有当预测框与某个参考框匹配、交并比达到 ≥ 0.9 且次级功能标签正确时，才将其计为真正例。未匹配的预测框和标签不匹配的匹配结果属于假正例；未匹配的参考框属于假负例。我们报告精确率、召回率、F1 值和平均边界框交并比，其中平均边界框交并比仅根据真正例匹配计算。

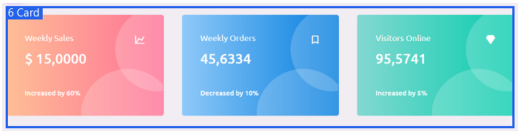
对于 RQ2 和 RQ3 中的生命周期状态推断，如果每种方法为每个标注状态生成一个标签，我们报告各状态的准确率；如果方法可能生成多余或缺失的记录，我们则报告精确率、召回率和 F1 值。所有汇总精确率、召回率和 F1 值均根据整个评估范围内的精确计数进行微平均。可比较的方法使用相同的标注页面版本、状态转移、候选对应关系和评分脚本。

5.4 RQ1：功能性 UI 分组恢复

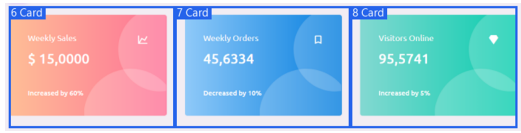
RQ1 评估 DesignEvo 将候选框提取与基于视觉语言模型的功能判断分离的效果。由于下游聚合和生命周期推断依赖于准确的功能性 UI 分组恢复，RQ1 将检验：对固定候选框进行受约束的视觉语言模型判断，是否优于无约束边界框生成和以往的 UI 分组方法。

5.4.1 实验设置。 我们使用 3,478 个手动标注的功能性 UI 组以及第 5.3 节所述的匹配协议评估 RQ1。我们将 DesignEvo（GPT-5.5）与三种基线方法和一种模型敏感性变体进行比较。**无约束视觉语言模型定位（GPT-5.5）**移除候选框提取步骤，并要求同一个视觉语言模型直接定位功能性 UI 组。**GUI感知分组**[63] 代表一种受心理学启发、基于规则的视觉分组方法；而 **GUIGroup**[69] 代表一种基于多模态大语言模型的功能分组基线方法。**DesignEvo（Qwen2.5-VL）**保留 DesignEvo 的流水线、提示词、候选框和评分脚本，仅替换模型。

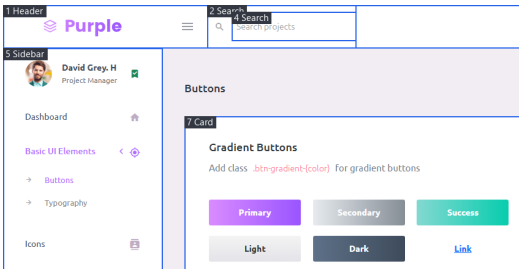
5.4.2 结果。 表 3 显示，DesignEvo（GPT-5.5）实现了最佳的功能性 UI 组恢复准确率，F1 值为 0.8510，平均交并比为 0.9791。无约束视觉语言模型定位（GPT-5.5）要求同一视觉语言模型直接绘制组框，其 F1 值降至 0.4223，召回率降至 0.3470，表明开放式定位在此任务中并不可靠。GUI感知分组和 GUIGroup 的表现优于无约束定位，但仍低于 DesignEvo，F1 分数分别为 0.6683 和 0.7664。DesignEvo（Qwen2.5-VL）的 F1 值达到 0.7362：固定候选框保持了定位的一致性，但较弱的合并和移除决策使精确率降至 0.6866。DesignEvo 的 95%置信区间紧密围绕点估计：精确率 0。8538 [0.8408，0。8647] ，召回率 0。8482 [0.8349，0。8589]。与最强基线 GUIGroup 相比，配对改进在精确率（[0.0351，0。0579]）和召回率（[0.0983，0。1462]）上均保持为正。图 2 中的定性案例解释了主要的不同机制。GUI感知分组和 GUIGroup 可能会将多个重复的统计卡片合并为一个大型连通区域，而 DesignEvo 将这些卡片保留为独立的功能性 UI 组，并将其分配至同一个卡片变体。模型敏感性示例进一步表明，固定候选集合并不会移除



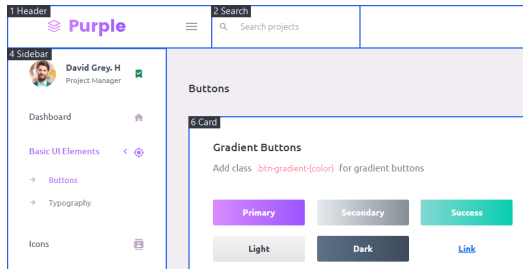
(a) GUI 分组合并重复卡片。



(b) DesignEvo 保留独立的卡片变体组。



(c) Qwen2.5-VL 未合并一个搜索组。



(d) GPT-5.5 合并了相关候选项。

图2. RQ1 的精简定性示例。第一行比较功能性 UI 分组恢复，第二行展示候选项合并中的模型敏感性。

表3. 功能性 UI 分组恢复的准确率。

方法	精确率	Rec.	F1	mIoU
GUI感知分组 [63]	0.7234	0.6210	0.6683	0.9365
GUIGroup [69]	0.8099	0.7274	0.7664	0.9451
无约束视觉语言模型定位（GPT-5.5）	0.5393	0.3470	0.4223	0.8656
DesignEvo（Qwen2.5-VL）	0.6866	0.7936	0.7362	0.9621
DesignEvo（GPT-5.5）	0.8538	0.8482	0.8510	0.9791

语义判断难度：Qwen2.5-VL 未合并一个搜索相关区域，而 GPT-5.5 正确地将相关候选项合并为一个功能性 UI 组。

5.5 RQ2：生命周期状态推断

RQ2 评估 DesignEvo 在相邻提交之间的生命周期推断能力。DesignEvo 不匹配孤立的 DOM 元素，而是比较恢复出的功能性 UI 组族。RQ2 测试这种代码仓库级表示是否能够改善生命周期状态标注，优于元素级匹配基线。

5.5.1 实验设置。 我们将 **DesignEvo（GPT-5.5）**与 **WebEvo**进行比较。WebEvo 是最接近的网页演化技术，可检测语义结构变化并映射相应元素 [48]。我们还纳入了 UI 测试修复中的三种对应关系方法：**WATER**使用基于属性的匹配 [12]，；**VISTA** 使用视觉匹配 [49]，；**SFTM** 执行灵活树匹配 [5]。此外，我们还纳入了基于同一项既有工作派生的两个基线 [32]：**EDIT DIS**使用 XPath 编辑距离匹配元素；**EDIT DIS+GPT-5.5**则进一步应用基于 GPT-5.5 的筛选和生命周期状态分类。最后，我们报告 **DesignEvo（Qwen2.5-VL）**作为模型敏感性变体，该变体使用相同的设置。

5.5.2 结果。 表格4 显示，DesignEvo (GPT-5.5) 的总体生命周期状态准确率最高，为 0.84；WebEvo 为 0.71，编辑距离为 0.62+GPT-5.5。最大的提升体现在需要设计系统级解释、而非局部元素

表格 4. 跨提交生命周期状态推断的准确率。DEvo-Qwen 使用 Qwen2.5-VL；EDIT DIS+GPT 和 DEvo GPT 使用 GPT-5.5。

状态	SFTM	VISTA	WATER	WebEvo	EDIT DIS	EDIT DIS+GPT	DEvo Qwen	DEvo GPT
复用	0.68	0.73	0.75	0.76	0.66	0.72	0.72	0.82
变更	0.62	0.71	0.65	0.72	0.54	0.53	0.72	0.83
新增	0.56	0.60	0.58	0.61	0.44	0.52	0.86	0.89
移除	0.21	0.29	0.24	0.36	0.17	0.26	0.77	0.88
歧义	0.00	0.33	0.00	0.50	0.00	0.00	0.00	0.50
总体	0.62	0.68	0.68	0.71	0.53	0.62	0.74	0.84

匹配的状态上。对于变更，DesignEvo 的准确率达到 0.83，而 WebEvo 为 0.72。对于新增和移除，DesignEvo 的准确率分别为 0.89 和 0.88，而 WebEvo 分别为 0.61 和 0.36。DesignEvo (Qwen2.5-VL) 的总体准确率为 0.7415，可作为模型敏感性的参考。这些结果表明，仓库级 UI 组族和共享样式值为区分真正的生命周期变更与视觉或 DOM 变化提供了有用依据。

5.6 RQ3：核心证据和约束的消融

RQ3 评估 DesignEvo 的核心证据来源和流水线约束所作出的贡献。由于功能性 UI 分组恢复、仓库级记录构建和生命周期状态推断依赖不同形式的几何、语义、视觉及样式证据，仅凭端到端准确率无法说明流水线的哪些部分不可或缺。因此，RQ3 测试移除每种主要证据来源或约束后，准确率会发生怎样的变化。

5.6.1 实验设置。 我们评估四组消融实验。**RQ3.1** 通过移除交互容器候选框、共享目标候选框或 DOM 容器候选框，测试功能性 UI 分组恢复中的候选框提取。**RQ3.2** 通过移除功能分类体系证据、边界框大小与位置证据或基于规则的验证门控，测试同版本 UI 组族的分配。**RQ3.3** 通过移除频率加权、近等价值分组或基于分类体系的命名，测试仓库共享样式值聚合。**RQ3.4** 通过在跨提交匹配中移除功能类别证据或仓库共享样式变更证据，测试生命周期推断。

5.6.2 结果。 **RQ3.1：候选框提取**。 表格 5显示，固定候选框对功能性 UI 分组恢复的贡献最大。允许视觉语言模型直接生成候选框，会使 F1 值从 0.8510 降至 0.4223，召回率从 0.8482 降至 0.3470。这证实了 RQ1 的核心机制：不受约束的视觉定位将坐标查找任务与判断功能含义的任务混为一谈。移除共享目标分组或共享交互容器分组，F1 值分别降至 0.5802 和 0.5827。这种相近的降幅表明，两项分组规则都能防止功能性 UI 组被恢复为碎片化区域。

RQ3.2：同版本 UI 组族分配。 表 6 表明，边界框大小与位置证据对抑制误报族分配的作用最大；移除该证据后，精确率从 0.9452 降至 0.8555。这是因为，视觉相似的卡片、筛选器和摘要面板可能具有相似的颜色或排版，但在不同页面区域中承担着不同角色；空间证据有助于避免将这些局部相似性误判为族关联。移除分类体系证据后，F1 值从 0.8536 降至 0.8406，表明功能类别有助于区分角色不同但视觉相似的组。移除基于规则的验证

表5. 候选框提取的消融实验结果。

变体	精确率	Rec.	F1	mIoU
DesignEvo	0.8538	0.8482	0.8510	0.9791
无约束视觉语言模型定位（GPT-5.5）	0.5393	0.3470	0.4223	0.8656
不采用共享目标分组	0.5918	0.5690	0.5802	0.9317
不采用共享交互容器分组	0.5894	0.5762	0.5827	0.9321

表6. 同版本UI组族分配的消融实验结果。

变体	精确率	Rec.	F1
DesignEvo	0.9452	0.7782	0.8536
不使用分类体系	0.9138	0.7782	0.8406
不使用边界框/尺寸/位置	0.8555	0.7782	0.8150
不使用基于规则的验证	0.9351	0.7782	0.8495

表7. 仓库共享样式值聚合的消融实验结果。

变体	精确率	Rec.	F1
DesignEvo	0.9750	1.0000	0.9873
不进行频率加权	0.4632	1.0000	0.6332
不进行近等价值分组	0.7835	0.9172	0.8451
不采用基于分类体系的命名	0.9889	0.9704	0.9796

产生了较小的 F1 值降幅，降至 0.8495。在这些变体中，召回率保持不变，而预测族的数量增加，这表明这些约束主要通过拒绝过于宽泛的族分配来提高精确率。

RQ3.3：仓库共享样式值聚合。表 7 显示，完整的三步聚合实现了 0.9873 的 F1 值。移除频率加权会导致最大的性能下降，使精确率降至 0.4632、F1 值降至 0.6332，因为许多偶发值或页面特定值不再被筛除。这是仓库级样式恢复不能简单地合并所有观测到的计算值的最有力证据：一次性宽度、局部边距、图表颜色以及内容特定尺寸都会被误报为共享的设计系统值。移除近等价值分组会使 F1 值降至 0.8451，表明等价惯例通常通过略有不同的字面值或变量来表达。移除基于分类体系的命名影响较小，使召回率降至 0.9704、F1 值降至 0.9796，这表明命名主要影响覆盖范围和可解释性。

RQ3.4：生命周期推断证据。表格8 表明，功能类别证据和样式变化证据都对生命周期推断有所贡献。移除样式变化证据后，总体 F1 值从 0.8328 降至 0.6935，对变更记录的影响最大，其 F1 值从 0.8119 降至 0.5769。没有样式变化证据时，模型仍能识别出两条记录扮演相同角色，但无法捕捉尺寸、层级、间距、颜色或组件构成的变化，而这些变化使记录成为有意义的变更案例。移除功能类别证据后，总体 F1 值降至 0.7676，且变更识别能力也有所减弱。在此变体中，具有不同面向用户的角色但视觉上相似的记录更容易被匹配，而外观发生变更的功能连续记录

表8。 生命周期状态推断的消融实验结果。

变体	精确率	Rec.	F1
DesignEvo	0.8356	0.8301	0.8328
没有样式变化证据	0.6958	0.6912	0.6935
没有功能类别证据	0.7702	0.7651	0.7676

则更容易被拆分。这些结果支持 DesignEvo 的跨工件匹配设计：生命周期推断需要功能连续性和样式值变化。完整的生命周期推断估计也较为稳定：在 95% 的置信水平下，精确率为 0.8356 [0.8224，0.8485]，召回率为 0.8301 [0.8167，0.8430]。

6 讨论

6.1 设计系统恢复的启示

结果表明，将几何定位与语义解释分开，有助于设计系统恢复。在固定 GPT-5.5 的情况下，候选约束恢复的 F1 值为 0.8510，而直接定位为 0.4223（表 3）。通过以运行时派生区域为依据，DesignEvo 使视觉语言模型能够专注于功能一致性，而非坐标生成。然而，恢复仍取决于候选覆盖率和模型的语义判断，这一点也体现在模型敏感性结果中。

仓库级恢复还需要在不同版本间进行选择性聚合并利用互补证据。RQ3 表明，空间上下文有助于避免错误的族分配，而频率加权能够区分反复出现的样式规范与偶发值。在生命周期推断中，功能证据有助于确定记录之间的对应关系，而构成和样式变化则有助于区分复用与变更（表 8）。结合 RQ2 中相较于元素级匹配基线的改进，这些发现支持在单个 UI 元素之外恢复和比较设计系统记录。尽管如此，最终得到的族和共享样式反映的是观察到的实现模式，并不一定代表预期的设计规则。

6.2 对仓库级变更审查的启示

DesignEvo 提供了渲染后的 UI 区域、仓库级族、共享样式及其生命周期状态之间可追溯的关系。在变更审查期间，变更记录可将注意力引向对应的匹配项及其结构或风格差异，而相关族成员则有助于了解其他位置是否也出现了类似变更。同样，新增和移除标识的是已分析清单中的未匹配记录，并不一定意味着整个应用程序范围内发生了添加或删除。这些关系为检查 UI 演化提供了结构化依据，而不局限于孤立的屏幕截图或源代码差异。

然而，观察到的差异并不自动构成设计系统违规。反复出现的实现模式可作为经验参考，而符合性判断则需要适用的设计规范以及存在违规的证据。有意变体和修订可能与既有模式存在合理差异。尽管实验评估了恢复效果和生命周期推断准确率，但并未证明这些记录能否改善人工审查。未来研究应比较使用和不使用 DesignEvo 进行审查的效果，并衡量受影响模式识别、错误不一致报告和审查工作量。

6.3 对效度的威胁

构念效度。 功能性 UI 组是对粒度的一种操作性选择，其依据是组件目录和第 4.2 节中的分类体系。它们无法涵盖每一种控件状态、导航流程或行为契约。族群边界的划定也涉及判断：外观相似可能掩盖不同用途，而同一用途也可能有多种有效实现。共享样式记录带来了进一步的近似性，因为对相似计算值进行分组，可能会将有意区分的样式合并在一起，却无法还原其底层的设计令牌关系。响应式布局和未观察到的交互状态也可能未纳入恢复后的表示中。出现位置引用有助于检查这些解释，但无法确定预期的设计规则、设计质量或完整性。

内部效度。 视觉语言模型（VLM）的决策可能因模型版本、提示词措辞和图像质量而异。固定的候选区域、结构化输出、确定性推理设置以及基于规则的检查可限制这种变异，同时模型敏感性分析会考察其影响；但这些措施都无法保证语义判断完全一致。错误还可能发生传播：遗漏的组会影响族分配，而不正确的族链接可能改变生命周期推断。渲染条件或所观察到的路由和状态存在差异，也可能产生表面上的新增或移除现象。

标注效度与结论效度。 标注需要对功能边界和有意义的变化作出判断。独立审查、部分重新标注和仲裁能够减少分歧，但无法证明所有合理的分解方式都已得到表示。生命周期标签分布不均衡：复用占标签的 71.01%，而歧义仅占 0.12%。逐状态结果补充了总体准确率，但歧义案例的稀少限制了关于不确定性处理的结论。同一页面的重复版本也是相互关联的观测，不应被视为独立应用程序。

外部效度。 该基准测试排除了无法构建或渲染的版本，并聚焦于存在可观察 UI 变化的路由。因此，结果反映的是所选且可渲染的历史记录，而非所有前端提交。组件复用较少、个性化程度较高、采用基于画布的渲染，或具有不同响应式状态和交互状态的应用程序，可能需要额外证据。维护同步设计系统清单的项目从恢复工作中获益也可能较少。还需在这些场景及下游审查任务中开展进一步评估，以确定其更广泛的适用性。

7 结论

本文介绍了 DesignEvo，这是一种用于在前端代码仓库及其版本中恢复和追踪已实现设计系统记录的方法。DesignEvo 将程序派生的候选区域与基于视觉语言模型的功能判断相结合，从而恢复功能性 UI 组，将重复出现的模式和共享样式汇总为仓库级记录，并推断其在各次提交中的生命周期状态。在 10 个开源代码仓库上进行的评估表明，DesignEvo 在组恢复任务上的 F1 分数为 0.85，总体生命周期状态准确率为 0.84，而表现最强的生命周期基线为 0.71。消融实验结果进一步表明，候选约束、功能类别和样式证据均发挥了作用。生成的记录为界面模式的复用和演变方式提供了结构化、可追溯的证据，可支持下游 UI 变更审计、设计系统漂移分析和前端维护。

参考文献

[1] Abdulaziz Alshayban 和 Sam Malek。2022年。Accessitext：在 Android 应用中自动检测文本无障碍问题。载于第*30*届 *ACM* 欧洲软件工程联合会议与软件工程基础研讨会论文集。984–995。[2] 蚂蚁集团。无日期。Ant Design。https://ant.design/。访问日期：2026年6月22日。

[3] Anthropic。2026。Claude Design。https://claude.ai/design。研究预览版。访问日期：2026-07-01。[4] Bolt.new。2025。Bolt.new：用于构建网站、Web 应用和移动应用的 AI 工具。https://bolt.new/ 一款由 AI 驱动的工具，可在几分钟内将自然语言提示转换为网站、Web 应用或移动应用。[5] Sacha Brisset、Romain Rouvoy、Lionel Seinturier 和 Renaud Pawlak。2023。SFTM：使用基于相似度的灵活树匹配快速匹配网页。Information Systems 112 (2023)，102126。[6] Jieshan Chen、Mulong Xie、Zhenchang Xing、Chunyang Chen、Xiwei Xu、Liming Zhu 和 Guoqiang Li。2020。图形用户界面的目标检测：传统方法、深度学习还是二者结合？载于第28届 ACM 欧洲软件工程大会与软件工程基础研讨会联合会议论文集。1202–1214。[7] Liuqing Chen、Yunnong Chen、Shuhong Xiao、Yaxuan Song、Lingyun Sun、Yankun Zhen、Tingting Zhou 和 Yanfang Chang。2024。Egfe：利用多模态学习对 UI 设计中的碎片化元素进行端到端分组。载于第46届 IEEE/ACM 国际软件工程大会论文集。1–12。[8] Sen Chen、Lingling Fan、Chunyang Chen 和 Yang Liu。2022。自动提取具有丰富特征的 Android 应用故事板。IEEE Transactions on Software Engineering 49，2 (2022)，667–683。[9] Yunnong Chen、Yankun Zhen、Chuning Shi、Jiazhi Li、Liuqing Chen、Zejian Li、Lingyun Sun、Tingting Zhou 和 Yanfang Chang。2023。UI 图层合并器：通过视觉学习和边界先验合并 UI 图层。Frontiers of Information Technology & Electronic Engineering 24，3 (2023)，373–387。[10] Elliot J. Chikofsky 和 James H Cross。1990。逆向工程与设计恢复：分类体系。IEEE software 7，1 (1990)，13–17。[11] Paul T Chiou、Robert Winn、Ali S Alotaibi 和 William GJ Halfond。2024。自动检测响应式网页中的重排可访问性问题。载于 IEEE/ACM 第46届国际软件工程大会论文集。1–13。[12] Shauvik Roy Choudhary、Dan Zhao、Husayn Versee 和 Alessandro Orso。2011。Water：Web 应用测试修复。载于首届端到端测试脚本工程国际研讨会论文集。24–29。[13] Design Tokens Community Group。2025。Design Tokens Format Module 2025.10。https://www.designtokens.org/tr/ 2025.10/format/。社区组最终报告，2025年10月28日。访问日期：2026年11月14日。[14] designsystems.surf。无日期。设计系统库。https://designsystems.surf/design-systems。访问日期：2026年6月22日。[15] Peitong Duan、Chin-Yi Cheng、Gang Li、Bjoern Hartmann 和 Yang Li。2024。UICrit：借助 UI 评析数据集提升自动化设计评估。载于第37届 ACM 用户界面软件与技术年会论文集。1–17。[16] Peitong Duan、Jeremy Warner、Yang Li 和 Bjoern Hartmann。2024。利用大语言模型为 UI 模型图生成自动反馈。载于2024年 CHI计算机系统中的人为因素大会论文集。1–20。[17] Fabio Ferreira、Hudson Silva Borges 和 Marco Tulio Valente。2024。重构基于 React 的 Web 应用。Journal of Systems and Software 215 (2024)，112105。[18] Brad Frost。无日期。原子设计方法论。https://atomicdesign.bradfrost.com/。访问日期：2026年6月22日。[19] 谷歌。2026。通过 Stitch 介绍“氛围设计”。https://blog.google/innovation-and-ai/models-and-research/google-labs/stitch-ai-ui-design/ [20] 谷歌。无日期。Material Design。https://m3.material.io/。访问日期：2026年6月22日。[21] Matthew Hague、Anthony W Lin 和 C-H Luke Ong。2015。在 HTML5 应用中检测冗余 CSS 规则：一种树重写方法。载于2015年 ACM SIGPLAN 面向对象编程、系统、语言与应用国际会议论文集。1–19。[22] Mouna Hammoudi、Gregg Rothermel 和 Andrea Stocco。2016。Waterfall：一种用于修复 Web 应用录制回放测试的增量方法。载于2016年第24届 ACM SIGSOFT 软件工程基础国际研讨会论文集。751–762。[23] Elliotte Rusty Harold。2012年。重构 HTML：改进现有 Web 应用的设计。Addison-Wesley Professional。[24] Mohammed Tayeeb Hasan、Nikolaos Tsantalis 和 Pouria Alikhanifard。2024。感知重构的代码块跟踪与提交历史。IEEE Transactions on Software Engineering 50，12 (2024)，3330–3350。[25] Joshua Hibschman 和 Haoqi Zhang。2015。Unravel：通过交互录制、源代码追踪和库检测快速逆向工程 Web 应用。载于第28届 ACM 用户界面软件与技术年会论文集。270–279。[26] Kai Huang、Jian Zhang、Xiaofei Xie 和 Chunyang Chen。2025。看见即修复：利用多模态大语言模型进行视觉软件问题修复的跨模态推理。载于2025年第40届 IEEE/ACM 自动化软件工程国际会议（ASE）。IEEE，1156–1168。[27] IBM。2026。按钮：使用指南。https://www.carbondesignsystem.com/building-blocks/core/components/button/guidelines。访问日期：2026年11月14日。[28] IBM。无日期。Carbon 设计系统。https://carbondesignsystem.com/。访问日期：2026年6月22日。

Mingyue Yuan、Jinbo Yang、JieshanChen、Aorui Yang、Aaron Quigley、Gelareh Mohammadi、Zhenchang Xing、Boxi Wu、20 和 Dehai Zhao

[29] Jan Keim、Sophie Corallo、Dominik Fuchß、Tobias Hey、Tobias Telge 和 Anne Koziolek。2024年。恢复软件文档与代码之间的追踪链接。载于《IEEE/ACM第46届国际软件工程会议论文集》。1–13。[30] Arun Krishna Vajjala、SM Hasan Mansur、Justin Jose 和 Kevin Moran。2024年。MotorEase：自动检测移动应用用户界面中的运动障碍无障碍问题。载于《IEEE/ACM第46届国际软件工程会议论文集》。1–13。[31] Jiazhi Li、Tingting Zhou、Yunnong Chen、Yanfang Chang、Yankun Zhen、Lingyun Sun 和 Liuqing Chen。2022年。ULDGNN：一种基于图神经网络的碎片化用户界面层检测器。arXiv预印本 arXiv:2208.06658（2022）。arXiv:2208.06658 [cs.CV][32] Yuanzhang Lin、Guoyao Wen 和 Xiang Gao。2023年。通过迭代式元素匹配自动修复网页用户界面测试。载于《2023年第38届IEEE/ACM国际自动化软件工程会议（ASE）》。IEEE，1188–1199。[33] Bo Liu、Hui Liu、Nan Niu、Yuxia Zhang、Guangjie Li 和 Yanjie Jiang。2023年。连续版本之间的自动化软件实体匹配。载于《2023年第38届IEEE/ACM国际自动化软件工程会议（ASE）》。IEEE，1615–1627。[34] Qinyu Luo、Yining Ye、Shihao Liang、Zhong Zhang、Yujia Qin、Yaxi Lu、Yesai Wu、Xin Cong、Yankai Lin、Yingli Zhang 等。2024年。Repoagent：由大语言模型驱动的仓库级代码文档生成开源框架。载于《2024年自然语言处理实证方法会议：系统演示论文集》。436–464。[35] Davood Mazinanian、Nikolaos Tsantalis 和 Ali Mesbah。2014年。发现层叠样式表中的重构机会。载于《第22届ACM SIGSOFT软件工程基础国际研讨会论文集》。496–506。[36] Ali Mesbah 和 Shabnam Mirshokraie。2012年。2012年 2012 34 408 418。用于支持样式维护的CSS规则自动分析。载于《第34届国际软件工程会议（ICSE）》。IEEE，–。[37] 微软。未知年份。Fluent 2设计系统。https://fluent2.microsoft.design/。访问日期：2026年6月22日。[38] Kevin Moran、Cody Watson、John Hoskins、George Purnell 和 Denys Poshyvanyk。2018年。检测并总结演进中的移动应用的GUI变更。载于《第33届ACM/IEEE国际自动化软件工程会议论文集》。543–553。[39] Alex Nederlof、Ali Mesbah 和 Arie van Deursen。2014年。Web软件工程：实践现状。载于《第36届国际软件工程大会伴随论文集》。4–13。[40] Tuan Anh Nguyen 和 Christoph Csallner。2015年。使用remaui（论文）对移动应用用户界面进行逆向工程。载于《2015年第30届IEEE/ACM国际自动化软件工程会议（ASE）》。IEEE，248–259。[41] OverlayQA。未知年份。什么是设计系统漂移，以及如何检测它。https://overlayqa.com/blog/design-system-drift/。访问日期：2026年6月29日。[42] Kunal Pai、Premkumar Devanbu 和 Toufique Ahmed。2025年。CoDocBench：软件维护中的代码与文档对齐数据集。载于《2025年IEEE/ACM第22届国际软件仓库挖掘会议（MSR）》。IEEE，451–455。[43] Pattern Lab。未知年份。使用Pattern Lab创建原子化设计系统。https://patternlab.io/。访问日期：2026年6月22日。[44] Leonard Punt、Sjoerd Visscher 和 Vadim Zaytsev。2016年。A?B*A模式：撤销CSS中的样式及其带来的重构机会。载于《2016年IEEE国际软件维护与演化会议（ICSME）》。IEEE，67–77。[45] Daniel Ramos、Hailie Mitchell、Inês Lynce、Vasco Manquinho、Ruben Martins 和 Claire Le Goues。2023年。MELT：从拉取请求中挖掘有效的轻量级转换。载于《2023年第38届IEEE/ACM国际自动化软件工程会议（ASE）》。IEEE，1516–1528。[46] React。未知年份。向组件传递Props。https://react.dev/learn/passing-props-to-a-component。访问日期：2026年11月14日。[47] Sabiha Salma、SM Hasan Mansur、Yule Zhang 和 Kevin Moran。2024年。GuiEvo：移动应用用户界面的自动化演化。载于《第21届国际软件仓库挖掘会议论文集》。335–347。[48] Fei Shao、Rui Xu、Wasif Haque、Jingwei Xu、Ying Zhang、Wei Yang、Yanfang Ye 和 Xusheng Xiao。2021年。WebEvo：通过检测语义结构变更驾驭Web应用演化。载于《第30届ACM SIGSOFT国际软件测试与分析研讨会论文集》。16–28。[49] Andrea Stocco、Rahulkrishna Yandrapally 和 Ali Mesbah。2018年。可视化网页测试修复。载于《2018年第26届ACM欧洲软件工程大会与软件工程基础研讨会联合会议论文集》。503–514。[50] Yuhui Su、Chunyang Chen、Junjie Wang、Zhe Liu、Dandan Wang、Shoubin Li 和 Qing Wang。2022年。蜕变：自动检测移动应用的缩放问题。载于《第37届IEEE/ACM国际自动化软件工程会议论文集》。1–12。

[51] Yuhui Su、Zhe Liu、Chunyang Chen、Junjie Wang 和 Qing Wang。2021年。OwlEyes-online：用于检测和定位用户界面显示问题的全自动平台。载于《第29届ACM欧洲软件工程大会与软件工程基础研讨会联合会议论文集》。1500–1504。[52] Hilal Taha、Jeongju Sohn 和 Michail Papadakis。2026年。MiningWeb Evolution：跨网站DOM变更数据集。载于《ACM Web 大会2026伴随论文集》。1083–1088。[53] Wen Siang Tan、Markus Wagner 和 Christoph Treude。2024年。检测软件仓库文档中过时的代码元素引用。《实证软件工程》29，1（2024），5。[54] Nikolaos Tsantalis、Matin Mansouri、Laleh M Eshkevari、Davood Mazinanian 和 Danny Dig。2018年。提交历史中准确且高效的重构检测。载于《第40届国际软件工程会议论文集》。483–494。[55] Vercel / v0。2025年。v0.app：用于设计、迭代和扩展全栈应用程序的协作式人工智能助手。https://v0.app/ 一个支持通过提示驱动全栈Web应用程序设计、迭代和扩展的协作式人工智能助手。[56] W3C。2024年。《网页内容无障碍指南》（WCAG）2.2。https://www.w3.org/TR/WCAG22/。W3C建议。[57] Yuxuan Wan、Chaozheng Wang、Yi Dong、Wenxuan Wang、Shuqing Li、Yintong Gold Huo 和 Michael R. Lyu。2025年。分而治之：从屏幕截图生成用户界面代码。《ACM软件工程论文集》2，FSE，文章171（2025），2399–2422页。[58] Hao Wen、Yuanchun Li、Guohong Liu、Shanhui Zhao、Tao Yu、Toby Jia-Jun Li、Shiqi Jiang、Yunhao Liu、Yaqin Zhang 和 Yunxin Liu。2024年。Autodroid：Android中的大语言模型驱动任务自动化。载于《第30届移动计算与网络年会论文集》。543–557。[59] Fan Wu、Cuiyun Gao、Shuqing Li、Xin-Cheng Wen 和 Qing Liao。2025年。基于多模态大语言模型的UI2Code自动化：由用户界面布局信息引导。《ACM软件工程论文集》2，ISSTA（2025），1123–1145。[60] Jason Wu、Yi-Hao Peng、Xin Yue Amanda Li、Amanda Swearngin、Jeffrey P Bigham 和 Jeffrey Nichols。2024年。UIClip：用于评估用户界面设计的数据驱动模型。载于《第37届ACM用户界面软件与技术年会论文集》。1–16。[61] Shuhong Xiao、Yunnong Chen、Yaxuan Song、Liuqing Chen、Lingyun Sun、Yankun Zhen、Yanfang Chang 和 Tingting Zhou。2024年。用户界面语义组件组检测：在移动图形用户界面中对具有相似语义的用户界面元素进行分组。《Displays》83（2024），102679。[62] Mulong Xie、Sidong Feng、Zhenchang Xing、Jieshan Chen 和 Chunyang Chen。2020年。UIED：GUI元素检测混合工具。载于《第28届ACM欧洲软件工程大会与软件工程基础研讨会联合会议论文集》。1655–1659。[63] Mulong Xie、Zhenchang Xing、Sidong Feng、Xiwei Xu、Liming Zhu 和 Chunyang Chen。2022年。受心理学启发的无监督GUI图像GUI控件感知组推断。载于《第30届ACM欧洲软件工程大会与软件工程基础研讨会联合会议论文集》。332–343。[64] Zhuolin Xu、Qiushi Li 和 Shin Hwei Tant。2025年。理解并增强使用大语言模型修复网页用户界面测试时的属性优先级排序。载于《2025年IEEE软件测试、验证与确认大会（ICST）》。IEEE，326–337。[65] Bo Yang、Zhenchang Xing、Xin Xia、Chunyang Chen、Deheng Ye 和 Shanping Li。2021年。别这样做！依据设计指南追查复杂用户界面中的视觉设计异味。载于《2021年IEEE/ACM第43届国际软件工程会议（ICSE）》。IEEE，761–772。[66] Mingyue Yuan、Jieshan Chen、Zhenchang Xing、Aaron Quigley、Yuyu Luo、Tianqi Luo、Gelareh Mohammadi、Qinghua Lu 和 Liming Zhu。2025年。DesignRepair：利用大语言模型进行双流设计指南感知型前端修复。载于《2025年IEEE/ACM第47届国际软件工程会议（ICSE）》。IEEE，2483–2494。[67] zeroheight。未知年份。设计系统报告2026。https://report.zeroheight.com/。访问日期：2026年6月29日。[68] Yiran Zhang、Zhengzi Xu、Chengwei Liu、Hongxu Chen、Jianwen Sun、Dong Qiu 和 Yang Liu。2023年。通过信息融合进行软件架构恢复。载于《第31届ACM欧洲软件工程大会与软件工程基础研讨会联合会议论文集》。1535–1547。[69] Yulei Zhang、Yifan Yang、Zejun Zhang、Xiaoqian Li、Kui Liu、Zhenchang Xing、Xin Xia 和 Lingfeng Bao。2026年。GUIGroup：利用多模态大语言模型实现功能布局分组。《IEEE Transactions on Software Engineering》（2026）。[70] Dehai Zhao、Zhenchang Xing、Chunyang Chen、Xiwei Xu、Liming Zhu、Guoqiang Li 和 Jinshui Wang。2020年。Seenomaly：依据设计禁忌指南进行GUI动画效果的基于视觉的检查。载于《ACM/IEEE第42届国际软件工程会议论文集》。1286–1297。[71] Wenting Zhao、Wuxia Jin、Yiran Zhang、Ming Fan、Haijun Wang、Li Li、Yang Liu 和 Ting Liu。2025年。语义增强的软件架构恢复。《IEEE Transactions on Software Engineering》（2025）。

Mingyue Yuan、Jinbo Yang、JieshanChen、Aorui Yang、Aaron Quigley、Gelareh Mohammadi、Zhenchang Xing、Boxi Wu、22 和 Dehai Zhao

A 提示模板

提示 A：功能性 UI 组恢复{ "system_message": "你是一名功能性用户界面组标注器。仅返回严格的 JSON。你只能选择现有的候选 _id。不要编造或编辑边界框坐标。", "user_prompt": { "task": " 从程序生成的候选框中选择并分类功能性 UI 组。", "goal": "识别网页上可见且功能上连贯的 UI 组。模型应决定哪些候选项应予以保留、移除、合并，并为其分配最合适的分类标签。", "page_context": { "repo_id": "{repo_id}", "page_id": "{page_id}", "commit": "{commit_hash}", "image_size": { "width": "{image_width}", "height": "{image_height}" } }, "important_principles": ["以功能角色和面向用户的用途作为判断的主要依据。", "将视觉分组、布局边界、文本内容、DOM 源、交互目标和候选证据作为辅助信息。", "仅输出所提供候选列表中的候选_id。", "不要输出新的边界框坐标。最终边界框将由程序使用所选候选框固定。", "如果多个候选项共同表示一个功能性 UI 组，请在一个输出组中列出所有对应的候选_id，以将它们合并。", "如果某个候选项只是装饰性片段、普通的重复父子框，或较大连贯块中的不完整部分，请将其省略。", "当具有独立功能时，保留有意义的第二层功能性 UI 组，例如搜索、面包屑导航、表单、数据表、可视化、卡片或操作栏。", "当一个大区域包含多个独立的功能块时，优先保留这些独立块，而不是只选择大的容器。", "当大区域本身具有连贯的功能角色时，例如完整侧边栏、完整页眉、完整表单、完整表格或完整卡片面板，将其作为一个功能性 UI 组保留。", "从所提供的分类定义中选择最接近的分类标签。不要创建新的分类标签。"]，"preprocessing_rules_used_to_generate_candidates": ["候选项可能通过对共享同一推断交互式父级的可见元素进行分组来生成。", "候选项可能通过对导航至同一目标超链接的可见元素进行分组来生成。", "候选项也可能根据具有明确功能角色的可见 DOM 容器生成，例如卡片、表格、表单、图表、列表、面包屑导航、搜索区域、页眉、侧边栏和页脚。"],"taxonomy_definitions": [

，第1卷，第1期，文章。发布日期：2026年11月。

{ "taxonomy_l1": "全局元素", "taxonomy_l2": "页眉", "definition": "页面级或站点级顶部全局区域，包括导航栏、应用栏、顶部栏、徽标区域、账户条目或全局操作。" }, { "taxonomy_l1": "全局元素", "taxonomy_l2": "页脚", "definition": "页面级或站点级底部全局区域，通常包含辅助导航、版权信息、链接组或支持信息。" }, { "taxonomy_l1": "导航", "taxonomy_l2": "面包屑导航", "definition": "以路径形式呈现的导航块，用于显示当前页面的层级或位置，通常由多个层级条目组成。" }, { "taxonomy_l1": "导航", "taxonomy_l2": "搜索", "definition": "用于查询、筛选或定位页面内容的搜索模块或搜索区域。它可能包含输入字段、图标、按钮或筛选控件。" }, { "taxonomy_l1": "导航", "taxonomy_l2": "侧边栏", "definition": "侧边导航或工具区域，通常包含多个导航条目、菜单组、账户条目、设置、通知或与状态相关的快捷方式。" }, { "taxonomy_l1": "操作", "taxonomy_l2": "操作栏", "definition": "为同一页面、分区、项目或选定集合组织操作的命令区域，例如编辑、删除、分享、筛选、排序、保存、导出或批量操作。" }, { "taxonomy_l1": "输入", "taxonomy_l2": "输入", "definition": "以字段为中心的输入块，将一个输入目标与支持性控件组合在一起。

g
建议、选项或筛选行为，例如文本字段、文本区域、自动完成 te
字段、组合框输入或筛选输入框。” }, { "taxonomy_l1": "输入", "taxonomy_l2": "选择器/下拉选择器", "definition": "用于选择值、视图或模式的选项组选择块

,
日期或状态。” },

Mingyue Yuan、Jinbo Yang、JieshanChen、Aorui Yang、Aaron Quigley、Gelareh Mohammadi、Zhenchang Xing、Boxi Wu、24 和 Dehai Zhao

{ "taxonomy_l1": "输入", "taxonomy_l2": "表单", "definition": "面向任务的输入区域，用于将多个字段、标签、辅助文本、验证消息和操作组合在一起，以供提交、筛选或配置。" }, { "taxonomy_l1": "数据展示", "taxonomy_l2": "卡片", "definition": "一个有边界的内容容器，将相关的标题、文本、媒体、数据、操作或状态信息组合成一个连贯的视觉单元。" }, { "taxonomy_l1": "数据展示", "taxonomy_l2": "统计面板", "definition": "一个摘要区域，用于组合多个关键值或指标，以便快速比较或监控。" }, { "taxonomy_l1": "数据展示", "taxonomy_l2": "可视化", "definition": "一个数据呈现区块，用于组合视觉标记、标签、图例或控件，以帮助理解值或模式，例如图表、图形或地图。" }, { "taxonomy_l1": "数据展示", "taxonomy_l2": "时间线", "definition": "按时间顺序组织事件、活动、阶段、更新或日志的区块。" }, { "taxonomy_l1": "数据展示", "taxonomy_l2": "网格", "definition": "一种基于网格的内容展示区块，其中多个重复的内容单元按二维结构排列，例如卡片网格、指标网格或图像网格。" }, { "taxonomy_l1": "数据展示", "taxonomy_l2": "代码片段", "definition": "代码展示区块，通常包含代码内容、语法高亮、复制操作、标签或语言指示器。" }, { "taxonomy_l1": "数据展示", "taxonomy_l2": " 数据表 ", "definition": " 一种结构化数据展示区块，包含行、列、表头、单元格、状态、操作或分页表格控件。" }, { "taxonomy_l1": "数据展示",

，第1卷，第1期，文章。发布日期：2026年11月。

"taxonomy_l2": "列表", "definition": "由多个列表项、菜单项、操作项、记录或重复文本条目组成的列表样式展示块。当侧边栏或页眉更为合适时，页面级导航不应归类为列表。" }, { "taxonomy_l1": "数据展示", "taxonomy_l2": "主视觉区", "definition": "主要视觉区域或介绍性区块，通常包含标题、描述、图像、行动号召或重要宣传内容。" }, { "taxonomy_l1": "数据展示", "taxonomy_l2": "内容块", "definition": "通用的内容强调或摘要区块，通常由标题、文本、图像、强调框、描述或辅助操作组成；当没有更合适的具体分类标签时使用。" }, { "taxonomy_l1": "状态", "taxonomy_l2": "消息", "definition": "将状态消息与相关上下文、严重程度或关闭控件组合在一起的反馈区域，包括警示横幅、内联消息、提示条、提示框、警告或成功消息。" }, { "taxonomy_l1": "状态", "taxonomy_l2": "空状态", "definition": "当某个区域没有内容、结果、数据或可用项目时显示的区块，通常包含解释性文本或操作建议。" }, { "taxonomy_l1": "状态", "taxonomy_l2": "加载器", "definition": "用于指示内容待加载或操作正在进行的加载反馈区域，例如加载旋转指示器或骨架屏加载器。" }, { "taxonomy_l1": "状态", "taxonomy_l2": "进度", "definition": "显示任务、流程或序列完成状态的进度反馈区域。" }, { "taxonomy_l1": "叠加层", "taxonomy_l2": "对话框/模态框", "definition": "位于当前页面之上、用于组合临时内容及相关操作的叠加区域，例如对话框、模态框、底部弹层或弹出窗口。" }, { "taxonomy_l1": "图像类型",

Mingyue Yuan、Jinbo Yang、JieshanChen、Aorui Yang、Aaron Quigley、Gelareh Mohammadi、Zhenchang Xing、Boxi Wu、26 和 Dehai Zhao

"taxonomy_l2": "徽标/广告区块", "definition": "以图像为基础的身份标识或宣传区块，将品牌标识、活动内容、赞助标签、链接或操作作为一个可识别的视觉区域呈现。" }, { "taxonomy_l1": "第三方嵌入内容", "taxonomy_l2": "嵌入式小部件", "definition": "外部嵌入或第三方模块，例如内嵌框架、小部件、滚动信息栏、社交组件或由外部托管的交互式组件。" }],"candidate_input_schema": { "candidate_id": "程序生成的候选框的唯一 ID。", "source": "候选框的生成方式，例如 DOM 容器、共享交互式父级、相同的 href 目标或功能容器检测。", "bbox": "程序生成的候选框。模型不得编辑该候选框。", "text": "从候选区域提取的可见文本。", "href": "可用时提供的导航目标。", "suggested_taxonomy": "程序提供的弱建议，仅用作辅助证据。" }, "output_schema": { "groups": [{ "label": "功能性 UI 组的简短、便于人类阅读的英文名称。", "taxonomy_l1": "来自 taxonomy_definitions 的一个一级分类名称。", "taxonomy_l2": "来自 taxonomy_definitions 的一个二级分类标签。", "candidate_ids": ["选择或合并到此组中的候选 ID"],"confidence": "高 | 中 | 低", "reason": "简要说明所选候选项为何构成功能性 UI 组。" }], "candidates": "{candidate_list}" } }

提示 B：UI 组族分配

{ "stage": "UI_组_族_分配", "prompt_模板": { "task": "判断来自同一网页的两个功能性 UI 组是否应属于同一同版本 UI 组族的变体。", "page_key": "...", "消融_变体": "完整", "决策_政策": [

，第1卷，第1期，文章。发布日期：2026年11月。

"仅当两个组具有相同的主要面向用户的用途，并且看起来是同一 UI 组族的重复变体时，才返回 same_family=true。", "如果功能类别或面向用户的用途不同，则即使样式和尺寸相似，same_family 也必须为 false。", "如果边界框的尺寸不同，same_family 应为 false。", "如果样式明显不同，same_family 应为 false，除非它是同一重复 UI 组族的有意视觉变体。", " 同一组族的示例：同一仪表板行中的 KPI 卡片、定价方案卡片、重复的警报面板、重复的个人资料信息卡片。", "不同组族的示例：页眉与侧边栏、主视觉区与卡片、可视化卡片与 KPI 卡片、表单/控件与数据卡片。", "如果某个组被错误标记为表单，但其理由说明它是 KPI/指标卡片，则它可以属于视觉上匹配的卡片组族。", "仅返回 JSON。"],"output_shape": { "pairs": [{ "pair_id": "p001", "same_family": true, "reason": "两者都是尺寸/样式相容的重复 KPI 摘要卡片" }]}, "pairs": ["... 候选组对 ..."]} }

提示 C：生命周期状态检测

{ "stage": "生命周期_推断", "prompt_模板": { "task": "推断相邻提交中功能性 UI 组族记录的生命周期状态。", "page": "...", "states": ["复用", "变更", "新增", "移除", "歧义"],"rules": ["按角色/功能和分类体系匹配", "复用=同一仓库级记录，且视觉、结构或样式值没有实质性变更。", "变更=同一仓库级记录，但布局、尺寸、样式值、内容排列、UI 元素组成或惯例级变更发生了实质性变化。",

Mingyue Yuan、Jinbo Yang、JieshanChen、Aorui Yang、Aaron Quigley、Gelareh Mohammadi、Zhenchang Xing、Boxi Wu、28 和 Dehai Zhao

```
"新增=当前记录没有先前                    前一个对应项。”,
“移除=前一条记录没有当前                  对应项。”,
“歧义=嘈杂、不可靠或未获充                分支持的对应关系。”,
"推理中不设固定数值阈值                  。仅输出严格的 JSON。"
]," prev": ["... 先前提交记录 ..."]," cur": ["...
当前提交记录 ..."]," pairs": ["... 候选跨提交
匹配。
. . "
]," schema": { "matches": [{ "
prev_record_id": "R1", "current_
record_id": "R1", "state": "复用", "
confidence": 0.9, "reason": "..." }]," 
introduced": [{ "current_record_id":
"R9", "state": " 新增 ", "confidence":
0.8, "reason": "..." }]," removed": [{ "
prev_record_id": "R8", "state": "移除",
"confidence": 0.8, "reason": "..." }]} } }
```

B 功能性用户界面分类体系

表 9 列出了 DesignEvo 中使用的功能性 UI 组分类法。每个标签代表一个功能上连贯的页面区域：其中的元素被归为一组，是因为它们共同服务于相同的面向用户的用途，例如导航、输入、数据展示、状态反馈、叠加层交互、身份、推广或第三方嵌入。因此，该分类体系在页面区域层面保留了功能性 UI 组

，同时记录源组件名称，作为合并到每个组标签中的设计系统组件术语的证据。

C 功能性 UI 组分布

表 10 报告了评估数据集在一级和二级分类体系层级中标注的功能性 UI 组分布。每个计数对应一个标注的功能性 UI 组。数据显示，数据展示是最大的一级类别，占所有标注组的 58.25%，其次是导航，占 24.15%。在二级层级中，卡片是最常见的类型。该数据集还包括全局页面区域、导航区域、输入区域、状态区域和命令区域，反映了评估数据集中反复出现的功能性 UI 组。

应将此分布理解为数据集覆盖范围，而非分类体系覆盖范围。表格 9 中的分类体系定义了 DesignEvo 使用的功能性用户界面分组标签空间，而表格 10 仅报告在抽样的移动端屏幕中观察到的组类型。因此，某些分类体系类别在此表格中缺失或较少出现，是因为它们在评估数据集中出现频率较低，而非因为它们超出了 DesignEvo 的功能范围。

表9 。Desig nEvo。

主要 分类— omy	次级分类体系 来源组件		描述
全局元素	页眉	页眉；应用栏；导航栏	页面或网站顶部区域，用于组织全局导航、品牌塑造和页面级控件。
全局元素	页脚	页脚	页面或网站底部区域，用于归组辅助导航、版权文本和相关链接。
导航	面包屑导航	面包屑导航	用于将有序层级项目分组以显示用户当前位置的路径导航块。
导航	搜索	搜索	将查询字段与相关控件（如图标、筛选器或建议）组合在一起的搜索区块。
导航	侧边栏	侧边栏；工具栏；全局栏	用于归纳导航条目的侧边区域以及相关的工具条目，例如账户、设置或通知。
操作	操作栏	操作栏	将同一页面、分区、项目或选定集合的操作进行分组的命令区域
输入	输入	自动补全；组合框；筛选输入；文本字段；文本区域	页面、分区、项目或选定集合。以字段为中心、将一个输入组合在一起的输入块目标，并提供建议、选项或筛选行为。
输入	选择器/下拉菜单选择-tor	日期选择器；选择；下拉菜单；分段控件	用于分组选项以供选择的选择块一个值, view, mode, 或状态。
输入	Form	表单；筛选表单	面向任务的输入区域，用于组合多个字段、标签、辅助文本和操作，以便提交、筛选或配置。
数据展示	Card	卡片；磁贴；手风琴；可折叠项分区	用于分组的自包含内容或任务块相关文本 ，媒体 ，元数据 ，和操作。
数据展示	统计面板	统计面板；指标面板；关键绩效指标面板	汇总多个关键值的摘要区域或用于快速比较或监控的指标。
数据展示	可视化	图表；图形；地图	将视觉标记分组的数据呈现区块，用于理解数值或模式的标签、图例或控件，值或模式。
数据展示	时间线	时间线；活动组件；Activity Feed	按时间顺序排列事件、活动、阶段、更新或日志的区域。
数据展示	Grid	卡片网格；网格布局	将相似项目以二维排列方式分组展示的重复内容区域。
数据展示	代码片段	代码片段	将代码与相关标签、语言指示器或复制操作组合在一起的代码展示块。
数据展示	数据表	数据表	将行、列组合在一起的结构化数据块，表头 ，以及相关的状态或控制区域。
数据展示	List	列表；菜单；操作列表	用于归组相似条目的重复项目区域，包括类似菜单或操作列表的条目，它们构成一个连贯的区块。
数据展示	Hero	主视觉单元；主视觉区块；主视觉横幅；轮播	醒目的介绍区域，汇集标题内容、辅助文本、媒体和主要操作。
数据展示	内容块	提示内容；推广内容；摘要	用于强调或作为摘要区域，对相关内容进行分组标题、正文文本、媒体或操作内容。
状态	消息	警示横幅；内联消息；提示条；警告消息；提示框通知	将状态消息与相关上下文、严重程度或关闭控件组合在一起的反馈区域。
状态	空状态	空状态	用于说明缺少内容、结果、数据或项目，并且可能包含一个恢复操作。
状态	加载器	加载旋转指示器；骨架屏加载器	用于指示待处理状态的加载反馈区域
状态	进度	步骤指示器；进度条	内容或正在进行的操作。显示完成情况的进度反馈区域在任务、流程或序列中的状态。
叠加层	对话框/模态框	对话框；模态框；底部弹层；弹出窗口	将临时内容归组的叠加层区域以及当前页面上方的相关操作。
图像类型	徽标/广告区块	徽标；品牌徽标；广告；广告；广告格式；促销；市场营销横幅	基于图像的身份标识或宣传区块，用于呈现品牌标识、活动内容、赞助商标签、链接或操作，作为一个易于识别的视觉区域。
第三方嵌入式-ded	嵌入式小部件	小部件；内嵌框架；股票行情 ticker；社-交按钮；第三方组-件	显示为一个整体的嵌入式外部模块页面内可识别的功能区域。

表10. 主分类体系和次级分类体系中的功能性 UI 组类型分布。评估数据集包含 3,478 个标注的功能性 UI 组。

主分类体系	次级分类体系	组	百分比
数据展示 2,026 (58.25%)	Card	1,142	32.83%
	统计面板	292	8.40%
	数据表	209	6.01%
	可视化	166	4.77%
	List	128	3.68%
	内容块	60	1.73%
	时间线	18	0.52%
	Hero	11	0.32%
导航 840 (24.15%)	侧边栏	305	8.77%
	面包屑导航	297	8.54%
	搜索	238	6.84%
全局元素 326（9.37%）	页眉	286	8.22%
	页脚	40	1.15%
输入 251（7.22%）	Form	150	4.31%
	选择器/下拉选择器	101	2.90%
状态 32（0.92%）	消息	32	0.92%
操作 3 (0.09%)	操作栏	3	0.09%
总计		**3,478**	**100.00%**