Figma2Code: 面向 Figma 设计稿的自动代码生成方法

朱琳[1)], 封颖超杰[1)], 朱航[1)], 王斯加[1)], 朱闽峰[2)], 喻晨昊[1)], 张钰荟[1)], 许达兴[3)], 赵德明[4)], 冯玉君[3)], 陈为[1)*]

[1)] (浙江大学计算机辅助设计与图形系统全国重点实验室 杭州 310058)
[2)] (浙江大学软件学院 杭州 310058)
[3)] (OPPO 广东移动通信有限公司 深圳 518000)
[4)] (成都欧珀通信科技有限公司 成都 610000)
(chenvis@zju.edu.cn)

摘 要: 设计类创作工具已被广泛用于提高用户界面的设计效率, 然而, 根据设计稿开发代码是一件耗时费力的工作. 针对现有的设计稿自动转代码的方案面临的代码可用性和复现结果准确性等问题, 基于 Figma 设计工具提出一种自动代码生成方法——Figma2Code. 首先, 通过节点和图层优化提高设计稿元数据质量; 其次, 采用元数据标注信息的语义理解和图像识别技术识别组件; 然后构建一套通用型的中间态数据结构, 表示优化后的元数据和识别后的组件属性, 以支持多种代码语言的生成; 最后, 基于模板生成可用代码, 并通过函数抽取和元素循环输出提高代码可用性. 采用生成代码的复现样式准确度量化评估和基于专家经验的代码可用性定性评估, 证明了所提方法的有效性.

关键词: 设计稿转代码; 逆向工程; 用户界面; 深度学习
中图法分类号: TP391.41 **DOI:** 10.3724/SP.J.1089.2023-00336

Figma2Code: Automatic Code Generation Method for Figma Design Drafts

Zhu Lin[1)], Feng Yingchaojie[1)], Zhu Hang[1)], Wang Sijia[1)], Zhu Minfeng[2)], Yu Chenhao[1)], Zhang Yuhui[1)], Xu Daxing[3)], Zhao Deming[4)], Feng Yujun[3)], and Chen Wei[1)*]

[1)] (*State Key Laboratory of CAD&CG, Zhejiang University, Hangzhou* 310058)
[2)] (*School of Software Technology, Zhejiang University, Hangzhou* 310058)
[3)] (*Guangdong OPPO Mobile Telecommunications Corp., Ltd., Shenzhen* 518000)
[4)] (*Chengdu OPPO Telecommunications Technology Corp., Ltd., Chengdu* 610000)

Abstract: Design tools are widely used to improve the efficiency of user interface (UI) design. However, coding for designs is a tedious and time-consuming task. To address the issues in terms of code availability and reproduction accuracy in the existing automatic design-to-code solutions. We propose Figma2Code, a method for automatically generating code for Figma-based designs. First, nodes and layers are reorganized in the Figma metadata to improve quality, and components are identified based on semantic understanding of metadata annotation and image recognition. Then, both the optimized metadata and recognized components are transformed into a universal intermediate state to support multiple code generation. Thereafter, code is created from templates and improved by splitting codes into functions and abstracting similar UI elements.

收稿日期: 2023-06-17; 修回日期: 2023-11-02. **基金项目**: 国家自然科学基金重点项目(62132017). **朱琳**(2000—), 女, 硕士研究生, 主要研究方向为可视化; **封颖超杰**(1997—), 男, 博士研究生, 主要研究方向为可视化; **朱航**(2000—), 男, 硕士研究生, 主要研究方向为可视分析; **王斯加**(1998—), 女, 硕士研究生, 主要研究方向为可视化; **朱闽峰**(1993—), 男, 博士, 研究员, 主要研究方向为人工智能、可视分析; **喻晨昊**(1998—), 男, 硕士研究生, 主要研究方向为图形渲染; **张钰荟**(1998—), 女, 硕士研究生, 主要研究方向为可信 AI、隐私计算和联邦学习; **许达兴**(1991—), 男, 学士, 高级应用工程师, 主要研究方向为 UI、系统组件化; **赵德明**(1997—), 男, 学士, 系统工程师, 主要研究方向为 Android 系统; **冯玉君**(1989—), 女, 硕士, 高级应用工程师, 主要研究方向为 UI; **陈为**(1976—), 男, 博士, 教授, 博士生导师, CCF 会员, 论文通信作者, 主要研究方向为可视化、可视分析、大数据分析.

We quantitatively evaluated the accuracy of the generated code reproduction style and qualitatively evaluated the usability of the code based on expert experience, which demonstrates the effectiveness of the proposed method.

Key words: design to code; reverse engineering; user interface; deep learning

在工业软件的开发过程中，诞生了各类用户界面(user interface, UI)设计协作工具，它们已被广泛运用于业务开发流程以提高效率，如 Figma①和 Sketch②. 这些工具允许设计师自由创建设计原型，并允许开发人员直接访问设计稿可视图像与包括图层类型、样式、空间位置在内的多模态 UI 信息. 尽管如此，开发人员在根据设计稿进行代码开发时仍然需要巨大的工作量；同时，团队成员间的协作开发也导致代码之间可能存在规范性的差异和样式准确度的损失.

现有的设计稿转代码方案大多依赖于单一的数据信息. 其中，很多轻量级的插件③④依赖对设计稿元数据(meta-data)的解析，这类数据能够提供准确的设计稿层级关系、节点样式等信息，帮助精准地复现 UI 样式和定位；但它们往往无法识别设计稿中的功能性组件，包括按钮、开关等. 学术界的研究[1]主要利用图像视觉数据进行代码生成，相关工具如 Screenshot2Code⑤是通过图像识别技术实现图形用户界面(graphical user interface, GUI)组件的识别；但是，仅根据图像数据难以生成精确的组件样式. 这些方法没有充分结合设计稿元数据和图像视觉数据的互补优势，导致生成代码的可用性和复现结果受到影响.

为了解决上述问题，本文提出一种设计稿转代码方法——Figma2Code，更充分地利用设计稿元数据和图像数据生成样式准确、可用性高的代码. Figma2Code 通过优化设计稿元数据的节点和图层关系获得准确的 UI 组件样式和图层结构，利用图像识别技术识别组件类型；利用启发式规则对识别结果进行检查，进一步提高准确性；结合规则手段并优化结果，生成具有通用性的中间态数据结构，以适应不同平台的代码语言结构，帮助开发人员高效地自动化生成代码.

针对 30 个不同风格、不同尺寸的设计稿进行实验，通过用户评估验证本文方法以及基于该方法实现的系统的有效性和实用性.

1 相关工作

1.1 图像转代码

早期，面对低保真草图，SILK[2]使用手势识别算法[3]将数字图形转换为应用代码. 之后，出现了高保真图像，Dixon 等[4]基于像素进行 GUI 逆向工程，但该方法完全依赖于工程启发式方法进行识别；Nguyen 等[5]利用计算机视觉(computer vision, CV)和光学字符识别(optical character recognition, OCR)技术从图像中提取结构信息，实现了图像到 Android 代码的转换. Beltramelli[6]提出 pix2code，通过注入机器学习知识，将图像转代码带入一个新的时代. Chen 等[7]提出一种神经机器翻译器，结合卷积神经网络(convolutional neural network, CNN)与循环神经网络(recurrent neural network, RNN)提取 GUI 骨架；Moran 等[8]通过 CNN 分类 GUI，实现组件识别；Robinson[9]通过对比实验，证明了深度学习在自动化设计稿转代码上优于经典的图像识别方法.

上述工作在图像转代码研究中已经获得巨大成果，但转换结果往往依赖于设计的视觉理解和 GUI 模板，难以支持多种平台代码的扩展. 不同于以上的工作，本文系统充分地结合图像数据与设计稿元数据，进行属性提取和组件识别，提高转换的准确性；加入中间态的概念，以适应不同平台的代码语言结构.

1.2 设计稿转代码

现有的开源设计稿转代码工具集中于依赖设计稿元数据的工程启发式方法，如 Figma to Code,

① https://www.figma.com
② https://www.sketch.com
③ https://github.com/bernaferrari/FigmaToCode
④ https://github.com/BuilderIO/figma-html
⑤ https://github.com/emilwallner/Screenshot-to-code

Builder.io, 它们支持多种代码语言的转换, 但不支持 GUI 组件的智能识别, 导致代码可用性差.

商业的解决方案在组件识别上表现良好. 在国外, Yotako①能支持多种代码语言转换和 15 种组件的识别, 但高度依赖设计稿标注; Anima②使用图像识别的方法支持自定义组件识别复用, 但缺少对图层结构的优化. 在国内, 2019 年, 阿里巴巴公司在 imgcook③中提出较为完整的工作方法, 结合深度学习的方法实现了基于图像视觉数据的组件和图层结构的识别[10-11]; 但其元数据信息的干预校正能力有限, 在布局的处理上灵活性稍显不足.

本文综合以上工作, 在结合工程方法和深度学习识别的基础上, 提出元数据优化和标签检查, 以弥补元数据的缺陷和识别结果不稳定的问题.

2 研究背景

2.1 UI 设计协作工具

市场上的 UI 设计协作工具大部分具有相似的元素类型供设计师创建 UI 设计稿, 并支持图层的嵌套和元素样式的修改. 部分设计工具(如 Figma, Sketch, MasterGo④等)支持以节点树的形式导出设计稿元数据信息, 并包含节点的类型, 样式信息(颜色、字体、边框等), 空间信息(边距、位移等), 为设计稿自动生成代码提供精准的数据支持.

Figma 封装了 31 种节点类型, 根据节点特征和适用场景, 本文将常用的节点分为 2 类: (1) 容器节点. 包括 Frame, Instance, Component 和 Group, 这类节点相当于组或图层容器, 支持包含其他节点, 其中, Figma 细致的节点分类为工程规则识别和标签修正提供了支持. (2) 基础节点. 如 Text, Rectangle, Ellipse 等, 这类节点是叶子节点, 内部无法再嵌套其他节点.

2.2 GUI 数据集

在深度学习时代, 以模型为中心的方法需要大量的领域数据支持. 目前, ImageNet[12]在图像理解领域的深度学习模型的发展中发挥了重要作用; 在移动应用数据集方面, RICO[13]包含超过 78 000 个来自 Android 应用程序的 UI 组件和对应的元数据; ReDraw 收集了 14 382 个独特屏幕和 191 300 个标记的 GUI 组件, 并通过艺术家手绘的方式重构了一系列来自 Android 系统的 UI 组件; 此外, 还有 ERICA[14]和 LabelDROID[15]均包含移动 UI 截图.

当前的 UI 风格和设计标准与现有的开源数据集仍存在较大差异. 基于 Redraw 和 RICO 数据集, 本文收集了来自 Figma 开源社区的 Material Design3 风格的组件设计稿, 提取超过 4000 个 GUI 组件作为增强数据集进行训练.

2.3 需　求

基于对现有的设计稿转代码工作的分析以及业内相关产品的调研, 并结合与 2 位 UI 设计师和 4 位开发人员的访谈, 本文明确了设计稿转代码的整体流程和主要问题, 总结出以下设计需求.

R1. 节点编码信息优化. 在使用 Figma 工具进行 UI 设计的时候, 设计师通常缺乏统一、明确的设计稿规范, 导致创建的 GUI 图层可能存在混乱、冗余等问题. 直接照搬这样的图层数据会影响代码的可用性, 因此需要对 Figma 的图层信息进行多层次的优化, 降低节点树的复杂度和深度.

R2. 多模态的 GUI 组件识别. 已有的研究集中于使用单一链路提取信息, 使用 CV 技术提高识别的准确度就放弃了更精准的编码信息, 而使用工程手段就难以识别 GUI 组件. 目前缺少一种结合工程与 CV 并弥补各自缺点的方法, 进一步提高识别和信息提取的准确性.

R3. 针对识别结果的标签检查. CV 的识别针对单一节点, 不支持对上下文包含关系的理解, 识别结果错误可能产生不符合逻辑的包含关系. 根据这一特点, 利用节点树层级关系的标签比较, 可以进一步修正错误标签, 提高识别的准确度.

R4. 适应多种代码语言的能力. 一套设计稿可能被用于多平台(多语言)开发实现, 这就要求设计稿转换的结果需要具备通用性和可扩展性.

3 Figma2Code 方法

基于设计需求, 本文提出一种 Figma2Code 方法. 图 1 所示为以 Figma 设计稿为例, 描述该方法的流程. 其中, 首先获取设计稿的节点元数据(见

① https://yotako.io
② https://www.animaapp.com
③ https://www.imgcook.com
④ https://developers.mastergo.com

A1)和屏幕截图(见 A2)作为数据输入源；其次针对元数据进行优化(见 B)，修剪冗余节点信息和图层结构，降低元数据的复杂度，提升代码结构可用性；再使用启发式规则和图像识别技术，从设计稿中识别检测 GUI 组件，并对结果标签进行检查(见 C)；根据识别结果和节点树生成具有通用性的领域专用语言(domain specific language, DSL)结构(见 D)；最后根据 DSL 结构生成不同语言代码模板(见 E).

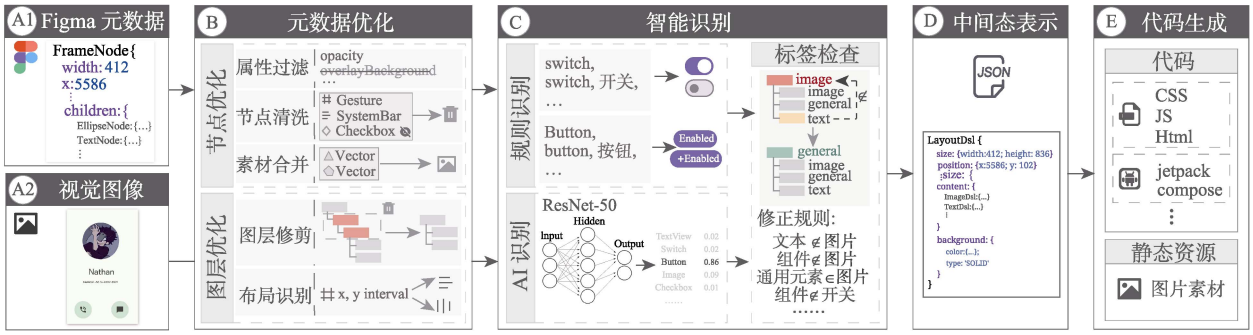


图 1 Figma2Code 方法流程

3.1 数据输入

Figma2Code 方法面向 Figma 设计稿画板，从中提取设计稿元数据信息和图像数据作为输入数据. 画板中的图层结构与节点样式信息以 json 数据的格式导出，并且支持导出节点的图像文件.

3.2 元数据优化

设计稿元数据包含一些与生成代码无关的属性和冗余的嵌套结构，该阶段的目标是精简数据结构树，获得足够干净的信息来进行后续的识别与数据转换.

3.2.1 节点优化

节点优化从节点属性过滤、无用节点删除、合并素材节点 3 个角度进行，其示例如图 2 所示.

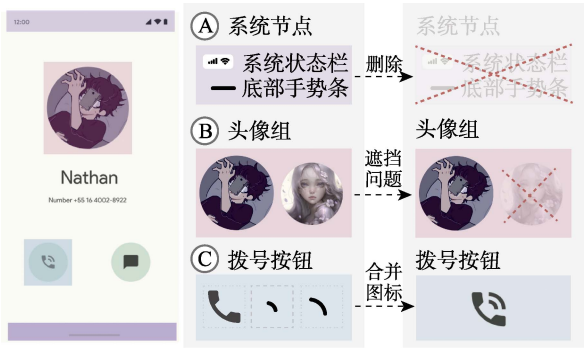


图 2 节点优化示例图

节点携带的信息中存在对设计稿的样式无影响的信息(如图层锁定)，无法帮助代码的生成，且增加了节点树的复杂程度. 本文通过封装新的节点数据类型，以过滤无关属性.

无用节点的存在将对复现的结果和代码产生影响. 例如，一些无需代码实现的系统组件(见图 2 中的 A)和一些被上层节点遮挡导致不可见的节点(见图 2 中的 B)产生了大量的无效代码，影响代码可读性和可用性，有的甚至导致复现结果错误. 为了解决这一问题，本文通过比较节点属性与名称模糊匹配的方式，识别并删除不可见节点与系统组件节点.

Figma 允许设计师自由绘制组装图标，导致部分图标可能由多个图形节点组合而成(见图 2 中的 C). 本文将属于同一图标的图形节点合并，并将合并后的图像导出作为静态资源使用.

3.2.2 图层优化

本文参考了约 60 张业内移动应用开发设计稿，以及超过 200 张公开设计稿，总结了冗余嵌套过滤和图层布局识别 2 种图层优化方法.

冗余嵌套过滤针对图层的排布结构. 由于缺乏统一、明确的设计稿规范，因此元数据节点树中存在大量冗余图层，如不包含任何样式信息的图层、分散标注一套样式信息的图层组合等. 本文通过删除无样式信息的图层和合并多层级样式等方法，优化节点树结构.

图层布局识别优化子元素的排列模式. Figma 支持用户设置容器节点的子元素的布局模式，实现页面相对布局和自适应的效果. 然而，部分设计稿可能存在未指定布局模式的情况. 本文针对缺少布局信息的容器节点，通过计算子元素位置的间距重新识别布局信息，进一步提高后续生成代码的可用性和兼容性.

3.3 智能识别

本文结合启发式规则的组件识别方法和深度学习的图像分类方法识别设计稿中的组件，利用

标签检查的手段进一步提升识别结果的准确性.

3.3.1　基于启发式规则的组件识别方法

通过分析节点元数据, 提出 3 种基于工程启发式方法的识别手段: (1) 基于节点标注信息的识别方法; (2) 基于元素构成信息的组件识别方法; (3) 基于层级结构的复合组件识别方法.

基于节点标注信息的识别方法从节点名称和描述信息中识别组件. 设计师往往会将组件类型标注在这类信息中, 该方法通过字符串模糊匹配算法提取标注信息, 根据设计师的意愿进行识别, 支持超过 40 种标签的识别, 结果更加可靠.

基于元素构成信息的组件识别方法通过判断内部元素的特征和组合信息识别组件, 如 switch=(圆角矩形+圆形)&&圆形圆心位于圆角矩形内部. 该方法针对具有单层结构并且特征明显的组件, 支持包括 switch(开关)在内的 6 种组件的识别.

基于层次结构的复合组件识别方法用于识别具有循环特性的复合组件, 如 list(列表)和 navigationBar(导航栏). 这类组件内部往往包含多个结构相似的项. 从代码开发的角度看, 组件内部所有子项基于同一模板, 可通过循环和传递参数生成子项列表. 本文通过比较节点结构树识别循环体组件, 对于每个潜在的循环项(如列表项元素), 从宽度、高度和内部子元素 3 个方面判断相似性. 元素必须满足宽度与高度均差距在一定阈值内, 且与兄弟元素具有 60%以上的公共相似子节点时, 认为该元素为循环项, 根据抽取的结构模板为后续代码转换提供支持.

3.3.2　基于深度学习的图像分类方法

基于深度学习的图像分类方法用于对设计稿图像数据的组件识别, 包括图像分割、图像分类、关联分类结果与元数据节点.

图像分割的目的是从 UI 设计稿页面中获取 GUI 组件的局部图像, 用于组件分类. 传统的图像分割技术, 如 Canny 边缘检测算法[16]和 R-CNN[17]深度学习模型, 无法较好地处理组件的复杂背景和形状、多层级复合组件等场景. 相比之下, Figma 元数据中的节点长宽和坐标信息有助于实现精确的组件图像分割. 因此, 本文通过获取这类信息分割图片区域, 获取局部图像.

在对 UI 元素图像进行分类的过程中, 本文使用 ResNet-50[18]模型进行预测. 训练模型时, 采用 Redraw 和 RICO 数据集作为初始数据集, 并对它们进行人工筛选, 以提升数据质量. 此外, 本文探索并收集 Figma 社区的开源设计稿, 制作了面向特定设计风格(如 Material Design3)的自定义数据集, 利用图像裁切调整图像颜色、对比度等方式进行数据增强, 最终获得 15 000 条自定义风格数据. 基于上述 3 个数据集, 本文获得包含 17 800 条组件图像-标签数据的数据集, 覆盖 button 和 switch 等 10 多种组件类型; 按照 8∶1∶1 的比例划分训练集、验证集和测试集, 最终识别准确率超过 90%.

3.3.3　标签检查

上述识别方法仅针对指定的节点信息和图像数据, 缺少对节点间包含关系的理解. 本文基于设计稿元数据节点树结构, 提出一种标签类型检查的方法, 基于一套标签嵌套规则对识别结果进行修正, 该方法包括标签自检查和层级检查.

标签自检查从节点自身属性出发, 通过比较节点类型和节点属性判断标签合理性. 由于元数据信息针对一些节点类型有明确标识, 如文本, 因此一旦有组件被识别成文本, 就将其标签修改为通用节点.

层级检查利用节点包含关系, 结合层级结构上下文校正标签. 本文根据组件的包含关系将组件分为基础组件与复合组件, 其中, 基础组件具有单一的结构, 内部不包含其他组件, 如 checkbox(复选框)和 switch(开关)等; 复合组件如 card(卡片)内部可能包含按钮这类的组件. 如果基础组件内部出现其他组件, 则认为标签有误, 将其修改为通用容器, 在后续代码转换的过程中不以功能性组件的形式实现. 此外, 如果父节点与子节点存在相同标签, 优先保留子节点标签.

3.4　中间态表示

基于 Pacheco 等[19]提出的中间模型转换, 本文设计一套基于 DSL 结构的中间态表示法来表示转代码所需的组件类型、属性、层级等信息, 并实现了一套从设计稿元信息到 DSL 结构树的转换方法, 以生成一套适应不同平台语言的中间表示.

3.4.1　中间态结构设计

UI 组件通常包含通用属性(如宽高、坐标)和专有属性(如形状). 如图 3 中的 A 所示, 本文抽取所有组件的通用属性构建父类, 并根据常见组件的专有属性设计超过 20 种 DSL 结构, 每种 DSL 结构继承公共的父类并维护自身特有的属性. 为了表示 list 列表等复合组件(见第 3.3.3 节), 本文设计一套特有属性, 用于记录复合组件的嵌套结构和对内部元素的约束. 图 3 中的 B 所示为一个按钮的 DSL

结构设计，其中，layout 属性标记对子元素的约束，包括子元素排列方向(layoutMode)和子元素在垂直方向和水平方向上的对齐方式(horizontalAlign，verticalAlign)；content 属性是一个 DSL 数组，表示包含的元素，并以此形成嵌套结构，生成 DSL 树.

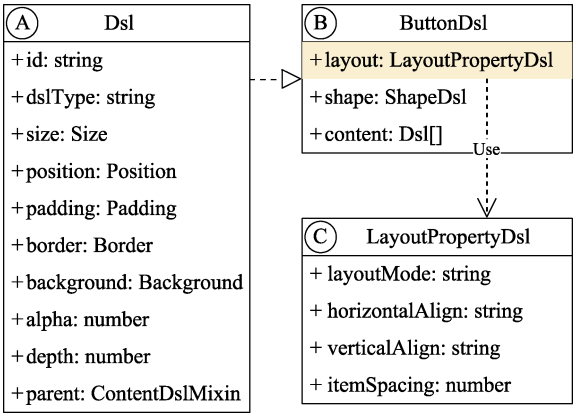


图 3 按钮组件 DSL 设计

3.4.2 跨层级属性提取

本文提出跨层级属性提取，用于构建从 Figma 节点属性到代码组件属性的转换方法．很多组件的属性需要从对应的 Figma 节点的内部子元素中获取．以图 4 的按钮组件为例，其对应的图层节点(Frame)缺少颜色填充信息，该填充信息需要从其内部的矩形获取．通过跨层级提取，从子元素中获取背景色填入按钮的 DSL 中.

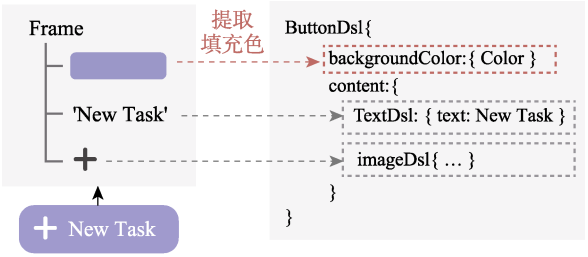


图 4 中间态转换过程图

3.4.3 DSL 结构树优化

DSL 结构对代码生成起决定作用，为了提高代码可用性和准确性，本文在 DSL 树成功创建并得到信息完整的中间态数据后，进一步优化 DSL 结构树，包括重排序同层节点，保证循环体的准确生成.

3.5 代码输出

代码输出的目标是生成具有规范性、可用性的多语言代码．本文采用基于模板的目标代码生成方法，预先整理常用的各类组件并生成对应的代码模板，将 DSL 中的信息整合到模板中，生成代码文件．基于本文方法实现的系统已支持 jetpack compose 和 html 的代码生成，这 2 套语言框架能够分别运行于 Android 和前端网页的环境.

本文系统遵循代码缩进、命名格式统一等编程规范，根据组件类型对函数进行命名．在代码可用性上，该系统支持循环输出和函数的抽取拆解；针对循环体组件，系统生成子项代码模板，提取子项可变数据并构建参数变量数组，在代码层面支持循环代码输出；针对页面功能模块实现函数抽取，抽取相应功能的代码组合成函数，如将循环列表和导航栏组件抽取为独立函数，在主结构函数中调用；统计函数内代码字节数、行数等，拆解长函数，对各布局进行逻辑上的解耦，提高代码的可维护性与利用率.

4 实验与结果分析

通过案例分析，验证本文系统的有效性和实用性，评估 Figma2Code 生成的应用程序的视觉相似性和代码可用性．本文收集了来自 Figma 社区以及由某手机公司专业设计团队设计的不同风格、不同平台的 30 份设计稿进行实验；其中，包含 Material Design3 风格、该手机公司的系统风格和自由设计师的设计风格，以及对话窗口、设置页、首页和相册页等，涉及约 16 种组件.

4.1 视觉相似性

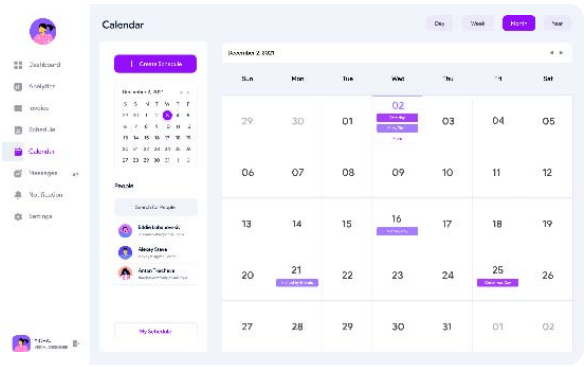
代码复现的首要目标是应用程序视觉上与设计稿的相似性．设计了综合评估和对照实验，通过计算、复现截屏和设计稿原图的相同像素点的占比，评估代码复现结果的视觉相似性.

4.1.1 综合评估

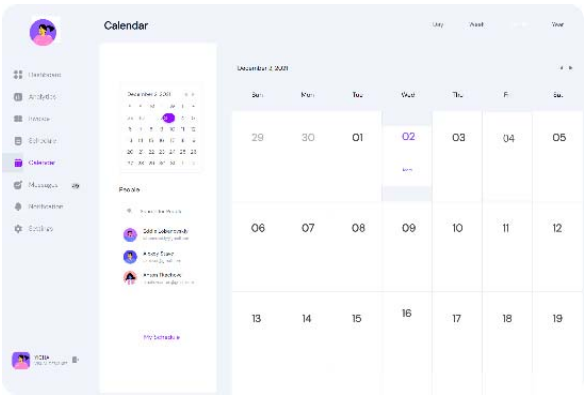
对于收集的所有实验设计稿计算像素平均相似度．实验结果表明，jetpack compose 复现结果的相似度达到 86.7%，html 复现结果的相似度达到 91.0%，证明了 Figma2Code 生成的代码在复现结果的视觉相似度上表现良好.

4.1.2 对照实验

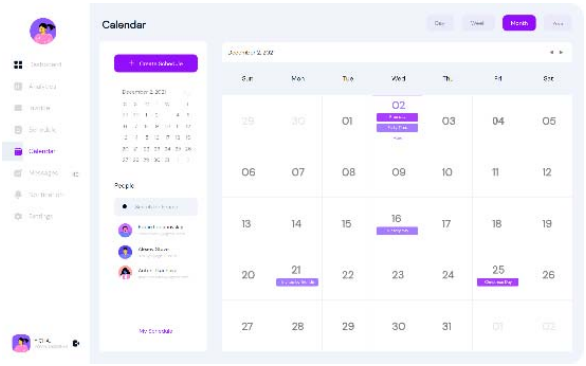
与市场上成熟的产品 imgcook 进行对照实验，证明 Figma2Code 方法的有效性．实验中，选取首页、日历页、图表页和列表页的案例，每个案例包含 3~5 张设计稿，使用同一套语言进行代码复现．图 5 所示为 2 个工具在一份日历设计稿上的复现结果对比，表 1 所示为 2 组结果的平均像素相似度.



a. 原设计稿截屏



b. imgcook



c. Figma2Code

图 5　日历设计稿复现结果对比

表 1　不同工具基于像素的相似度比较结果　%

案例类型	imgcook	Figma2Code
首页	67.6	88.9
日历	65.0	85.5
可视化图表	29.1	88.9
列表	83.0	94.1
平均值	60.6	90.4

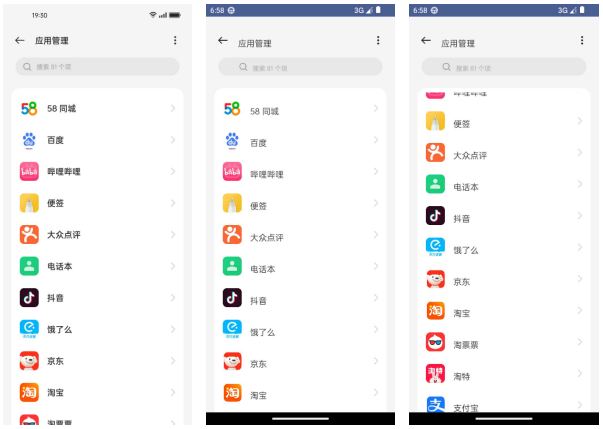
可以看出，Figma2Code 生成的应用页面代码复现结果整体表现优于 imgcook，且在复杂页面设计稿上提升较大，证明该方法有效地提升了代码复现结果的样式准确度.

4.2　代码可用性

通过 2 个案例验证代码的可用性，并在后续的用户研究中进行辅助评估.

4.2.1　案例 1. 循环长列表

现有的产品大多只支持对设计稿可视区域的复现，导致长组件的部分信息丢失. 图 6 所示为 Figma2Code 复现包含循环长列表组件的移动端设计稿的结果，可以看出，实现了循环生成列表项和长屏滚动，画布外的元素通过页面滚动的方式被复现，具有较强的代码可用性.

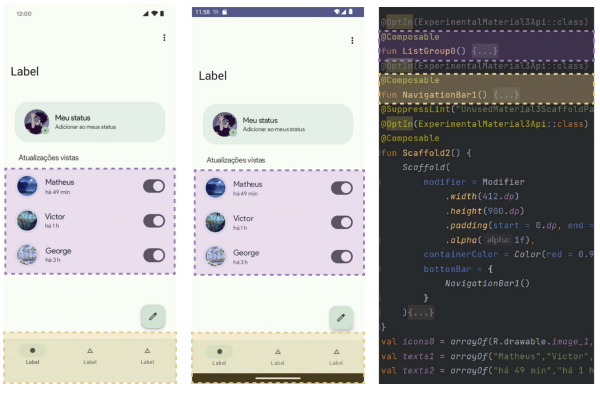


a. 原始设计稿　　b. 滚动前　　c. 滚动后

图 6　案例 1. 循环长列表复现结果

4.2.2　案例 2. 函数抽取

在实际代码开发的过程中，开发人员通常会对页面代码进行解构，抽取功能函数以实现代码复用，一些代码语言也对一个函数承载的代码量进行了约束；但是，现有的工具几乎不支持对功能函数的抽取. 图 7 所示为一个 Material Design3 风格的开源设计稿的页面复现情况，以及生成的 jetpack compose 的代码结构，可以看出，本文系统抽取了底部导航栏(NavigationBar)和列表(ListGroup)作为独立的组件函数，在主函数中调用.



a. 原始设计稿　　b. 复现结果　　c. 代码结构

图 7　案例 2. 函数抽取

5 用户研究

为了验证 Figma2Code 的实用性，本文邀请 6 位用户(2 位女性, 4 位男性)参与实验. 其中, 2 位在前端开发领域有 3 年以上开发经验，4 位具有 3 年以上移动端开发经验.

5.1 实验流程

在系统介绍环节，针对插件使用流程和生成的文件内容进行了约 10 min 的介绍. 介绍完成之后，所有用户均表示能够理解系统功能和使用方法.

在实验阶段，用户可以自由探索开源设计稿或自己创建设计稿，利用工具完成设计稿的代码生成，并利用对应的集成开发环境(integrated development environment, IDE)运行生成的代码. 每位用户需要对至少一份生成的代码进行评估和优化调整. 整个实验过程持续 10~20 min.

5.2 结果分析

实验完成后，用户需要回答一系列半结构化问题，其中，每个问题包含一个评分项，1 分表示非常不满意，5 分表示非常满意. 用户还需提供一些开放式的主观说明. 对问卷数据进行整理后，统计结果如下:

(1) 在代码复现结果的视觉相似性上，5/6 用户表示非常满意，1/6 用户表示比较满意. 用户在相似性上给予了较高的评价，如“复现结果看起来与设计稿原图相似度很高”.

(2) 在代码可用性的表现上，3/6 用户表示非常满意，3/6 用户表示比较满意. 用户认为“循环结构的处理很有效”，也有用户认为“如果能加上人为干预，可以进一步提高代码可用性”.

(3) 在系统使用体验的评价中，4/6 用户表示非常满意，2/6 用户表示比较满意. 用户均认为“系统整体很轻量，简洁易操作” “生成代码的速度比较快”.

(4) 被问及是否会考虑在实际的开发流中采用 Figma2Code 时, 5/6 用户表示非常愿意, 1/6 用户表示比较愿意. 用户对系统的功能价值给予了较高的评价，认为“对提高设计稿代码复现的效率有很大帮助，能减少很多工作量”.

此外，用户也提供了一些有价值的建议，如适当加入一些人机交互，允许用户对识别结果、DSL 结构属性进行人为修改等.

6 讨 论

Figma2Code 方法具有普适性，适用于大部分提供设计稿元数据的 UI 设计工具. 如第 2.1 节所述，大部分设计工具提供相似的节点类型功能，包括容器节点、文本节点等，同样采取图层树结构进行画布组合编排，并且支持节点的命名. 本文考察了包括 Figma, Sketch 和 MasterGo 在内的 6 个被广泛使用的 UI 设计协作工具，其导出的元数据结构均符合上述特征. 因此, Figma2Code 方法也能适用于其他设计工具.

7 结 语

本文提出 Figma2Code 方法，能够高效、准确地自动生成 GUI 代码. 在此基础上，设计并实现了一个基于 Figma 设计稿自动化生成代码文件的工具，支持 2 种平台语言代码的生成. 案例分析和用户研究结果表明，本文系统提供了良好的用户体验，与其他产品相比，生成的代码具有更高的准确性.

目前，Figma2Code 方法中的一些步骤采用规则的方法来识别，如循环组件、图层结构等，并通过实验验证了其有效性. 未来将引入深度学习方法，进一步提高模型的通用性和准确率，在其他环节提高识别的灵活性.

本文也从用户研究中挖掘了一些新的功能需求，如增强用户交互，包括允许用户自主选择和修改识别结果、修改 DSL 结构等，为未来的工作提供了方向.

参考文献(References):

[1] de Souza Baulé D, von Wangenheim C G, von Wangenheim A, *et al*. Recent progress in automated code generation from GUI images using machine learning techniques[J]. Journal of Universal Computer Science, 2020, 26(9): 1095-1127

[2] Landay J A, Myers B A. Sketching interfaces: toward more human interface design[J]. Computer, 2001, 34(3): 56-64

[3] Rubine D. Specifying gestures by example[J]. ACM SIGGRAPH Computer Graphics, 1991, 25(4): 329-337

[4] Dixon M, Fogarty J. Prefab: implementing advanced behaviors using pixel-based reverse engineering of interface structure[C] // Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. New York: ACM Press, 2010: 1525-1534

[5] Nguyen T A, Csallner C. Reverse engineering mobile application user interfaces with REMAUI (T)[C] //Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering. Los Alamitos: IEEE Computer Society Press, 2015: 248-259

[6] Beltramelli T. pix2code: generating code from a graphical user interface screenshot[C] //Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems.

New York: ACM Press, 2018: Article No.3

[7] Chen C Y, Su T, Meng G Z, *et al*. From UI design image to GUI skeleton: a neural machine translator to bootstrap mobile GUI implementation[C] //Proceedings of the 40th IEEE/ACM International Conference on Software Engineering. Los Alamitos: IEEE Computer Society Press, 2018: 665-676

[8] Moran K, Bernal-Cárdenas C, Curcio M, *et al*. Machine learning-based prototyping of graphical user interfaces for mobile apps[J]. IEEE Transactions on Software Engineering, 2020, 46(2): 196-221

[9] Robinson A. Sketch2code: generating a website from a paper mockup[OL]. [2023-06-17]. https://arxiv.org/abs/1905.13750

[10] Li J Z, Zhou T T, Chen Y N, *et al*. ULDGNN: a fragmented UI layer detector based on graph neural networks[OL]. [2023-06-17]. https://arxiv.org/abs/2208.06658

[11] Chen Y N, Zhen Y K, Shi C N, *et al*. UI layers merger: merging UI layers via visual learning and boundary prior[J]. Frontiers of Information Technology & Electronic Engineering, 2023, 24(3): 373-387

[12] Deng J, Dong W, Socher R, *et al*. ImageNet: a large-scale hierarchical image database[C] //Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. Los Alamitos: IEEE Computer Society Press, 2009: 248-255

[13] Deka B, Huang Z F, Franzen C, *et al*. Rico: a mobile app dataset for building data-driven design applications[C] //Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology. New York: ACM Press, 2017: 845-854

[14] Deka B, Huang Z F, Kumar R. ERICA: interaction mining mobile apps[C] //Proceedings of the 29th Annual Symposium on User Interface Software and Technology. New York: ACM Press, 2016: 767-776

[15] Chen J S, Chen C Y, Xing Z C, *et al*. Unblind your apps: predicting natural-language labels for mobile GUI components by deep learning[C] //Proceedings of the 42nd IEEE/ACM International Conference on Software Engineering. Los Alamitos: IEEE Computer Society Press, 2020: 322-334

[16] Zeng Jun. Research on image edge detection and its application[D]. Wuhan: Huazhong University of Science and Technology, 2011(in Chinese)
(曾俊. 图像边缘检测技术及其应用研究[D]. 武汉: 华中科技大学, 2011)

[17] Girshick R, Donahue J, Darrell T, *et al*. Rich feature hierarchies for accurate object detection and semantic segmentation[C] //Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. Los Alamitos: IEEE Computer Society Press, 2014: 580-587

[18] Wang Yuandong. Research on image classification method and application based on ResNet model[D]. Nanchang: East China Jiaotong University, 2019(in Chinese)
(王元东. 基于 ResNet 模型的图像分类方法及应用研究[D]. 南昌: 华东交通大学, 2019)

[19] Pacheco J, Garbatov S, Goulão M. Improving collaboration efficiency between UX/UI designers and developers in a low-code platform[C] //Proceedings of the ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion. Los Alamitos: IEEE Computer Society Press, 2021: 138-147