南京大學

研 究 生 毕 业 论 文

（申请工程硕士学位）

论 文 题 目 基于图像理解强化学习的通用遍历测试技术

作 者 姓 名 恽叶霄

学科、专业名称 工程硕士（软件工程领域）

研 究 方 向 软件工程

指 导 教 师 陈振宇 教授 房春荣 助理研究员

2022 年 05 月 20 日

学　　号 ：**MF20320212**

论文答辩日期 ：**2022 年 05 月 20 日**

指 导 教 师 ：　　**（签字）**

General Test Input Generation via Deep Image Understanding and Reinforcement Learning

By

Yexiao Yun

Supervised by

Professor Zhenyu Chen and Research Assistant Chunrong Fang

A Thesis
Submitted to the Software Institute
and the Graduate School
of Nanjing University
in Partial Fulfillment of the Requirements
for the Degree of
Master of Engineering

Software Institute

May 2022

学 位 论 文 原 创 性 声 明

任何收存和保管本论文的单位和个人，未经作者本人授权，不得将本论文转借他人并复印、抄录、拍照或以任何方式传播，否则，引起有碍作者著作权益的问题，将可能承担法律责任。

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不句含其他个人或集体已经发表或撰写的作品成果。本文所引用的重要文献，均已在文中以明确方式标明。本声明的法律结果由本人承担。

论文作者签名：_____

日期：　　年　　月　　日

南京大学研究生毕业论文中文摘要首页用纸

毕业论文题目：基于图像理解强化学习的通用遍历测试技术

工程硕士（软件工程领域）专业 2020 级硕士生姓名：恽叶霄

指导教师（姓名、职称）：陈振宇 教授 房春荣 助理研究员

摘 要

在软件测试领域中，遍历测试具有重要地位。遍历测试是指探索待测软件的各种不同状态，以测试其在每一个状态下的界面内容是否正确展示及其功能是否正常。遍历测试方法的核心在于测试输入的生成。目前的遍历测试探索策略主要可分为三类：随机探索策略、基于模型的探索策略和系统性探索策略。随机探索策略以简单快捷为特点，基于模型的策略则有着更好的覆盖率，系统性的探索策略则往往能更快地发现隐藏状态。

目前的遍历测试方法大多基于平台底层的布局文件，而不同平台的软件其底层框架大不相同，导致不同平台的布局文件其分析方式不同，进而给这些遍历测试方法的跨平台移植带来了困难。因此，现有的遍历测试方法，不论其属于何种探索策略，均存在一定的平台局限性，无法实现平台通用。

除此以外，安卓平台支持 H5 页面，这种类型的页面本质上是将一个基于 HTML 5 的页面嵌入到安卓原生页面中，从而导致在布局文件中该 H5 页面在底层的布局文件中只对应了一个单独的控件，不包含 H5 页面的详细内容。除了 H5 页面的问题外，所有平台均支持 Canvas 组件，而 Canvas 组件本质上是一块画布，其中可以画任何内容，与 h5 页面的问题类似，在底层的布局文件中，Canvas 组件的内容同样会被完全忽略。这些被忽略的内容同样是构成页面状态的重要部分；作为遍历测试的基础，页面状态提取的不精确往往会使最终效果大打折扣。

另一方面，不论何种平台均提供了便捷的截屏功能，且生成的截屏质量较高。根据这些截屏，应用深度学习和图像处理技术，我们可以实现对这些截屏布局与内容的理解。利用该深度截屏理解技术，我们可以解决上述平台局限性的问题。

基于上述问题和技术，本文提出了一个基于图像理解强化学习技术的平台通用遍历测试技术 UniRLTest。该技术结合 Canny 边缘识别和光学字符识别的技术，识别截屏中的每一个控件，并根据这些控件进一步生成以控件组、控件行、控件列所表示的页面布局结构，最后利用卷积神经网络、长短期记忆网络等对控件与布局进行嵌入，整合为一个嵌入向量，以此作为待测软件的状态；根据页

面中的控件，生成所有可执行动作，即与该应用的交互操作，包括各类与控件的交互、与窗口的交互以及一些特殊的系统事件；利用深度强化学习技术，根据识别的状态向量为每一个可执行事件赋予一个概率，其概率与当前状态下该动作能带来的预期累计奖励呈正相关，最后基于此概率随机挑选下一个待执行动作，以尽可能提高预期累计奖励，同时保留随机化防止算法陷入局部最优解；在强化学习的奖励函数设计上，系统采用好奇心驱动，根据每一次页面转移后的新页面探索率与页面转移次数来给予合适的奖励，随着新页面探索率和页面转移次数的提高，系统给予的即时奖励会不断降低，通过这种反馈机制，使其逐渐失去对已探索页面的好奇心而增强对未探索页面的好奇心，使其能探索待测应用的更多未知状态。由于状态识别方法基于截屏图像而非基于框架提供的底层布局，该工具具有良好的平台通用性；同时好奇心驱动的深度强化学习也能达到对源码较高的覆盖率，以此保证对足量目标场景的覆盖。

本文还基于以上技术实现了一个同名的遍历测试工具，该工具包含前端、服务端和算法端，其中算法端利用 Tensorflow 和 Flask 等框架技术实现了上述 UniRLTest 技术，而服务端则利用 ADB 和 ChromeDriver 等工具实现与不同平台的交互以及遍历测试的完整流程，而前端则利用 Vue.js 框架和 WebSocket 技术实现了对遍历测试的配置与监控界面；三者的交互基于 HTTP 协议和 WebSocket 协议。

我们组织了若干实验来验证 UniRLTest 的代码覆盖率和事件生成效率。根据我们的实验可以得出，该工具的代码行覆盖率相比常见的遍历测试工具 Monkey、Q-testing 和 WebExplor 较高，而事件生成效率相比而言略低，但不影响效果。

关键词：遍历测试，深度截屏理解，强化学习

南京大学研究生毕业论文英文摘要首页用纸

THESIS: General Test Input Generation via Deep Image Understanding and Reinforcement Learning

SPECIALIZATION: Software Engineering

POSTGRADUATE: Yexiao Yun

MENTOR: Professor **Zhenyu Chen** and Research Assistant **Chunrong Fang**

Abstract

Automated test input generation plays an important role in software testing. Automated test input generation is to exploring various states of a software to test whether the contents of its interface is displayed correctly and whether its function is proper in each state. Currently, exploration strategies can be mainly divided into three categories: random exploration strategies, model-based exploration strategies and systematic exploration strategies. Random exploration strategies are well-known due to their simplicity and fast speed, model-based strategies have better coverage, and systematic exploration strategies tend to discover hidden states faster.

Most of the current test input generation methods are based on the layout files of the platform, and the underlying frameworks of software on different platforms are quite different, resulting in different analysis methods of layout files on different platforms, which in turn brings difficulties to the cross-platform porting of these test input generation methods. Therefore, the existing traversal testing methods, no matter what kind of exploration strategies they belong to, have certain platform limitations.

In addition, the Android platform supports h5 pages. This type of page essentially embeds an HTML5-based web page into an Android native page, resulting in the h5 page only corresponds to a single widget in the layout file, while the contents of the h5 page are ignored. Except for the problem of h5 pages, all platforms support the canvas component in which any content can be drawn. Similar to the problem of the h5 page, in the layout file, the content of the canvas component will also be completely ignored. These ignored contents are also an important part of the state of the page. As the basis

for test input generation, the imprecise extraction of page state often greatly worsens the final effect.

On the other hand, a convenient screenshot function is provided regardless of the platform, and the generated screenshots are of high quality. Based on these screenshots, applying deep learning and image processing techniques, we can achieve an understanding of the layout and widget contents of these screenshots. Using this deep screenshot understanding technique, we can address the platform limitations mentioned above.

Based on the above problems and technologies, this paper proposes a platform-universal traversal testing technology UniRLTest based on deep image understanding and reinforcement learning technology. This technology uses a combination of Canny edge recognition technology and optical character recognition technology to identify each widget in the screenshot, and further generates a page layout structure represented by widget groups, widget rows, and widget columns based on these widgets, and finally uses Convolution Neural Networks, Long Short-Term Memory and other networks to embed widgets and layouts and concatenate them into an embedding vector as the current state of the app under testing. According to the widgets on the page, UniRLTest generates all executable actions, i.e. interactions with the application, including various interactions with widgets, interactions with windows, and some special system events. Using deep reinforcement learning technology, UniRLTest assign a probability to each executable action according to the identified state vector. The probability is positively correlated to the expected cumulative reward that can be brought by the action at current state. The next action to be executed is randomly selected based on the probability in order to maximize the expected cumulative reward, while retaining randomization to prevent the algorithm from falling into a local optimal solution. In the design of the reward function of reinforcement learning , UniRLTest uses curiosity-driven strategy, and gives appropriate rewards according to the exploration rate of the new page and page transition counts after each page transition. With this reward system, it gradually loses its curiosity about the explored pages and increases its curiosity about the unexplored pages, so that it can explore more unknown states of the application under testing. Since the state recognition method is based on screenshots rather than the underlying layout provided by the framework, the tool has good platform versatility; at the same time, curiosity-driven deep reinforcement learning can also achieve a high coverage of the source code, so as to ensure sufficient amount of coverage of the target scene.

Based on the above technologies, this paper also implements a test input generation tool with the same name. The tool includes front end, server end and algorithm end. The algorithm end uses Tensorflow and Flask and other framework technologies to realize the above UniRLTest technology, while the server end uses ADB and ChromeDriver and other tools to implement the interaction with different platforms and the complete process of application testing, and the front end uses Vue.js framework and WebSocket technology to realize the configuration and monitoring interface of application testing. The interaction of the three ends is based on HTTP and WebSocket protocol.

We organized several experiments to verify the code coverage and event generation efficiency of UniRLTest. According to our experiments, it can be concluded that the code line coverage of the tool is higher than the existing traversal testing tools, and the event generation efficiency is lower, but its effect is not affected.

Keywords: Automated Test Input Generation, Deep Screenshot Understanding, Reinforcement Learning

目录

表 目 录 ····· **xi**

图 目 录 ····· **xiv**

第一章 引言 ····· **1**

1.1 背景介绍 ····· 1

1.2 国内外研究现状 ····· 3

1.3 本文的主要工作 ····· 4

1.4 本文的组织结构 ····· 5

第二章 相关技术概述 ····· **7**

2.1 Canny 边缘检测 ····· 7

2.2 光学字符识别 ····· 8

2.3 卷积神经网络 ····· 9

2.4 长短期记忆 ····· 11

2.5 深度强化学习 ····· 11

2.6 本章小结 ····· 13

第三章 需求分析与概要设计 ····· **15**

3.1 需求分析 ····· 15

3.1.1 用例分析 ····· 15

3.1.2 功能性需求 ····· 18

3.1.3 非功能性需求 ····· 19

3.2 概要设计 ····· 19

3.2.1 系统架构设计 ····· 19

3.2.2 系统 4+1 视图 ····· 22

3.2.3 系统接口设计 ····· 27

3.3 本章小结 ····· 28

第四章 技术设计与实现 ····· **29**
4.1 图像处理模块 ····· 29
4.1.1 控件边界框列表的获取 ····· 31
4.1.2 页面布局的生成 ····· 39
4.1.3 页面状态嵌入向量的生成 ····· 47
4.2 动作生成模块 ····· 51
4.2.1 事件生成 ····· 52
4.2.2 动作编码 ····· 53
4.3 执行策略模块 ····· 53
4.3.1 核心神经网络 ····· 55
4.3.2 随机化策略 ····· 56
4.4 即时奖励模块 ····· 58
4.4.1 奖励函数的设计 ····· 58
4.4.2 类图 ····· 58
4.4.3 活动图 ····· 60
4.4.4 页面相似度的计算 ····· 61
4.4.5 即时奖励的计算 ····· 63
4.5 回放记录模块 ····· 64
4.6 页面截屏模块、动作执行模块与前端界面 ····· 65
4.7 本章小结 ····· 67

第五章 系统测试与实验分析 ····· **69**
5.1 测试准备 ····· 69
5.2 功能测试 ····· 69
5.3 实验评估 ····· 73
5.3.1 实验设计 ····· 73
5.3.2 实验执行与结果分析 ····· 75
5.4 本章小结 ····· 78

第六章 总结与展望 ····· **79**
6.1 总结 ····· 79
6.2 展望 ····· 79

参考文献 **81**

简历与科研成果 **87**

致谢 **89**

表 目 录

3.1 多平台测试配置用例描述 ····· 16
3.2 多平台遍历测试用例描述 ····· 16
3.3 测试过程监控用例描述 ····· 17
3.4 测试数据下载用例描述 ····· 17
3.5 系统功能性需求列表 ····· 18
3.6 系统用例与功能性需求对应关系表 ····· 18
3.7 系统非功能性需求列表 ····· 19
3.8 前端与服务端交互的 HTTP 接口定义 ····· 27
3.9 服务端与前端交互的 WebSocket 数据格式定义 ····· 27
3.10 服务端与算法端的 HTTP 接口定义 ····· 28

5.1 算法端服务器环境信息 ····· 69
5.2 前、服务端计算机环境信息 ····· 70
5.3 安卓应用测试配置功能的测试用例设计 ····· 70
5.4 Web 应用测试配置功能的测试用例设计 ····· 71
5.5 安卓应用遍历测试功能的测试用例设计 ····· 71
5.6 Web 应用遍历测试功能的测试用例设计 ····· 72
5.7 设备屏幕监控功能的测试用例设计 ····· 72
5.8 测试日志监控功能的测试用例设计 ····· 72
5.9 下载日志文件功能的测试用例设计 ····· 73
5.10 功能测试结果 ····· 73
5.11 安卓应用列表 ····· 74
5.12 Web 应用列表 ····· 74
5.13 安卓应用 90min 覆盖率 ····· 76
5.14 安卓应用 500 个事件的覆盖率 ····· 77
5.15 Web 应用 90min 覆盖率 ····· 77
5.16 Web 应用 500 个事件的覆盖率 ····· 77
5.17 事件生成效率对比表（单位：个/min） ····· 78

图 目 录

1.1 安卓平台的布局结构 ····· 2
1.2 Web 平台的布局结构 ····· 3

2.1 VGG16 网络结构图 ····· 10
2.2 马尔可夫决策模型示意图 ····· 12

3.1 UniRLTest 系统用例图 ····· 15
3.2 UniRLTest 架构图 ····· 20
3.3 UniRLTest 整体流程图 ····· 21
3.4 UniRLTest 系统逻辑视图 ····· 23
3.5 UniRLTest 系统开发视图 ····· 25
3.6 UniRLTest 系统进程视图 ····· 26
3.7 UniRLTest 系统物理视图 ····· 27

4.1 图像处理模块的类图 ····· 29
4.2 图像处理模块的活动图 ····· 30
4.3 基于 Canny 算法的控件提取流程 ····· 31
4.4 基于 Canny 算子的控件提取算法 ····· 32
4.5 Canny 控件提取效果图 ····· 34
4.6 深度 OCR 核心算法 ····· 35
4.7 细粒度 OCR 为有效字符分组 ····· 36
4.8 Canny 边界框与 OCR 文本框的整合代码 ····· 38
4.9 计算三类 IOU 的函数代码 ····· 39
4.10 组行列效果图 ····· 40
4.11 初步分组的实现代码 ····· 41
4.12 细化分组的实现代码 ····· 42
4.13 合并存在包含关系控件组的实现代码 ····· 42
4.14 分离部分重叠控件组的实现代码 ····· 43

4.15 调整远距离控件组上下边界的实现代码 ····· 43
4.16 处理高度过小控件组的实现代码 ····· 44
4.17 组内分行的实现代码 ····· 45
4.18 merge_close_lines 函数的实现代码 ····· 46
4.19 行内分列的实现代码 ····· 46
4.20 控件边界框嵌入的实现代码 ····· 48
4.21 树的括号表示法示例 ····· 49
4.22 提取布局纯结构信息的实现代码 ····· 50
4.23 布局嵌入核心 LSTM 的实现代码 ····· 50
4.24 动作生成模块的类图 ····· 52
4.25 执行策略模块的类图 ····· 54
4.26 执行策略模块的活动图 ····· 55
4.27 策略模块核心神经网络的实现代码 ····· 56
4.28 Boltzmann 策略的实现代码 ····· 57
4.29 即时奖励模块的类图 ····· 59
4.30 即时奖励模块的活动图 ····· 60
4.31 计算控件相似度的实现代码 ····· 62
4.32 计算即时奖励的实现代码 ····· 63
4.33 回放记录模块获取训练数据的实现代码 ····· 65
4.34 前端测试配置界面 ····· 66
4.35 前端测试监控界面 ····· 66

第一章　引言

1.1　背景介绍

当今社会，各类软件不断问世，被越来越多的人所熟悉并使用，已经成为了人们生活中不可或缺的事物。而这些软件的质量参差不齐，低质量软件容易给用户带来时间上、精力上、乃至财富上、甚至是生命的损失。保证这些软件的质量至关重要。软件测试是软件生命周期的重要组成部分，是保障软件质量的重要环节。在软件测试中，界面测试又称 UI 测试，主要内容是测试页面中的内容是否正常展示，以及测试用户交互界面上各控件的交互功能是否正常。界面测试是软件测试中不可或缺的一环。

遍历测试是界面测试的重要测试方法之一。遍历测试的目标通常是在一段给定的时间内实现较高的覆盖率，以覆盖较多的目标场景。通过遍历测试，测试人员能快速测试软件在大量目标场景的展示与功能正确性，并快速发现软件的无响应或崩溃问题。

遍历测试方法的核心在于测试输入的生成，而测试输入的生成策略就是待测软件状态的探索策略，其中探索指的是发现待测软件的新状态。一般情况下，软件的一个状态指代该软件的一个特定的页面。因此探索策略的重点在于如何在一个给定页面下生成一个合适的事件（交互事件、系统事件等），以使其发生更有价值的状态变化。

在事件生成过程中，状态的识别是其中的第一步，状态识别的效果很大程度上影响着遍历测试的质量。目前的遍历测试方法在状态识别方面均依赖平台底层的布局文件以获取页面内容及布局结构。平台种类多样，常见的有安卓平台、Web 平台和 PC 平台等；不同平台的底层架构不同，因此其布局文件的要素也不同。图 1.1展示了安卓平台的布局结构，其控件树的每个节点均包含 index、text、resource-id、class 等属性；图 1.2展示了 Web 平台的布局结构，其每个节点可能包含 id、class、style 等属性，还可能有其它特殊属性（例如 type、data-adsbygoogle-status）。这些不同的属性导致了不同平台布局文件的解析方式不同，使得现有的遍历测试方法难以从一个平台移植到另一个平台上使用。这给不同平台的软件遍历测试带来了巨大的挑战。

除此以外，还存在一些控件内容被忽略的情况。例如安卓平台下部分页面会嵌入 H5 页面，而嵌入的 H5 页面内容无法被底层框架提取，导致这部分内容

被忽略。除了 H5 页面，所有平台均支持 Canvas 组件，而该组件的内容同样也不会被底层框架提取，进而导致该 Canvas 的内容被完全忽略。这些控件多数包含重要内容，这些内容一旦被忽略，测试工具便无法很好地对当前状态进行识别，进而生成低质量的事件，降低本次遍历测试的效果。

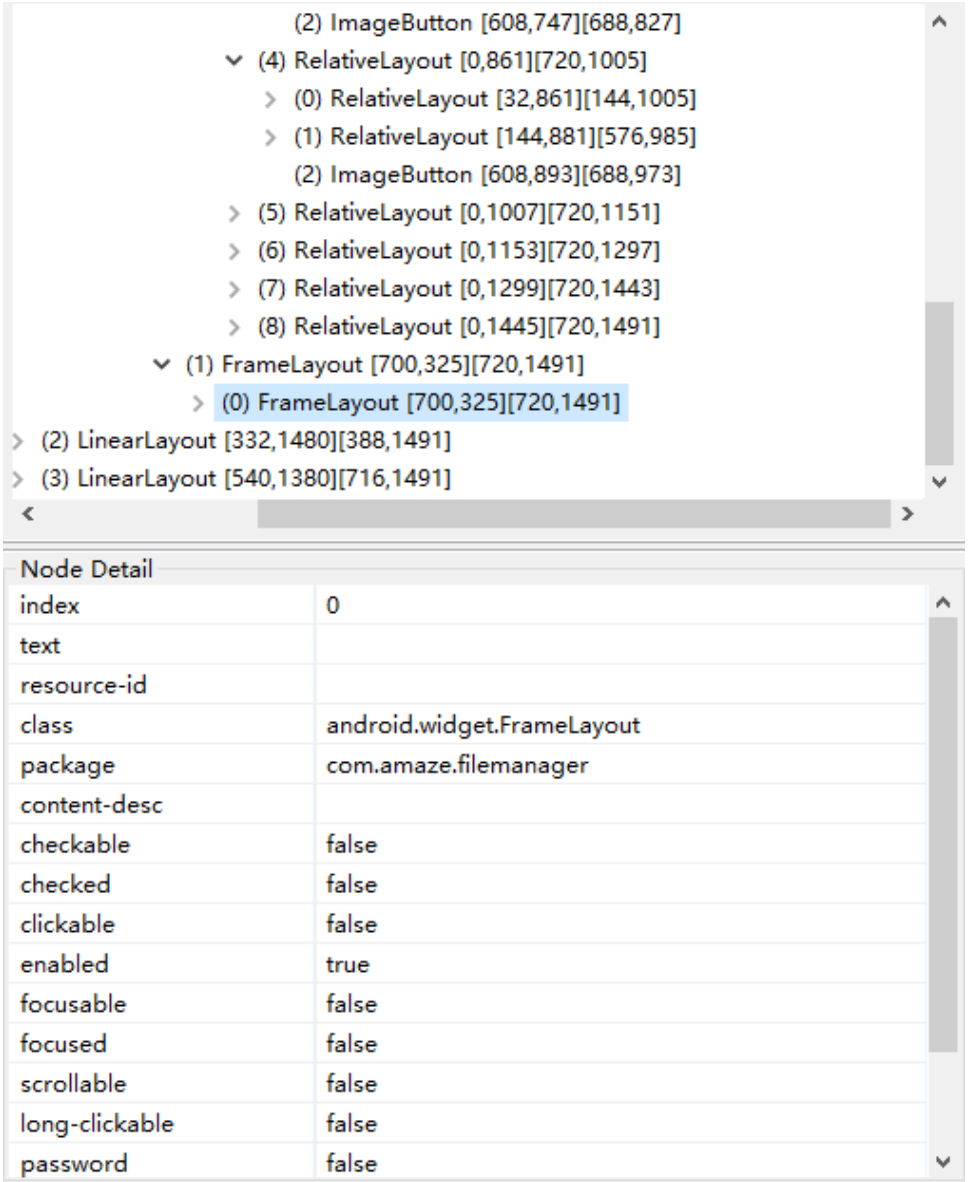


图 1.1: 安卓平台的布局结构

另一方面，各类平台都提供了简便的截屏接口以获取当前页面的截屏，且捕获的截屏质量较高。我们可以使用图像处理和深度学习技术来提取并识别截屏中的每个控件，并识别页面的布局结构，以实现对截屏的深度理解，并以此进

行待测软件的状态识别。这种状态识别方法摆脱了对平台底层的依赖，仅依赖截屏图像，从而实现了平台通用性，解决了平台移植问题。

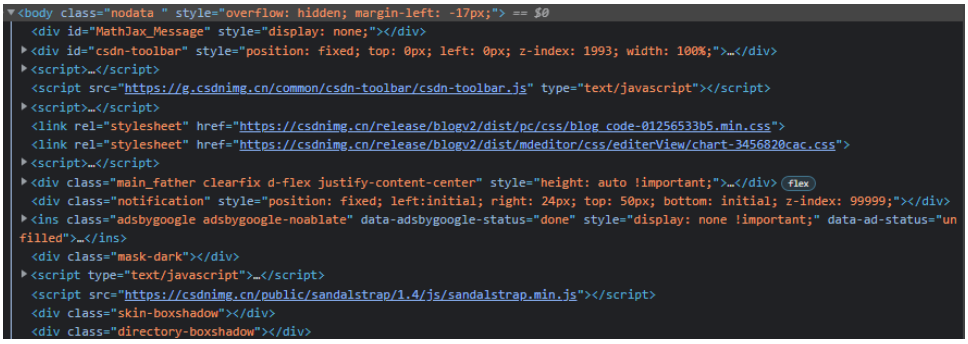


图 1.2: Web 平台的布局结构

基于以上考虑，本文提出了一个基于深度截屏理解强化学习的平台通用遍历测试方法 UniRLTest，并基于该方法实现了一个遍历测试工具。该方法利用深度学习和图像理解技术进行待测软件的状态识别，利用强化学习和好奇心驱动策略引导探索状态空间，从而对任意平台的软件进行遍历测试。

1.2 国内外研究现状

在遍历测试领域，有许多研究人员做了充足的研究工作，为我们的研究铺平了道路。

Monkey 是测试安卓应用程序最常用的工具，也是 Google 官方提供的安卓应用界面压力测试工具，由于嵌入在安卓平台内部、无需额外的安装工作而受到测试人员的欢迎。Monkey 采用了最基本的随机探索策略 [1]，即不论待测应用当前状态如何，按照一定时间间隔随机生成各类交互事件。测试人员可以配置 Monkey 所生成的交互事件的数量上限，而 Monkey 在生成的交互事件数量达到该上限后结束本次测试。随机探索策略的优点在于，由于不涉及对待测应用状态的分析与事件的挑选，其事件生成效率极高，因而成为了安卓应用界面压力测试的重要工具。虽然 Monkey 工具仅适用于安卓应用，但其随机探索策略则适用于任何平台，不具有平台局限性。随机探索策略的缺点则在于，对于一些由特定事件序列才能发现的页面或状态，由于触发特定事件序列的概率极低，它很难发现该类页面或状态。除此以外，随机探索策略不关心当前已覆盖的待测页面，因此总是会生成大量冗余事件，从而造成时间的浪费和覆盖率层面的效率降低。

Q-testing 是一种基于 Q-learning 的 Android 遍历测试工具 [2]。Q-testing 以

好奇心驱动策略 [3] 探索待测 Android 应用程序，利用内存集合记录部分以前访问过的状态，并引导系统对不熟悉的功能进行测试。Q-testing 中的状态识别模块是一个孪生 LSTM 神经网络 [4]，经过大量数据的训练，可以用于在功能场景的粒度上划分不同的状态。它可以确定 Q-testing 中的强化学习奖励，并帮助好奇心驱动策略有效地探索不同的功能。Q-testing 能够达到较高的代码覆盖率和错误检出率，但是由于其状态识别方法仍然基于安卓的布局文件，该方法同样无法摆脱对安卓平台的依赖。

WebExplor 是一个端到端的 Web 应用遍历测试工具 [5]，可以实现对 Web 应用程序的自适应探索。与 Q-testing 类似，WebExplor 采用好奇心驱动的强化学习来生成满足时间逻辑关系的高质量动作序列（即测试用例）。此外，WebExplor 在在线测试过程中增量构建了一个自动机，为进一步提高测试效率提供了高层次的指导。该工具也能够实现较高的代码覆盖率和错误检出率，但它同样依赖 Web 平台的布局文件，因此仅适用于 Web 平台的应用，不具有平台通用性。

另一方面，在图像理解领域，已有较为成熟的技术以实现对屏幕截图的理解。

Canny 算法是一种边缘检测算法 [6]，是截屏中控件提取算法的重要组成部分。该算法以高斯滤波来平滑图像，以非最大抑制技术来消除边缘误检，以双阈值决定可能的边界，最后利用滞后技术跟踪边界，最终完成图像的边缘检测工作。虽然该算法年代久远，但它仍然是当前最标准的边缘检测算法，被广泛运用于各项研究中。

卷积神经网络（CNN）是一种能够较好地处理图像的前馈神经网络 [7]。CNN 就其结构而言是由若干卷积层和一个全连通层组成，其间穿插着若干池化层，其中，卷积层所执行的运算为数学上的卷积运算，与常规的矩阵乘法运算不同。基于卷积运算的卷积层和相关的池化层为卷积神经网络赋能，使其能利用二维图像的局部相关性，提取出其中的复杂特征信息，这种特征信息与传统的 SIFT [8]、SURF [9] 乃至空间金字塔 [10] 相比，往往能更好的反映图像的特征，因此在图像分类、物体识别等方面效果显著 [11]。正是因此，CNN 对于截屏控件的提取与识别至关重要。

1.3 本文的主要工作

为了解决当前遍历测试技术在状态识别方面的平台局限性，本文提出了基于深度图像理解强化学习的通用遍历测试技术。该技术不依赖平台提供的底层布局，而是基于对应用截屏的理解进行状态识别，并应用深度强化学习的方法实现对待测应用的遍历测试。该技术整体分为以下 5 个模块：

1. 图像处理模块：利用截屏处理技术处理并分析捕获到的实时应用截屏，识别其中的所有控件与布局，并以此生成状态向量表示应用当前状态。

2. 动作生成模块：根据当前截屏中所有的控件生成各类交互动作和系统动作，并对所有动作进行嵌入，生成相应的嵌入向量。

3. 执行策略模块：为每一个 < 状态，动作 > 对计算一个价值，以表示在该状态下执行该动作能带来的预期累计奖励，这一部分使用了一个深度神经网络来拟合该函数；基于这些 < 状态，动作 > 对的价值，挑选一个合适的动作作为下一个待执行的动作。

4. 即时奖励模块：使用好奇心驱动，根据页面探索度和页面转移次数对一次状态转移给出一个合适的奖励，以引导系统能探索待测应用的更多未知状态。

5. 回放记录模块：记录每一次状态转移及其对应的即时奖励，为执行策略模块中的深度神经网络提供训练数据。

本技术中遍历测试的完整过程为：捕获应用截屏，接着图像处理模块分析截屏，接着动作生成模块生成动作集合，然后执行策略模块利用回放记录模块的数据和即时奖励模块的奖励函数训练神经网络，然后执行策略模块挑选合适动作，最后执行动作，待测应用发生页面变化，捕获应用截屏，重复上述循环直至达到一定时间或步数。

本文还基于上述技术实现了一个相应地遍历测试工具，分为算法端、服务端和前端。其中算法端由上述五大模块构成，实现了基于图像理解强化学习的通用遍历测试技术的核心模块；服务端则由页面截屏模块和动作执行模块组成，通过算法端提供的算法实现了遍历测试的全流程；最后前端则负责展示遍历测试的配置以及测试过程的监控。

1.4 本文的组织结构

本文主要介绍了基于深度截屏理解强化学习的平台通用遍历测试技术（UniRL-Test）及相关工具的实现。该技术的主要目标是解决当前遍历测试技术普遍存在的平台局限性，基于用户能直接观察到的界面实现安卓和 Web 平台通用的遍历测试。

第一章：概述和前言部分，主要介绍技术背景、当前遍历测试的研究现状，并描述了本文的主要工作。

第二章：主要介绍 UniRLTest 所涉及的相关理论知识和理论研究。

第三章：需求分析和概要设计，分析 UniRLTest 技术与工具的功能性和非功能性需求，给出相关的图表，并对其进行整体框架与模块的划分。

第四章：对 UniRLTest 技术与工具的各模块进行详细设计，结合其概要设计的内容，详细描述该工具的三个端七大模块的设计与实现，并给出关键的流程图、类图、顺序图和相关代码等。

第五章：介绍针对 UniRLTest 的功能性测试与有效性实验设计与验证过程，以图表的形式给出测试与实验的结果并对此加以详细分析。

第六章：总结 UniRLTest 的优点与不足，并指出该技术未来的扩展和发展方向。

第二章　相关技术概述

2.1　Canny 边缘检测

Canny 边缘检测技术是 1986 年由 John Canny 首次提出 [6] 的一种用于检测图像中的边缘特征点的技术，其主要功能是从图像中提取出一定程度的结构信息，这些信息可被表示为图像中各个物体的边缘线，因此被称为边缘检测技术。边缘信息属于图像的低层结构信息，该信息能在很大程度上左右高层语义信息提取的准确程度，因此边缘检测技术是本技术中控件识别工作的基础，目前已被广泛应用于各种计算机视觉系统 [12] [13] [14]。

即使对于不同类型的物体和边缘，边缘检测技术通常都有着相对不变的标准。一般而言，边缘检测算法存在以下四个标准：

1. 低误报率，即检测出的边缘特征点必须尽可能是真正的边缘点。
2. 精确性，即由检测算法检测出来的边缘必须位于真实边缘所对应范围的中心。
3. 不重复性，即图像中的任何真实边缘都应当被标记至多一次，不可出现多条检测出来的边缘对应同一个真实边缘。
4. 抗干扰性，即尽可能降低图像中的噪声对边缘识别准确率的影响。

Canny 边缘检测算法通过以下五个步骤实现对图像边缘的检测，并同时满足以上四个标准：

1. 平滑图像。该算法利用高斯滤波器实现对图像的平滑化操作。一般而言，平滑化是为了尽可能减少图像中的噪点对边缘检测效果的影响。以高斯滤波器进行平滑化，可以有效缓解因图像中的噪点而产生的边缘检测效果的下滑问题。
2. 计算梯度。对图像中每个像素点计算其梯度，包括该梯度的强度和方向。由于图像中往往包含了各个方向的边缘，因此 Canny 算法中有四个不同的算子以分别检测图像中的水平、垂直和对角这四个方向的边缘。

3. 抑制边缘。常规的边缘检测技术往往会有杂散响应问题，即检测到了多个边缘点却只对应同一个真实边缘，不符合上述标准中的第三条。为了消除上述杂散响应，Canny 算法应用了一种非极大值（Non-Maximum Suppression）抑制的边缘稀疏技术，该技术可以细化粗边缘，通过保留极大值而将其它所有梯度值抑制为 0。

4. 双阈值（Double-Threshold）检测。即便有高斯滤波和非极大值抑制的辅助，仍然会有一些假边缘被识别出来，这类假边缘通常来自特殊噪点或者是渐变颜色。参考上文第一条标准，为了去除这类假边缘，Canny 算法设计了高低双阈值，将图像中梯度低于低阈值的像素作为假边缘直接去除，对图像中梯度高于高阈值的像素作为强边缘（即真实边缘）而保留；如果一个像素的梯度值介于高低阈值之间，则将其看作弱边缘像素，等待后续处理。该算法中阈值可以由参数指定，通常而言需要根据输入图像的内容特点来确定。

5. 处理弱边缘。对于弱边缘像素，它们有两种可能的来源：从真实边缘提取的或者是因图像噪点或渐变颜色引起的。为了尽可能减少误报率和提升召回率，由噪点或渐变颜色引起的弱边缘像素需要被去除，因为它们不是真实边缘。通常，如果一个弱边缘像素能够连接到一个强边缘像素，则可以认为该弱边缘像素来自于该强边缘像素，因而这样的边缘点会被保留；其余未能连接一个强边缘像素的均会被去除。通常情况下，如果一个弱边缘像素的周围八个邻域像素中，有一个或更多强边缘像素，就可以认为二者存在连接，即该弱边缘像素会被保留。

在当下较为常用的边缘检测方法中，Canny 边缘检测算法定义较为严格，检测效果较为可靠，能够满足上述边缘检测技术的四大通用标准，加之该算法逻辑清晰、实现简单，因此，该算法是一种当前较为常用的边缘检测算法和图像处理算法。在本文提出的基于图像理解强化学习的通用遍历测试技术中，同样利用了 Canny 边缘检测技术来实现控件识别。

2.2 光学字符识别

光学字符识别（OCR）是一种提取图像中文本内容的技术，主要利用光学技术和计算机技术，实现对手写或打印的字符的识别，并将其转换为计算机文本 [15]。对于富含文本内容的图像而言，光学字符识别能够提取其中的文本信息以便后续处理，这是对这类图像进行理解的基础。

光学字符识别算法主要由以下 3 个步骤组成 [16]：

1. 预处理：包括二值化、噪声去除和倾斜矫正等步骤。其中二值化可以将输入的三通道图像根据前景色与背景色处理成黑白图像，即二值化图；噪声去除则是对不同类型的图像，根据不同噪声的特征进行去噪；倾斜矫正则是对包含倾斜文本的图像进行矫正。

2. 字符识别：包含版面分析、字符切割、文字特征抽取和对比识别等步骤。其中版面分析则是将图片分段、分行，然而在实际操作中，由于文档的多样性、复杂性，目前还没有一个固定的、最优的切割模型；字符切割则是为了处理部分场景下由于拍照、书写条件的限制而导致的字符粘连、断笔；文字特征抽取是 OCR 算法的核心，直接影响着识别的结果，通常提取统计的特征或是结构的特征；在对比识别时，将提取出的特征与特征数据库中的数据进行比对得到结果。

3. 后处理：包含了字词识别结果的纠正工作。字符识别的结果并不是完全准确的，常常需要根据上下文和相关词库进行纠正。

光学字符识别技术目前已经较为成熟，已被广泛运用于各行业的卡票证件的识别和手写识别等领域 [17] [18] [19]。

2.3 卷积神经网络

卷积神经网络（CNN）是一种基于卷积操作的前馈神经网络 [20]，能够在处理图像信息时表现出较好的效果 [21]。卷积神经网络能够很好地降低图像的数据量并提取图像的高层特征以便进一步处理，是图像理解工作的基础。

典型的 CNN 由以下三个部分组成 [22]：

- 卷积层：负责提取图像中的局部特征。卷积层通过在输入图像上滑动不同的卷积核并执行卷积运算来生成一组平行的特征图。

- 池化层：负责大幅降低参数的量级。池化是一种非线性的降采样方法，在各类池化函数中，最大池化最为常用，其具体方法是将卷积后的特征图划分出若干矩形区域作为池化视野，取每个池化视野中的最大值作为该视野的池化输出，以此形成数据量大幅减小的特征图。池化层具有减小数据量而保留图像特征的能力，因此池化后神经网络的参数数量和整体计算量会被大幅削减，从而提高训练与预测的效率。此外，由于削减了神经元中所包含的数据，池化操作能够在一定程度上缓解网络的过拟合问题。

- 全连接层：负责输出想要的结果。与一般的神经网络一样，此处的全连接层经过训练可以根据提取的特征推理出想要的结果。

具有代表性的CNN主要有AlexNet [23]、VGG [24]、Inception系列 [25] [26] [27]和ResNet [28]。AlexNet突破性地使用ReLU非线性激活函数、多GPU训练等手段加速了模型的训练，并采用数据增强和Dropout层来缓解过拟合问题。VGG模型通过对AlexNet网络加深以提取更好的图片特征，尽管其带来了参数过多、内存占用量过大的缺点。Inception系列模型的特点则是模块化、卷积核的并行合并和对1×1卷积核的使用，在加深网络和提升了性能的同时保持了相对较小的计算量。ResNet则将学习目标由学习一个完整的输出转变为学习输出与输入的差别，即残差，以此解决深层神经网络的退化问题，同时提升模型的训练速度和训练效果。

这里重点介绍在UniRLTest技术中使用的VGG16网络。该网络是VGG系列神经网络之一，该网络的特点在于结构简单，卷积核较小而网络较深，图像特征提取效果较好。图2.1为VGG16网络的结构图，其中Conv代表卷积模块，P代表最大池化模块，FC代表全连接模块；该网络的输入为224×224×3尺寸的图像；在卷积模块中，第一行的“3x3”表示卷积核大小，第三行的数字（例如“64”）代表卷积核数量，第四行（例如“x2”）代表卷积层的数量；池化模块中“/2”表示该池化层将特征图尺寸减半，最后一个池化层的输出特征图尺寸为7×7；全连接层模块的第二行（例如“4096”）代表输出维度数，第三行（例如“x2”）代表全连接层的数量；VGG16网络最终输出为一个1000维的特征向量。

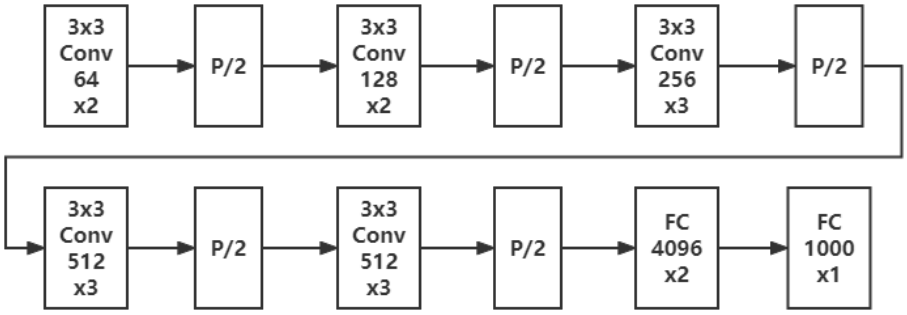


图2.1: VGG16网络结构图

由于卷积神经网络在图像处理领域的出色表现，目前已被广泛运用于影像识别和视频分析领域 [29] [30]。在深度截屏理解中，卷积神经网络同样也发挥着重要作用。

2.4 长短期记忆

普通的前馈神经网络（FNN）一旦结构固定，其输入与输出向量的维度便被固定了，无法改变；而循环神经网络（RNN）不受此限制。RNN 能够对任意长度的输入进行处理，也能产生任意长度的输出。除此以外，RNN 结构中每一层的输入均包含了上一层的输出和本层上一次的输出，因此能够对序列数据具有一定的记忆能力，从而使其在处理具有上下文联系的序列数据时表现出较好的效果 [31]。

长短期记忆（LSTM）是循环神经网络的一个重要变种 [32]。RNN 虽然能较好地处理序列数据，但是当输入序列较长时，会因网络过深导致梯度爆炸或梯度消失问题，使其难以训练，降低最终的效果。LSTM 正是为了解决该问题，在保留 RNN 的整体结构时，对常规 RNN 的每个神经元进行修改，通过引入遗忘门神经元、输入门神经元和输出门神经元来防止梯度的爆炸与消失。在 LSTM 中，每个门神经元本质上是一个全连接层，其中遗忘门控制着每一层对上一时刻所记忆内容的保留的量，输入门控制着每一层对当前时刻输入数据的记忆的量，最后的输出门则控制着当前时刻记忆内容的输出的量。

由于 LSTM 在处理较长的输入序列时的优秀表现，目前已被广泛运用于机器控制 [33]、自然语言处理 [34]、语音识别 [35] 等领域。

2.5 深度强化学习

强化学习（RL）是一种根据环境反馈而作出决策改变行为以获取最大化预期收益的机器学习算法 [36]，其关注点在于对未知领域的探索和对已知知识的利用之间的平衡。强化学习的优点主要有以下三点：

- 强化学习能够根据长时间的行为序列进行推理；
- 对于包含长期反馈的问题比短期反馈的表现更好；
- 可以在与环境的互动中不断优化行为。

强化学习可被建模为一个马尔可夫决策模型（MDP） [37]，它包含：环境状态集合 S，动作集合 A，状态转换规则（状态转移概率矩阵）P，以及状态转移后的奖励规则（即奖励函数）R。其基本流程为：解释者根据自己对环境当前状态的观察给出上一个动作的奖励，执行者则根据该奖励进一步认识环境，并基于新的认知和当前环境状态做出下一个动作的决策并执行，环境由于新的动作而发生状态转移，接着再由解释者给出该动作的奖励，如此往复，如图2.2所

示。在这一过程中，执行者不断地调整自己对环境的认知，不断减少其与真实环境的偏差，以便做出尽可能正确的决策。遍历测试同样可以被建模为一个 MDP，其中环境状态集合通常为待测应用的页面集合，动作集合则是在给定页面上可执行的交互操作集合，状态转换规则即为待测应用的页面切换逻辑，最后，奖励规则是一个完全自定义的函数。因此，我们可以将强化学习应用于遍历测试中，通过制定良好的奖励函数实现高效的遍历测试。

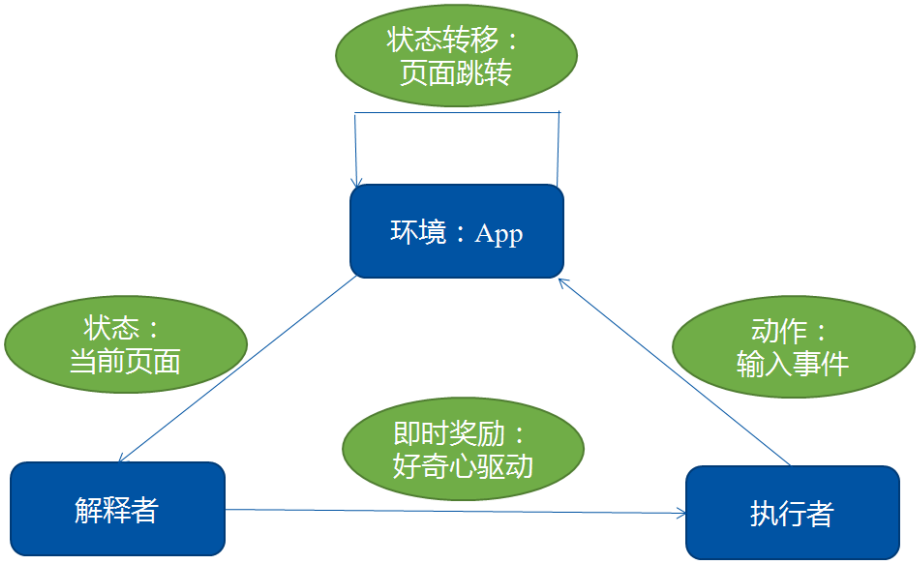


图 2.2: 马尔可夫决策模型示意图

从实现层面而言，强化学习有两种实现算法：基于模型的算法和无模型的算法。基于模型的算法往往需要对 MDP 的四要素 <S,A,P,R> 完全预知，典型的有动态规划法；而无模型的算法则是在探索过程中逐步获取对 MDP 四要素的认知，典型的有 Monte-Carlo 算法和 Q-learning 算法，其中 Monte-Carlo 算法的原理是通过大量的尝试来使均值不断逼近数学期望 [38]，因而效率低下，Q-learning 则是通过模仿神经网络的方式不断调整系统认知与实际环境的差距，以此引导决策 [39]。

在实际应用问题中，通常情况下输入数据大多维数较大，典型的输入数据有图像和声音等，即便经过神经网络的处理也很难将数据限制到低维空间中；除此以外，实际情况中的状态空间庞大，状态数量较多。传统的强化学习算法不论是 Monte-Carlo 还是 Q-learning 在处理如此巨大的数据时均消耗过大、效率低下，由此，深度强化学习应运而生 [40]。深度强化学习是深度学习和强化学习的结合体，它结合了深度学习对大量数据的特征提取能力和强化学习在复杂环境下的强大决策能力 [41]，为强化学习技术赋能，使其有能力处理高维数据和庞大

的状态空间，使其变得更加实用，能够解决现实场景下的复杂问题。

深度强化学习的典型代表为深度 Q 网络算法（以下简称 DQN）[42]，其整体逻辑与 Q-learning 相似，但 DQN 利用深度卷积神经网络来拟合 Q-learning 算法中的 Q 函数。该网络可以根据输入的图像生成在当前状态下执行各动作的价值，即 Q 值。DQN 的损失函数定义为 Q-learning 算法中 Q 的更新值与 DQN 的输出 Q 值的均方误差，训练样本则是通过不停地执行动作动态产生的。为了防止算法陷入局部最优解和最小化训练样本之间的相关性，DQN 在训练时采用回放记录机制，在所有执行记录中随机挑选部分样本作为训练数据。

2.6 本章小结

本章介绍了深度截屏理解强化学习的平台通用遍历测试技术中所使用的五个关键技术：Canny 边缘检测、光学字符识别、卷积神经网络、长短期记忆和深度强化学习技术的概念与特点，并阐释了使用这些技术的原因与好处。

第三章　需求分析与概要设计

本章根据软件测试领域的遍历测试需求，分析了基于图像理解强化学习的通用遍历测试技术的系统需求，接着基于这些需求进行了系统的概要设计。本章首先从系统用例开始分析，逐步得到该系统的功能性需求和非功能性需求；然后根据这些需求，对本系统进行架构设计，并给出架构设计的 4+1 视图，包含逻辑视图、开发视图、进程视图、物理视图和场景视图，最后给出系统的接口设计。

3.1　需求分析

3.1.1　用例分析

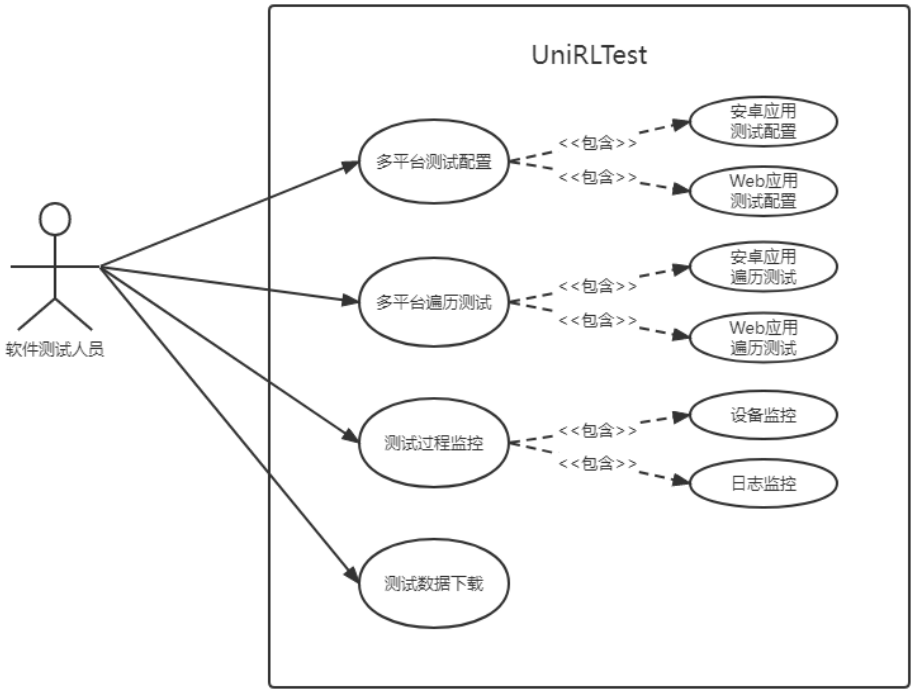


图 3.1: UniRLTest 系统用例图

图 3.1是本系统的用例图。软件测试人员是本系统的使用者，在遍历测试方面，测试人员有多平台测试配置、多平台遍历测试、测试过程监控和测试数据下载四个用例，其中多平台测试配置和多平台遍历测试两个用例可以根据平台进一步拆分为针对安卓平台和 Web 平台应用的测试配置与遍历测试子用例，而测

试过程监控可以被拆分为设备监控和日志监控两大子用例。下面针对每个用例进行详细描述。

多平台测试配置是进行遍历测试的前提，在这个用例中，软件测试人员需要选择待测应用所属的平台，并根据不同平台填入不同的遍历测试所需的参数，而系统则负责对这些参数做记录，使接下来的测试按照给定参数顺利进行。当软件测试人员填入的参数格式不正确或存在参数缺失时，系统会做出响应的提示并要求重新填写或补全参数。多平台测试配置用例的描述如表 3.1所示。

表 3.1: 多平台测试配置用例描述

ID	**UC1**
名称	多平台测试配置
参与者	软件测试人员
触发条件	测试人员准备开始测试
前置条件	无
后置条件	无
优先级	高
正常流程	1. 测试人员打开配置页面 2. 测试人员选择待测应用所属的平台 3. 测试人员填入相应平台的应用遍历测试所需的所有参数 4. 系统记录相关参数
异常流程	3a. 参数格式不正确 1. 系统显示参数格式不正确提示信息，要求用户重新填写 3b. 缺少参数 1. 系统显示缺少参数提示，要求用户补全参数

表 3.2: 多平台遍历测试用例描述

ID	**UC2**
名称	多平台遍历测试
参与者	软件测试人员
触发条件	测试人员准备开始遍历测试
前置条件	测试参数完整且正确
后置条件	系统连接设备并开始测试
优先级	高
正常流程	1. 测试人员点击开始测试按钮 2. 系统检查确认参数无误后操作相应设备进行遍历测试 3. 系统跳转到测试详情页面，将测试详情展示给测试人员
异常流程	无

多平台遍历测试是本系统的核心功能，也是对软件测试人员而言最为重要

的功能。当软件测试人员准备开始遍历测试时，点击配置页面的开始测试按钮，而系统在确认参数完整无误后即可开始连接相关测试设备并对待测应用进行遍历测试。在遍历测试开始后，系统前端会跳转到测试详情页面，向软件测试人员展示本次测试的详细且实时的信息。多平台遍历测试的用例描述如表 3.2所示。

测试过程监控是在遍历测试开始后便于测试人员对本次测试状况与进度建立直观了解的重要功能。测试过程监控主要包含对测试设备的实时监控和对测试状态的实时监控。具体而言，在测试人员开始了一次任意平台的遍历测试后，系统会连接相关测试设备开始测试，并将测试设备的截屏实时展示在测试详情页面，同时还会把本次测试的日志信息打印在测试详情页面，日志中包含了每个对测试应用与测试设备的操作。测试过程监控的用例描述如表 3.3所示。

表 3.3: 测试过程监控用例描述

ID	**UC3**
名称	测试过程监控
参与者	软件测试人员
触发条件	遍历测试已开始
前置条件	系统与测试设备已连接
后置条件	无
优先级	高
正常流程	1. 软件测试人员开始测试 2. 系统将当前测试设备的截屏和测试日志实时输出在测试详情页面
异常流程	无

表 3.4: 测试数据下载用例描述

ID	**UC4**
名称	测试数据下载
参与者	软件测试人员
触发条件	测试人员点击下载按钮
前置条件	遍历测试已结束
后置条件	将测试数据传输到测试人员的计算机上
优先级	中
正常流程	1. 遍历测试结束后，系统提示可以下载测试日志 2. 软件测试人员点击日志下载按钮 3. 系统将本次测试的日志整理后传输给测试人员的本地
异常流程	无

测试数据下载是为了便于软件测试人员对本次测试进行复盘的一个功能。在一次遍历测试结束后，系统会提示软件测试人员可以进行测试日志的下载，

相关下载按钮会被激活。软件测试人员此时可以点击该下载按钮将本次测试的日志下载到本地，该日志中记录了本次测试的配置信息以及对测试设备与待测应用所作的每一个操作。测试数据下载的用例描述如表 3.4所示。

3.1.2 功能性需求

根据上述系统用例分析，我们可以得到本系统的功能性需求。本系统的功能性需求如表 3.5所示。在进行遍历测试前，本系统首先要允许软件测试人员填写本次遍历测试的配置参数，不同平台的配置参数种类和数量不尽相同，需要分开填写。本系统的核心功能需求为对安卓平台和 Web 平台这两大平台的应用进行遍历测试。在遍历测试进行过程中，还需要将测试设备的屏幕实时展现给测试人员，并将测试日志实时打印在界面上，以便于测试人员对测试的每一步有所了解。在测试结束后，系统还需要提供测试日志的下载功能，将本次测试日志提供给软件测试人员以便其对本次遍历测试过程进行复盘。

表 3.5: 系统功能性需求列表

需求 ID	需求名称	需求描述
R1	安卓应用测试配置	对于安卓应用进行遍历测试前对本次遍历测试填写配置参数
R2	Web 应用测试配置	对于 Web 应用进行遍历测试前对本次遍历测试填写配置参数
R3	安卓应用遍历测试	对安卓应用根据给定配置进行遍历测试
R4	Web 应用遍历测试	对 Web 应用根据给定配置进行遍历测试
R5	设备屏幕监控	在遍历测试过程中监控安卓或 Web 设备的屏幕，将其实时截屏展示给用户
R6	测试日志监控	在遍历测试过程中监控测试日志，将测试日志实时打印到前端界面上展示给用户
R7	下载日志文件	在遍历测试结束后将本次测试的日志文件传输给用户

表 3.6: 系统用例与功能性需求对应关系表

用例 ID	用例名称	对应需求编号
UC1	多平台测试配置	R1,R2
UC2	多平台遍历测试	R3,R4
UC3	测试过程监控	R5, R6
UC4	测试数据下载	R7

表 3.6为系统用例与功能性需求的对应关系表，需求 R1 和 R2 对应系统用例 UC1: 多平台测试配置，需求 R3 和 R4 对应系统用例 UC2: 多平台遍历测试，需求 R5 和 R6 对应系统用例 UC3: 测试过程监控，需求 R7 对应系统用例 UC4: 测

试数据下载。所有系统用例均有若干功能性需求与之对应，所有功能性需求有一个系统用例与之对应。

3.1.3 非功能性需求

本系统的非功能性需求如表 3.7所示，包含稳定性、可用性、可扩展性、性能、兼容性和可伸缩性 6 个方面的非功能性需求。

表 3.7: 系统非功能性需求列表

需求名称	需求描述
稳定性	本系统应当能在短时间高流量的情况下正常稳定运行，不发生崩溃或无响应问题，对过量部分的请求采用排队机制逐个响应。
可用性	本系统应当在流量正常的情况下持续正常运行，对于来自用户或环境的意外情况有正确的应对措施。若发生诸如断电、服务器重启等难以控制的突发情况导致服务下线，要求能在服务器恢复后 60s 内恢复服务。
可扩展性	本系统应当对不同平台的测试配置与相关测试数据的传输进行抽象，并提供相关的扩展接口以便未来支持更多平台和更多类型的测试数据。
性能	在较为稳定的网络状况下，系统接口的返回时间应在 1000ms 以内。
兼容性	本系统应当支持将部署到各大主流操作系统的服务器上，包括 Unix、Linux、Window 和 MacOS 服务器。
可伸缩性	若当前服务器由于硬件配置原因难以进行多个应用遍历测试，应能快速切换服务器部署以响应其它应用的遍历测试请求。

3.2 概要设计

3.2.1 系统架构设计

本系统基于对待测应用截屏的深度理解，利用深度强化学习技术，实现平台通用的遍历测试。图 3.2是基于深度截屏理解强化学习的平台通用遍历测试系统的架构图。该工具分为前端、服务端和算法端三个部分，其中前端主要负责接收软件测试人员的测试配置，并将遍历测试的实时信息展现出来；服务端台主要负责与不同的平台交互，包括获取该平台当前的屏幕截图以及在该平台上执行一个指定的动作；算法端主要负责生成交互动作并不断优化动作选择策略，是该系统的核心所在。

该工具的前端是一个 Web 应用，采用 http 协议向服务端发送信息，而采用 websocket 协议接收来自服务端的实时测试信息。前端主要可分为两个界面：

- 测试配置界面提供给软件测试人员输入遍历测试的配置信息，包括待测应用信息、测试设备信息和测试时间等；

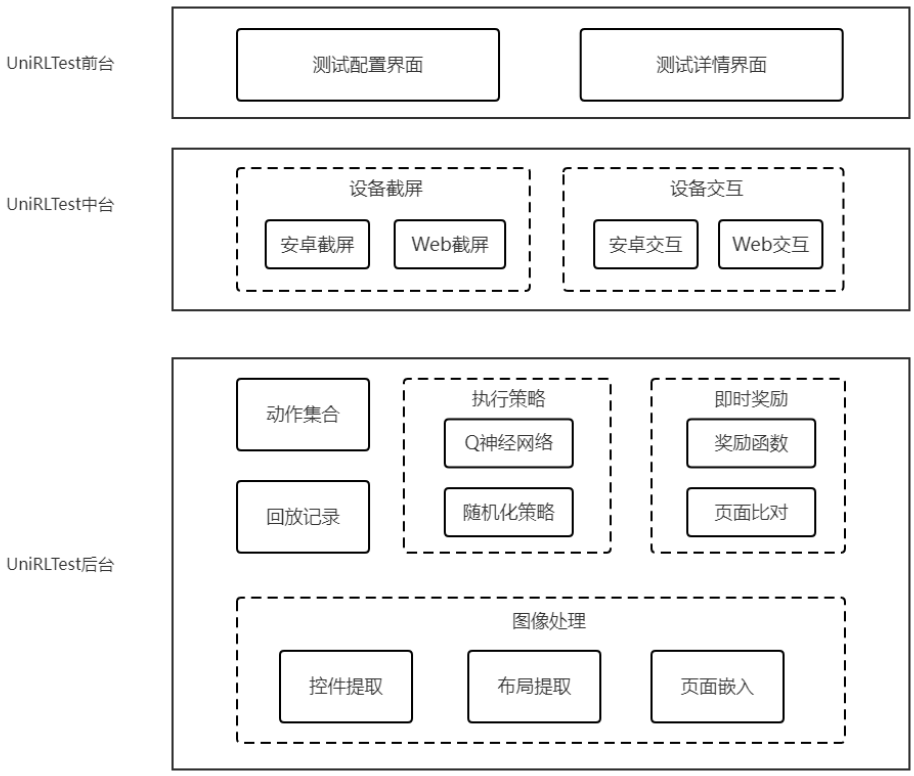


图 3.2: UniRLTest 架构图

- 测试详情页面则向软件测试人员展示本次测试中测试设备的实时屏幕和系统的实时测试日志。

该工具的目标平台目前有两个：安卓和 Web 平台。而该工具的服务端部署于计算机上而非移动设备，因此对于安卓平台的遍历测试，需要先将安卓设备连接至部署了该工具服务端的计算机上。服务端分为以下两个模块：

- 页面截屏模块：负责在目标平台上获取待测应用的当前屏幕截图。
- 动作执行模块：负责在目标平台上执行各类指定操作。

该工具的算法端是本技术的核心，也是本文的重点所在。算法端实现的动作决策无关平台，真正实现了平台通用性。该工具的算法端主要由以下 5 个模块组成：

- 图像处理模块：负责从截屏图像中提取控件和布局信息，提供给其他模块使用，是算法端其他模块的基础；除了提取相关信息，该模块还负责对当前页面进行嵌入，生成一个嵌入向量以表征当前页面状态。

- 动作集合模块：负责根据当前页面截屏生成所有可执行的动作，并提供该动作集合每个动作的嵌入向量。
- 执行策略模块：负责根据当前页面状态，从动作集合中选出一个最优动作。所谓最优动作，即在系统对待测应用当前的认知下能最大化累计奖励的动作。在挑选最优动作时，本模块还引入了一定程度的随机化策略已解决强化学习中的探索-利用困境。
- 即时奖励模块：负责根据页面转移以及新页面的探索度赋予本次页面转移一个合适的奖励。赋予即时奖励的目的在于激励系统进一步探索待测应用的其他未知状态。本模块还有一个页面比对功能以辅助即时奖励的计算。
- 回放记录模块：负责存储所有状态转移及其奖励并为策略模块的神经网络提供训练数据。

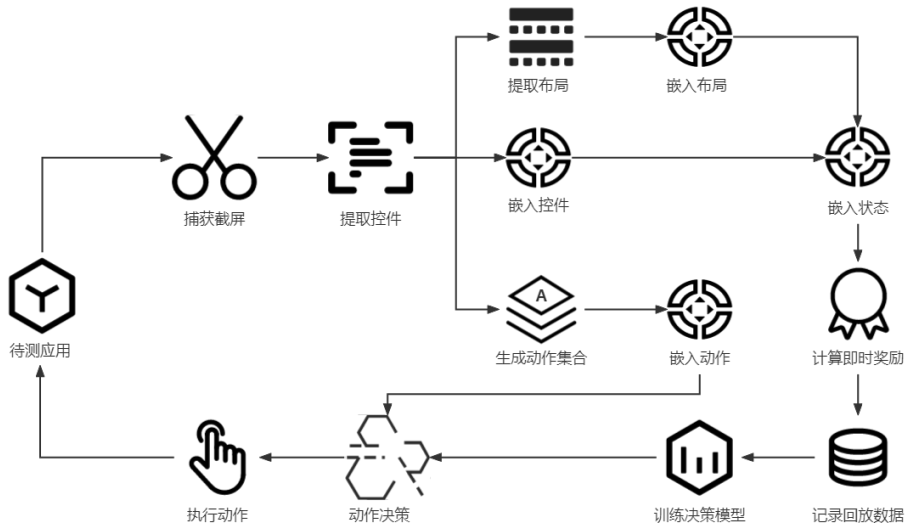


图 3.3: UniRLTest 整体流程图

图 3.3为 UniRLTest 的整体流程图，整体采用好奇心驱动探索。首先捕获待测应用当前的屏幕截图，接着利用控件提取技术将截屏中所有控件的边界框提取出来，其中包含了每一个控件的位置和大小信息。这些提取出的控件会被同时进行以下 3 步处理：一是根据控件边界框识别这些控件的布局，形成一棵控件树，进而将该控件树进行嵌入，形成布局嵌入向量；二是将所有控件整合入一个嵌入向量中，形成该页面的控件嵌入向量；三是针对每一个控件，生成所有可执行的动作，形成一个动作集合，并对这些动作进行嵌入，生成每一个动作的嵌入

向量。接着，将上述布局嵌入向量和控件嵌入向量整合入一个向量中，形成代表了该页面的状态嵌入向量。系统会依据新页面状态与本次页面转移（即从上一个页面通过一个动作转移至当前页面）计算一个对应的即时奖励，该奖励会与本次页面转移一同存储至回放记录中。接着，系统会从回放记录中选择一些回放数据，基于这些回放数据生成决策模型所需的训练数据，并以此训练系统的决策模型。训练完成后，根据当前的状态嵌入向量与所有动作嵌入向量，应用最新的模型，完成动作决策，选择出一个当前条件下的最佳动作并于目标平台上执行，使其发生状态转移，进入新的页面或发生页面元素的改变。接着继续从捕获截屏开始，重复上述流程直至结束。结束条件通常是达到了一个给定的时间限制。

3.2.2 系统 4+1 视图

系统的 4+1 视图是一种描述软件体系结构的可视化方法，该方法的核心为 5 张不同角度的视图：逻辑视图、开发视图、进程视图、物理视图和场景视图，顾名思义，它们分别时从一个系统架构的逻辑、开发、进程、物理和用户场景五大角度对系统展开的描述。采用 4+1 视图能让不同的系统参与人员从多个角度对系统有一个较为完整的、全局的、清晰的认知，因此本文采用 4+1 架构进一步描述本系统的体系结构设计。在上述 5 个视图中，场景视图已经由上一节中的系统用例图及相关解释加以详细描述了，因此在本节，仅介绍本系统的逻辑、开发、进程和物理这四大视图，不对本系统的场景视图多加赘述。

本系统的逻辑视图如图 3.4所示，该图展示了本系统的遍历测试主要逻辑。遍历测试逻辑主要涉及以下几个类：

- TestLauncher 类。该类为遍历测试启动类，提供一个 launch(config) 方法以启动遍历测试，其参数 config 为本次测试的各项配置。

- ScreenCapturer 类。该类为设备截屏器，负责从测试设备上获取截屏，并将其导出至指定路径下。

- ActionExecutor 类。该类为动作执行器，负责执行所有与设备的交互操作，如点击、输入、滑动等。

- StrategyService 类。该类为策略服务，负责根据当前页面状态生成下一个需要被执行的动作，该策略需要实现最大化探索待测应用状态空间的目的。该类的实现主要依赖 PageProcessor 类、 ActionGenerator 类和 RandomizationStrategy 类。

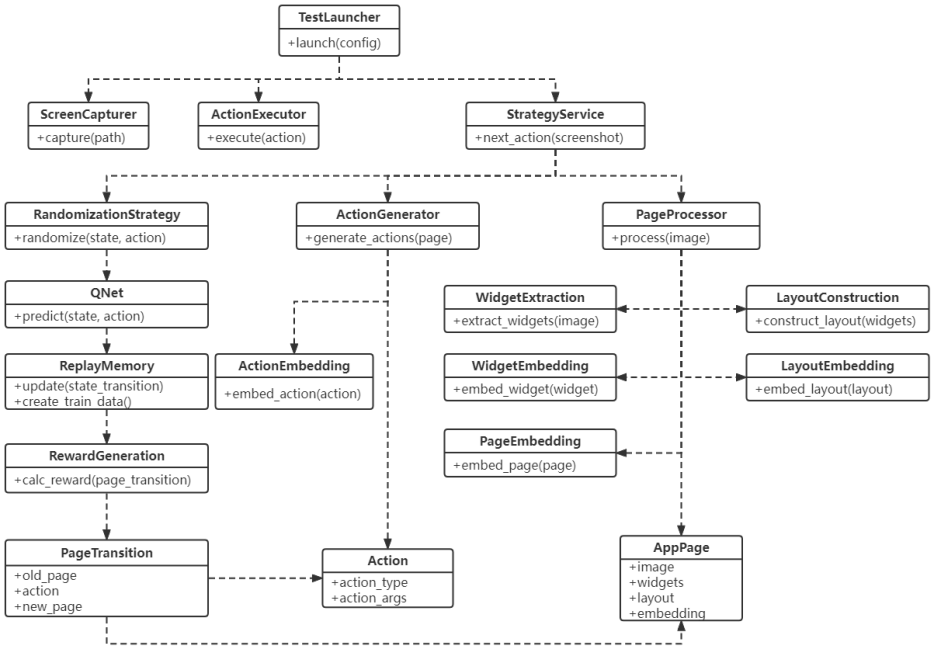


图 3.4: UniRLTest 系统逻辑视图

- PageProcessor 类。该类负责截屏处理工作，从截屏中提取出控件信息和布局信息，并完成对当前页面所示状态的嵌入。该类主要利用 WidgetExtraction 类、LayoutConstruction 类、WidgetEmbedding 类、LayoutEmbedding 类和 PageEmbedding 类来实现上述功能，并最终将所有信息整合到一个 AppPage 对象中。

- WidgetExtraction 类。该类负责从截屏中提取所有的控件信息。

- LayoutConstruction 类。该类负责根据截屏中的控件信息重新构建该页面的布局。

- WidgetEmbedding 类。该类负责对截屏中的所有控件进行单独嵌入与整体嵌入。

- LayoutEmbedding 类。该类负责对页面布局信息进行嵌入。

- PageEmbedding 类。该类负责整合 WidgetEmbedding 类和 LayoutEmbedding 类生成的嵌入向量，以整合后向量作为当前页面的嵌入向量（即待测应用的当前状态）。将页面嵌入后，才能输入 QNet 神经网络。

- ActionGenerator 类。该类为动作生成器，负责根据当前页面的控件集合生成所有可执行的动作 Action。
- ActionEmbeding 类。该类负责为每一个由 ActionGenerator 类生成的动作进行嵌入，以便输入 QNet 神经网络。
- QNet 类。该类为一个拟合强化学习中的 Q 函数的神经网络，负责为输入的 < 状态，动作 > 对计算相应的 Q 值，即该 < 状态，动作 > 对的价值。
- RandomizationStrategy 类。该类为 QNet 的包装类，在其基础上添加了随机化策略以应对强化学习中常常出现的探索-利用困境，以避免其陷入局部最优解。
- RewardGeneration 类。该类负责为每一次状态转移赋予一个即时奖励，以鼓励或惩罚系统上一次的决策，使系统学习到状态空间的知识从而优化其决策。
- ReplayMemory 类。该类记录了所有的状态转移及相关奖励，为了给 QNet 网络提供训练数据。

本系统的开发视图如图 3.5所示，该图展示了系统前端、服务端和算法端的主要模块。其中：

- 前端主要由页面组件 components、图像资源 image、javascript 脚本 javascript 和样式文件 css 四部分组成。
- 服务端主要由负责与前端页面交互的控制器 controller 模块、负责启动遍历测试的 launcher 模块、负责与算法端连接交互的 connection 模块、负责日志记录的 logger 模块、负责与 Android 设备交互的 android 模块和负责与 Web 页面交互的 web 模块组成，其中 android 模块内进一步可被分为负责截屏的 screen_capturer、负责执行各类操作的 action_executor 和二者公用的工具模块 utils。类似地， web 模块也可以划分为 screen_capturer、action_executor 和 utils 模块。
- 算法端主要由负责与服务端交互的 controller 模块、负责启动探索逻辑的 launcher 模块、负责记录状态转移的 replay_memory 模块、负责日志记录的 logger 模块、负责处理截屏信息的 img_processing 模块、负责实现强化学习策略的 strategy 模块、负责计算即时奖励的 reward 模块和负责生

成可执行动作的 action 模块。其中 img_processing 模块可进一步被拆分为负责保存页面信息的 app_page、负责提取控件信息的 widget_extraction、负责提取布局信息的 layout_construction、负责对控件信息、布局信息做嵌入的 widget_embedding 和 layout_embedding，以及负责整合两部分嵌入形成最终页面嵌入的 page_embedding。

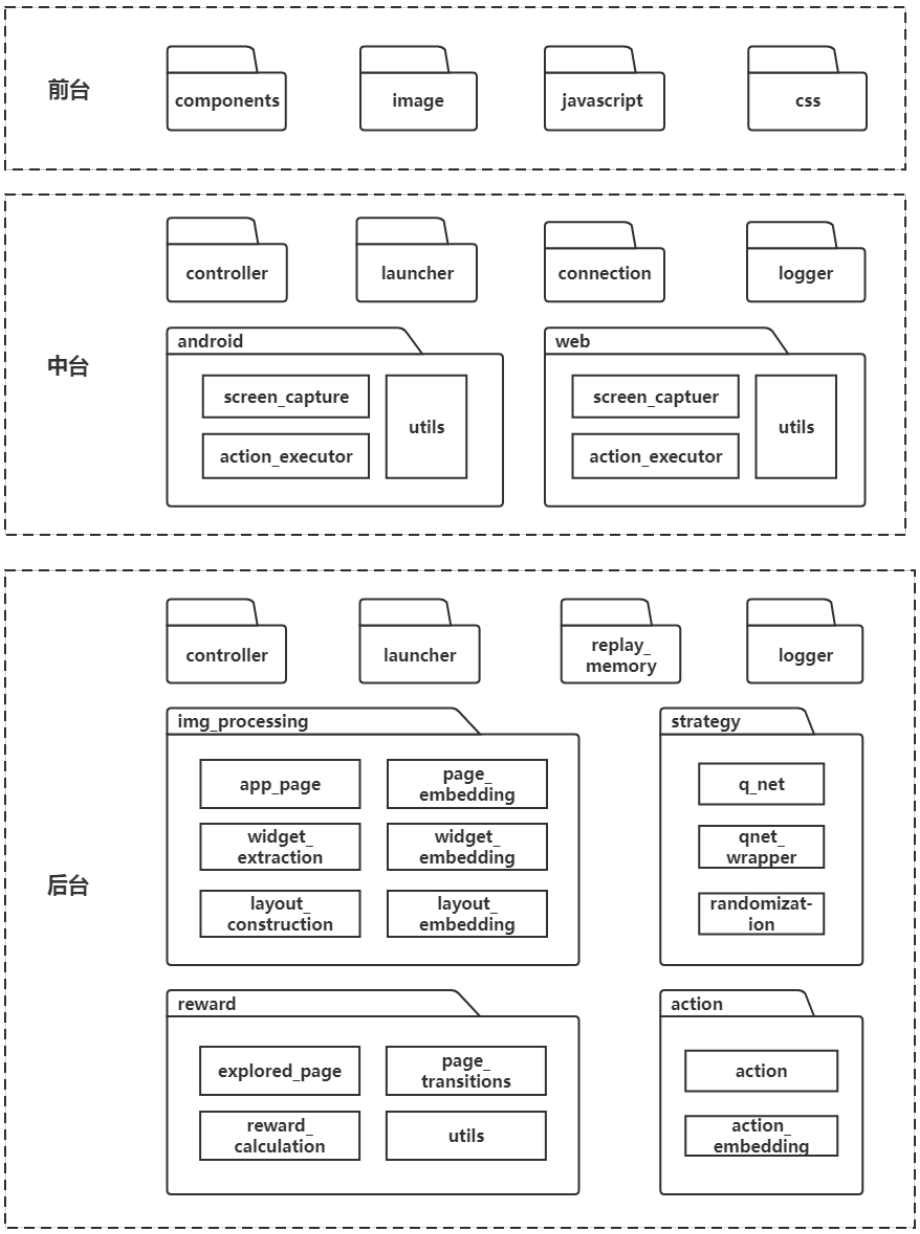


图 3.5: UniRLTest 系统开发视图

本系统的进程视图如图 3.6所示。该图中展示了从前端发起的一次遍历测试

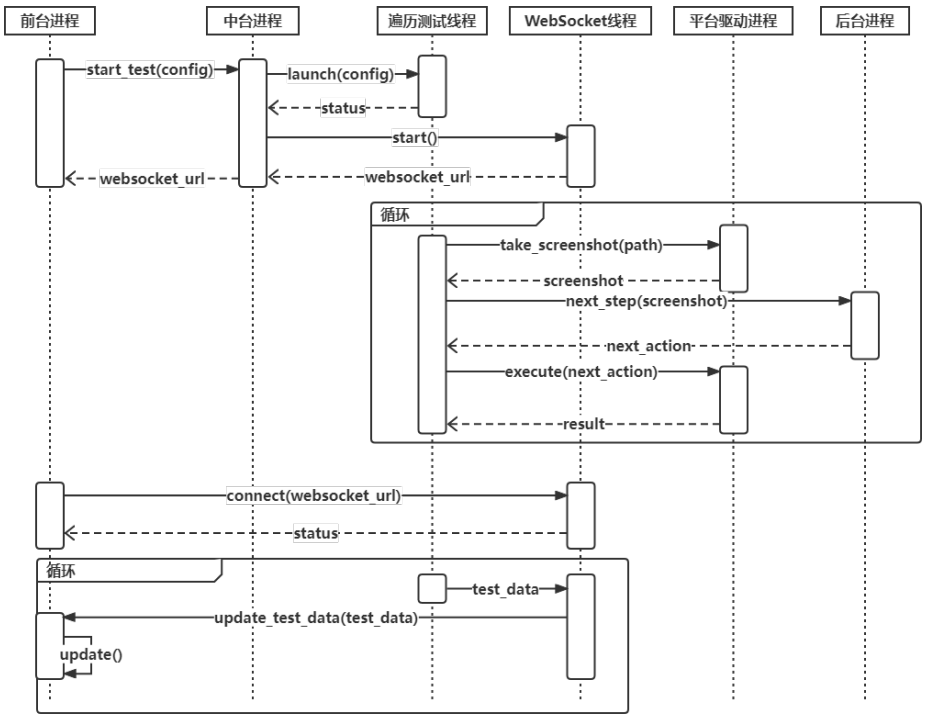


图 3.6: UniRLTest 系统进程视图

请求时系统各进程/线程间的交互情况。当前端进程通过 start_test(config) 开始遍历测试后，服务端进程会建立并启动一个相应的遍历测试线程负责遍历测试，同时还会建立并启动一个 WebSocket 线程负责将相关数据传输给前端，完成两个线程的启动后，服务端将该 WebSocket 线程中开启的 WebSocket 服务端地址返回给前端进程，以便前端进程连接相关 WebSocket 服务端。对于遍历测试线程而言，开始根据给定配置进行遍历测试，主要是与相关平台的驱动进程进行交互以捕获设备截屏和执行动作，以及与算法端进程交互以执行策略获取下一个最合适地动作。对于 WebSocket 线程而言，遍历测试线程会不断将每一步的测试数据传输给该线程，而该线程则将这些数据通过 WebSocket 传输给前端进程，前端进程从而可以在界面上展示这些数据，包括截屏和日志。当 WebSocket 断开连接时，本次遍历测试已结束。

本系统的物理视图如图 3.7所示。软件测试人员在自己的工作电脑上通过浏览器访问本系统的前端，通过 HTTP 协议和 WebSocket 协议与服务端交互，其中 HTTP 协议为前端向服务端发送请求使用的协议，而 WebSocket 协议为服务端主动向前端发送数据使用的协议；本系统的服务端则部署在一台连接着不同平台的设备的服务器上，该服务器上还需要同时部署一个 Web 驱动和 ADB，而

本系统的服务端为一个 Flask 应用，通过 HTTP 协议与算法端交互；本系统的算法端也是一个 Flask 应用，部署于一个服务器上。

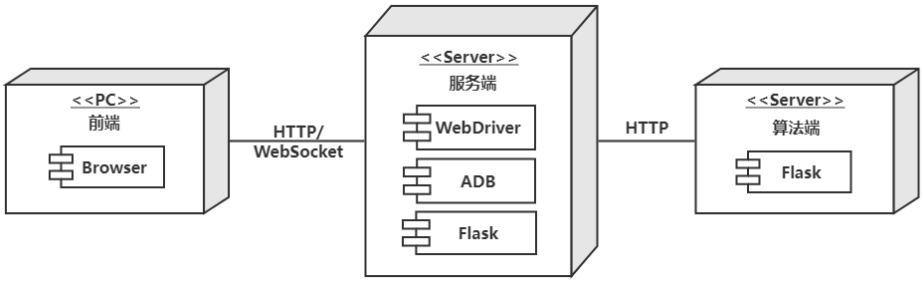


图 3.7: UniRLTest 系统物理视图

3.2.3 系统接口设计

本系统涉及到前端与服务端、服务端与算法端的交互，需要分别设计相关交互接口；除了 HTTP 接口外，还要对服务端主动向前端发送的测试数据的 WebSocket 设计数据交互接口。

表 3.8: 前端与服务端交互的 HTTP 接口定义

接口名称	接口 URL	方法	接口描述
遍历测试启动	http://<host_ip>:<host_port>/start	POST	根据给定配置启动本次遍历测试

对于前端与服务端的交互中基于 HTTP 协议的部分，其接口列表如表 3.8所示，其中 <host_ip> 和 <host_port> 分别表示服务端所部署于的服务器 IP 地址和端口。这部分仅设计一个接口，即遍历测试启动接口，其参数为 JSON 格式的配置，该接口使用 POST 方法，负责根据给定配置启动本次遍历测试。

表 3.9: 服务端与前端交互的 WebSocket 数据格式定义

格式名称	数据格式	描述
截屏数据格式	urt-img::<base64_image_data>	以 urt-img:: 开头，后面为图像的 Base64 编码字符串
日志数据格式	urt-log::<log>	以 urt-log:: 开头，后面为日志字符串
其它信息格式	urt-msg::<msg>	以 urt-msg:: 开头，后面为信息内容

对于服务端向前端发送测试数据的部分，即基于 WebSocket 协议的部分，主要需要为其传输的数据定义良好的数据格式，其数据格式定义列表如下表 3.9所

示，主要需要考虑截屏、日志和其它信息的数据传输格式，其中截屏信息需要以 Base64 编码后形成字符串以传输给前端，其它信息主要是遍历测试结束后服务端的 WebSocket 服务端主动于前端连接着的客户端断开连接前发送的提示信息。

表 3.10: 服务端与算法端的 HTTP 接口定义

接口名称	接口 URL	接口方法	接口描述
平台信息更新	/urt-backend/sys-info	POST	更新当前测试平台的信息
平台信息获取	/urt-backend/sys-info	GET	获取当前测试平台的信息
单步测试	/urt-backend/next-action	GET	获取下一个应当执行的动作

对于服务端与算法端的交互，其基于 HTTP 协议，相关接口定义如表 3.10所示，每个接口的 URL 前缀均为 http://<host_ip>:<host_port>，其中 <host_ip> 和 <host_port> 分别表示算法端所部署于的服务器 IP 地址和端口。接口列表中，平台信息更新接口需要一个 JSON 格式的平台信息参数，平台信息获取接口返回的同样也是一个 JSON 格式的平台信息，单步测试接口则返回一个列表格式的动作，其中包含该动作的详细信息。

3.3 本章小结

本章从系统用例分析着手逐步分析了系统的功能性需求和非功能性需求，接着根据这些需求进行系统的概要设计，结合系统架构图和 4+1 视图介绍了本系统的整体架构，然后进一步定义了不同模块之间交互的接口与数据格式，为本系统的详细设计与具体实现奠定基础。

第四章　技术设计与实现

本系统分为算法端、服务端和前端，本章着重介绍其中算法端关键模块的设计与实现，包括图像处理模块、动作生成模块、执行策略模块、即时奖励模块和回放记录模块；此外，本章还会简要介绍服务端两大关键模块：页面截屏模块和动作执行模块的设计与实现；最后展示本系统的前端界面。对于算法端和服务端的控制器模块、启动模块、日志模块等非关键模块的设计与实现，本章不做详细介绍。

4.1　图像处理模块

图像处理模块负责从待测应用截屏中提取所有控件和布局信息以供 UniRL-Test 算法端其他模块使用。该模块主要提供三种信息：控件边界框列表、页面布局和页面状态嵌入向量。后文会分别详细介绍这三种信息的提取与生成。

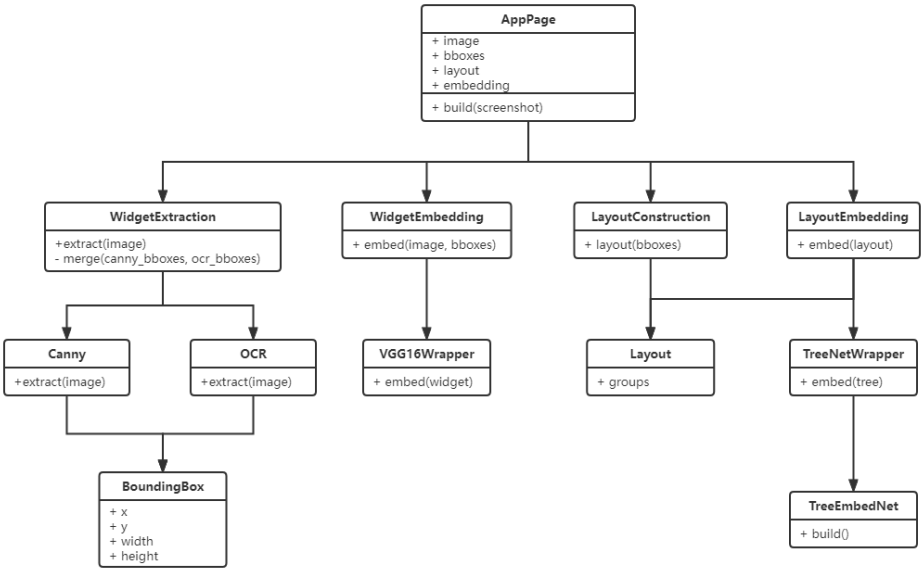


图 4.1: 图像处理模块的类图

图像处理模块的类图如图 4.1所示。其中 AppPage 类为页面模型类，包含了一个页面的图像、控件边界框列表、布局和最终的页面嵌入；该类还提供了一个静态方法用于从截屏图像中构建一个 AppPage 实例化对象。在构建 AppPage 类的实例对象时，首先需要利用 WidgetExtraction 类和 LayoutConstruction 类分

别提取其中的控件并构建页面布局结构，然后利用 WidgetEmbedding 类和 LayoutEmbedding 类实现控件信息和布局信息的嵌入，最终在 AppPage 中整合为最终的页面嵌入向量。WidgetExtraction 类依赖 Canny 类和 OCR 类进行控件边界框 BoundingBox 提取，然后进行边界框整合；而 LayoutConstruction 类所产生的页面布局 Layout 则为一个控件组的列表，其具体含义和内容会在下文中给出详细解释；在 WidgetEmbedding 类对控件进行嵌入时，需要借助 VGG16Wrapper 类对其中的图像信息进行嵌入；对于 LayoutEmbedding 类对布局进行嵌入的情况，则依赖 TreeNetWrapper 类，该类封装了 TreeEmbedNet 类，实现了它的训练与预测功能；而 TreeEmbedNet 类则是一个基于 LSTM 的神经网络，其功能为嵌入一个树状结构的输入，在本系统中，该树状结构的输入即为 LayoutConstruction 类构建出的布局结构。

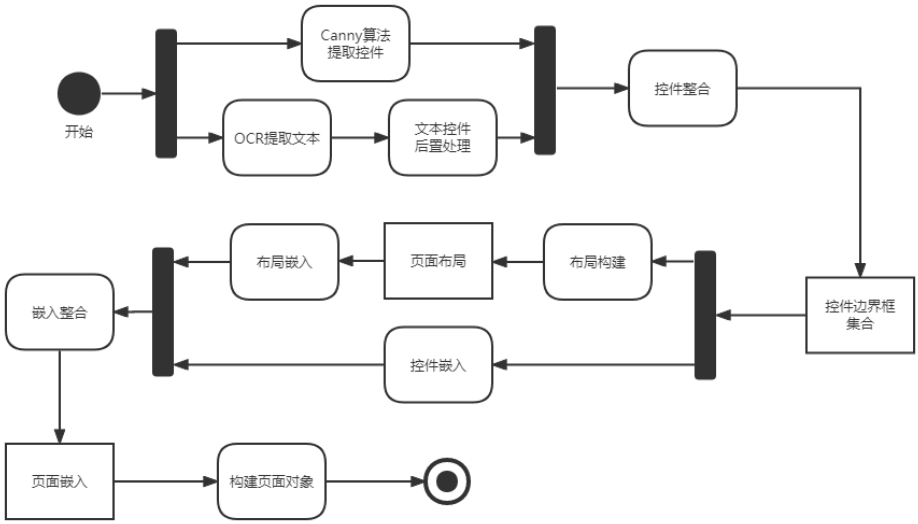


图 4.2: 图像处理模块的活动图

图像处理模块的活动图如图 4.2所示。该模块的输入为一个截屏图像，首先对该图像使用 Canny 算法和 OCR 技术分别提取一般控件和文本控件的边界框，在 OCR 技术提取出文本控件边界框后，还会进行一个后置处理提高控件边界框精度。接着对二者提取出来的边界框列表进行整合，形成最终的控件边界框集合。根据该集合可同时进行布局构建和控件嵌入，在布局构建后可以得到该页面的布局结构，进一步对该布局进行嵌入；最终将控件嵌入与布局嵌入整合为页面嵌入，将上述过程产生的控件边界框集合、页面布局和页面嵌入一同构成一个页面对象。至此，图像处理部分的工作便结束了，该页面对象可被动作生成、执行策略等模块复用以获取当前页面的各类信息。

接下来针对图像处理环节中的三个重点：控件边界框的提取、布局的构建和页面的嵌入进行一一详细介绍。

4.1.1 控件边界框列表的获取

控件边界框一般以 (x, y, w, h) 的形式给出，其中 (x, y) 分别是该控件左上角顶点的 x 坐标和 y 坐标，(w, h) 分别是该控件的宽度和高度。

控件边界框的提取主要利用 Canny 边缘提取技术和 OCR 技术。截屏中的控件主要有两种存在形式：图标和文本。对于图标类的控件，Canny 边缘提取技术更加优秀；而对于文本类的控件，OCR 技术效果更好。

基于 Canny 算法的控件边界框提取（以下简称为 Canny 控件提取）整体流程如图 4.3所示。我们首先将截屏灰度化，即从原始的 RGB 三通道图转化为灰度图，这一步是 Canny 算法的前置条件；接着，我们应用 Canny 算法，根据灰度图识别其中所有控件的边缘，并将这些边缘提取出来形成一幅黑白二值图，其中白色前景色表示控件边缘，黑色背景色表示非边缘；由 Canny 提取出来的边缘会受到该算法参数的影响从而不够细致，本该完整的边缘常常断裂，为了修复这些断裂的边缘，需要使用图像处理中的闭操作，将所有开口的边缘闭合起来，形成完整边缘；接着根据这些完整边缘绘制出等高线；对每一条等高线均生成一个外接矩形框作为其边框，这样就形成了初步的控件边界框列表；但是这些边界框在截屏原图中代表的区域可能存在诸如全黑、全白或者是面积过小等问题，这些问题区域均是噪点，而非真正的控件，因此需要将其去除，以得到最终的边界框列表。

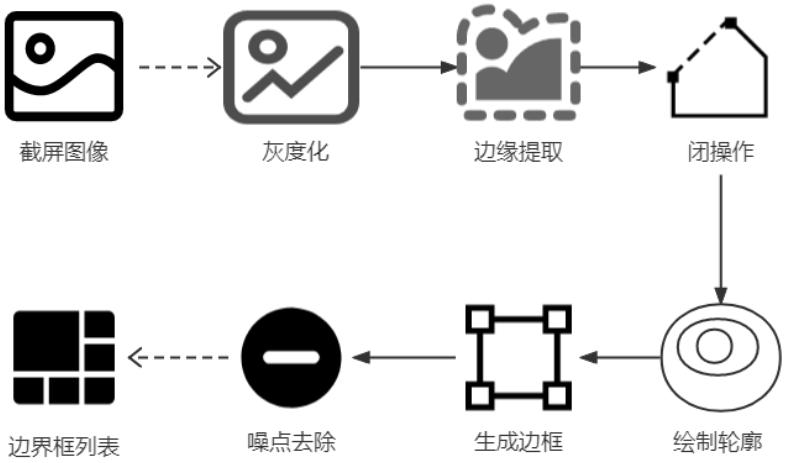


图 4.3: 基于 Canny 算法的控件提取流程

```
def canny_boundings(image, canny_sigma=0.33, close_iterations=4):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    v = np.median(gray)
    lower_threshold = int(max(0, (1 - canny_sigma) * v))
    upper_threshold = int(min(255, (1 + canny_sigma) * v))
    img_binary = cv2.Canny(gray, lower_threshold, upper_threshold, -1)
    kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (3, 3))
    img_closed = cv2.morphologyEx(img_binary, cv2.MORPH_CLOSE, kernel,
                                  iterations =close_iterations)
    contours, _ = cv2.findContours(img_closed, cv2.RETR_EXTERNAL,
                                   cv2.CHAIN_APPROX_SIMPLE)
    boundings = [cv2.boundingRect(c) for c in contours]
    return post_process(boundings)

def post_process(boundings, img, threshold=0.01):
    # 去除全黑或全白控件
    non_blanks = []
    for bounding in bounding_list:
        x, y, w, h = bounding
        node = img[y:y + h, x:x + w, :]
        if not np.count_nonzero(node) == 0 and \
           not np.count_nonzero(255 - node) == 0:
            non_blanks.append(bounding)
    # 去除面积较小的控件
    img_h, img_w = img.shape[:2]
    th = img_h * threshold
    tw = img_w * threshold
    results = [b for b in non_blanks if b[2] >= tw and b[3] >= th]
    return results
```

图 4.4: 基于 Canny 算子的控件提取算法

接下来详细介绍其中的实现细节。基于 Canny 算法的控件提取部分的实现代码如图 4.4。Canny 算法所需的两个阈值有高低之分，其中高阈值表示梯度超过该阈值的像素为强边缘像素，低阈值则表示梯度低于该阈值的像素为非边缘像素，至于梯度落在高低阈值之间的像素则根据其周围 8 个像素判断，若其周围存在一个强边缘像素，则该像素为弱边缘像素，反之则为非边缘像素。不难发现，高阈值越高则边缘像素越少。Canny 推荐的高低阈值之比在 2:1 到 3:1 之

间。而此处采用了基于中位数的自适应阈值，在将图像灰度化后，计算该灰度图的中位数，并以该中位数为基础，给定一个参数 sigma 作为偏差以确定 Canny 算法所需的高低阈值。这种自适应的 Canny 算法简单有效。

在图像处理的闭操作中有两个关键参数需要指定：内核与迭代数。闭操作的本质是先进行一次膨胀操作，将前景色（白色）的色块扩大，使得不相连的前景色块连接在一起；接着进行一次腐蚀操作，将扩大的前景色块缩回原来的大小，但并不会将已经合并成一整块的前景色块拆分开来。通过若干次闭操作，我们可以将二值图中破碎的前景色块连接到一起，使其形成一个完整的控件边缘。其效果见图 3-4（d）。在做膨胀与腐蚀操作时，需要指定相应的内核，通过该内核与图像做数学运算的方式实现相关操作。因此，在闭操作中所需的内核即是用于膨胀和腐蚀操作的内核，而迭代数则是闭操作的次数。内核越大，迭代次数越多，膨胀/腐蚀的范围就越大；但也不能过大，否则容易导致将多个独立控件合并为单个控件的问题。在具体实现中迭代次数通常设置为 4，而内核通常使用的是如下所示的 3x3 的内核：

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix} \tag{4.1}$$

在绘制轮廓时，需要指定两个参数：轮廓的检索模式和近似方法。轮廓的检索模式指明了是否检测外层或内层轮廓，以及是否对检测出的多层轮廓建立等级关系。通常情况下，由于边缘提取后的二值图中的控件边缘存在一定厚度，对这样的边缘提取出的轮廓会存在层叠，即出现多层轮廓；除此以外，图片类的控件由于其中的内容较为丰富，必然会在边缘提取过程中产生多层边缘，从而导致多层轮廓。考虑到以上情况，内层轮廓就不再准确，因此仅仅检测外层轮廓即可；由于只检测外层轮廓，这些轮廓之间自然也不存在等级关系。至于轮廓的近似方法，仅保留轮廓上的拐点即可，这样可以用较少的数据量实现保留有效信息的最大化，便于后续对轮廓的处理。

在噪点去除阶段，我们需要考虑存在哪些类型的噪点。常见的噪点有纯白和纯黑区域以及面积过小的区域。纯白或纯黑区域往往是由一些文字或图标中的一些中空区域造成的；而面积过小的区域常常是一些提示性的文本或图标，这些文本或图标通常不会有与用户的交互功能，考虑到重要内容常常会有较大面积的控件来承载，小面积的控件就表明这部分内容不太重要，可不做考虑；除此以外面积过小的区域还可能是截屏图像中的杂色块导致的，因此需要加以去除。

图 4.5是 Canny 控件提取的效果图，从（a）到（e）依次是：截屏原图，灰

度图，边缘提取结果，闭操作结果，轮廓绘制结果，以及提取出的边框。此处省略了生成边框的效果，因为其效果与最终结果同样也是一个边框列表。

(a) 截屏原图 (b) 灰度图 (c) 边缘提取结果

(d) 闭操作结果 (e) 轮廓绘制结果 (f) 边框绘制结果

图 4.5: Canny 控件提取效果图

接下来是**基于 OCR 技术的文本提取部分**。目前的 OCR 技术已经可以在短时间内识别一幅图像中的所有文本内容及其位置和大小，也有很多公司开放了他们的相关 API 以供开发者使用。在具体实现中，考虑到在中英文识别的准确度和效率，本工具采用了百度 OCR 的接口。该接口只需传入一张二进制的图像数据即可快速识别其中的所有中英文文本。但是百度 OCR 仍然存在一些问题。

- 文本合并的问题：由于实现层面的原因，对于在原图像中位于同一行（即 y 坐标相近）的文本，该 OCR 接口的返回结果中，有一定概率会被识别为同一段文本，即便其语义上存在断裂，即并非同一个控件上的文本。换言之，该 OCR 接口返回的结果中，位于同一行的多个控件上的文本会被合并到一起，很难进行拆分。

- 边界框精确度较低的问题：在该 OCR 接口的返回结果中，虽然文本行的宽度较为精确，但是其识别出的文本行的高度有时与其实际高度有所偏差，时高时低，这有可能是在版面分析时为版面文本分行的偏差所致。

```python
def deep_ocr(image_path, prob_threshold=0.9, gamma_h=0.8, gamma_d=0.8):
    coarse_texts = ocr(open(image_path, 'rb').read())
    text_boxes = []
    for words in coarse_texts['words_result']:
        if words['probability']['average'] < prob_threshold: continue
        left = words['location']['left']
        top = words['location']['top']
        width = words['location']['width']
        height = words['location']['height']
        if width < 10 or height < 10: continue
        wordbox = image[top:top + height, left:left + width, :]
        cropped = cv2.imencode('.jpg', wordbox)[1].tobytes()
        details = ocr(cropped, show_char=True)
        for detailed_words in details['words_result']:
            if detailed_words['probability']['average'] < prob_threshold: continue
            text_bboxes += process_detailed_words(detailed_words, left, top,
                prob_threshold, gamma_h, gamma_d)
    return text_boxes
```

图 4.6: 深度 OCR 核心算法

为了缓解或消除上述问题的影响，本系统采用了深度 OCR 的方法，即粗粒度 OCR 与细粒度 OCR 结合的方法，其具体实现如图 4.6所示。具体而言，对一张包含文本的图片，首先进行粗粒度 OCR：使用 OCR 技术提取其中所有的文本行，此处并不需要关注其内容细节，只需关注文本行的位置与大小，以及它的置信度。我们去除置信度较低的文本行，留下有效的文本行作细粒度处理。在具体实现中，该置信度阈值设置为 0.9。

```
def process_detailed_words(detailed_words, left, top, prob_threshold, gamma_h,
    gamma_d):
    split_idx = []
    for i, (p, q) in enumerate(zip(detailed_words['chars'][:-1],
        detailed_words['chars'][1:]) ):
        h1 = p['location'][ 'height']
        h2 = q['location'][ 'height']
        if abs(h1 - h2) > gamma_h * min(h1, h2):
            split_idx.append(i + 1)
            continue
        distance = q['location'][ ' left '] - p['location'][ ' left '] -
            p['location'][ 'width']
        threshold = gamma_d * min(h1, h2)
        if distance > threshold:
            split_idx.append(i + 1)
    results = []
    for i, j in zip ([0] + split_idx, split_idx + [len(detailed_words['chars'])]):
        q_loc = detailed_words['chars'][i][ 'location']
        box_x = q_loc['left'] + left
        box_y = q_loc['top'] + top
        r_loc = detailed_words['chars'][j - 1][ 'location']
        box_w = r_loc['left'] + r_loc['width'] - q_loc[' left ']
        box_h = q_loc['height']
        results .append((box_x, box_y, box_w, box_h))
    return results
```

图 4.7: 细粒度 OCR 为有效字符分组

提取出有效文本行后，我们针对每个文本行单独从原截屏中截取出来，并应用 OCR 技术提取该文本行的细节，这一步为细粒度 OCR。在提取细节时，我们需要关注该文本行中每个字符的位置和大小，以及它的置信度。与粗粒度 OCR

类似，这里需要去除置信度较低的字符（在具体实现中其阈值也一致）。接下来要为留下的有效字符合理分组，使其符合一个控件一组文本的原则，其具体实现代码如图所示 4.7。应用格式塔心理学的一些原理，我们制定出了一套文本分组原则如下：

1. 记相邻字符 c_1 和 c_2 的高度分别为 h_1 和 h_2，高度差阈值记为 γ_h，若 $|h_1-h_2| > \gamma_h \times min(h_1,h_2)$，则认为 c_1 和 c_2 不属于同一组。这一原则的原理是考虑到界面的整齐美观性，同一个控件内的字体大小常常是一致的。该原则中高度差阈值 γ_h 需要指定，γ_h 越小，对相邻字符的高度差就越敏感。

2. 相邻字符 c_1 与 c_2 的间距记为 d，间距阈值记为 γ_d，若 $d > \gamma_d \times min(h_1,h_2)$，则认为 c_1 和 c_2 并不属于同一组。这一原则是考虑到界面中通常不会使同一个控件中的短文本字符分布较为稀疏，否则影响美观。该原则中间距阈值 γ_d 需要指定，γ_d 越小，对相邻字符的间距越敏感。

3. 不满足上述两条原则中的分组条件的相邻字符被认为属于同一组。

利用上述深度 OCR 技术，我们可以实现对图像中的文本类控件边界框的精细提取。由于 Canny 控件提取时同样也会提取出文本类控件的边界框，而这样的边界框相比于基于深度 OCR 技术提取出的边界框而言不够准确，因此，最后需要将二者提取出的边界框整合至一个边界框列表中。

边界框整合的具体代码逻辑如图 4.8所示。在边界框整合过程中，由于基于 Canny 控件提取得到的控件边界框（代码中为 canny_boxes）常常比基于深度 OCR 得到的（代码中为 text_boxes）多，所以考虑将后者整合至前者中。对于 text_boxes 中的每一个边界框，首先根据矩形 IOU（Intersection of Union）分数从 canny_boxes 中找到一个最佳匹配，然后根据一定规则对该匹配进一步确定或将其中的文本边界框 rect_o 作为新的边界框加入 canny_boxes 中，或以 rect_o 替换该匹配中的另一个边界框 rect_c。以上过程中，矩形 IOU 分数是指两个矩形相互重叠部分与合并后图形的面积比。

计算 IOU 分数的函数 intersect 具体实现如图 4.9所示。该函数除了计算常规的 IOU 分数，还需要计算两个矩形相互重叠部分分别与这两个矩形的面积比，主要是考虑到一个优化：在用 rect_o 替换 rect_c 时，若出现一个文本控件被 Canny 识别为多个独立控件的情况，则一个 rect_o 应当替换多个 rect_c，且这种情况下 rect_o 与 rect_c 的 IOU 分数并不高，因为 rect_c 所代表的只是 rect_o 中的一部分。而这种情况下二者的重叠部分与 rect_c 的面积比 ratio_c 较高，因此可以利用这一点对该情况加以识别并处理。

```python
def merge_boxes(canny_boxes, text_boxes, threshold=0.7):
    mods = []
    for rect_o in text_boxes:
        best_match = (None, 0, 0, 0)
        for rect_c in canny_boxes:
            ratio_o, ratio_c, ratio = intersect(rect_o, rect_c)
            if ratio > best_match[3]:
                best_match = (rect_c, ratio_o, ratio_c, ratio)
        rect_c, _, ratio_c, _ = best_match
        if rect_c is not None:
            if ratio_c > threshold:
                mods.append(('replace', rect_c, rect_o))
                for rect_cc in canny_boxes:
                    if rect_cc == rect_c:
                        continue
                    _, ratio_cc, _ = intersect(rect_o, rect_cc)
                    if ratio_cc > threshold:
                        mods.append(('delete', rect_cc))
            else:
                mods.append(('add', rect_o))
        else:
            mods.append(('add', rect_o))

    for mod in set(mods):
        if mod[0] == 'replace':
            if mod[1] in canny_boxes:
                canny_boxes[canny_boxes.index(mod[1])] = mod[2]
            else:
                canny_boxes.append(mod[2])
        elif mod[0] == 'add':
            canny_boxes.append(mod[1])
        elif mod[0] == 'delete':
            if mod[1] in canny_boxes:
                canny_boxes.remove(mod[1])
```

图 4.8: Canny 边界框与 OCR 文本框的整合代码

由此，边界框的替换规则如下：若两个边界框重叠部分与文本边界框的面

积比 ratio_c 超过一个给定阈值，则用文本边界框替换另一个边界框。考虑到该情况下需要用文本边界框替换所有小边界框，因此在发现这样的边界框时还需进一步检索其它符合条件的边界框，并将这些边界框删除。

```
def intersect (rect_a, rect_b):
    a_x, a_y, a_w, a_h = rect_a
    b_x, b_y, b_w, b_h = rect_b
    dx = max(0, min(a_x + a_w, b_x + b_w) - max(a_x, b_x))
    dy = max(0, min(a_y + a_h, b_y + b_h) - max(a_y, b_y))
    S_i = dx * dy
    S_a = a_w * a_h
    S_b = b_w * b_h
    return S_i / S_a, S_i / S_b, S_i / (S_a + S_b - S_i)
```

图 4.9: 计算三类 IOU 的函数代码

至此，截屏中的控件提取工作就完成了，最终得到了该截屏中所有重要控件的边界框列表以供后续处理。

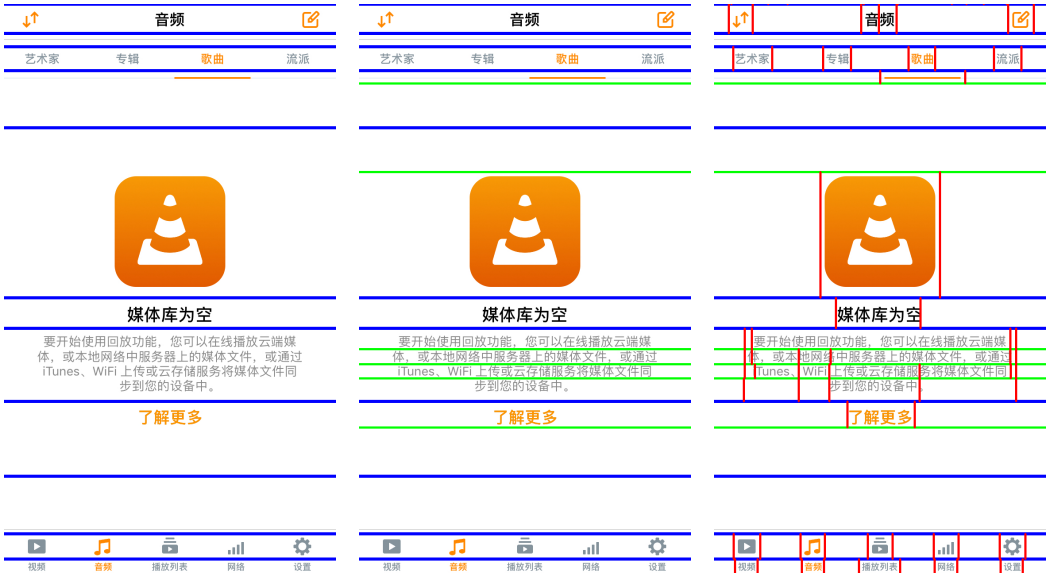
4.1.2 页面布局的生成

提取出页面中的所有重要控件后，就可以基于此生成该页面的控件树了。

页面控件树是页面布局的一种表现形式。在前端设计师设计页面时，基于格式塔心理学的部分原则，通常会将语义相近的控件较为紧密地排列在同一行、同一列，或者整体的排列在同一区域；与之相对的，语义差距较远的控件之间排列距离往往也较远。由此形成了页面的一个个区域，每个区域内包含了若干控件。由于人的阅读习惯通常是先横向阅读而后纵向阅读，因此多数布局中，一个区域总是在横向上占据一整行，不同区域按列排布。

基于以上设计通用实践，我们将页面的布局分为三层：控件组、控件行和控件列，如图 4.10所示，并由此组织起一个页面的控件树。其中控件组对应了页面的一个语义相近的控件集合形成的区域，而控件行是对该区域进行的横向上的细分，控件列则是在控件行内的细分。之所以控件组内存在分行和分列，是因为通常情况下，尽管语义相近，然而不同控件语义仍有不同，需要用不同的大小来表示，因此存在一些高度倍于其它同组控件的较大控件，或是一些高度较小于其它同组控件的较小控件，为了准确表明这些控件在布局中的不同，必须精细地进行分行分列。

生成以上所述的以控件组、行、列表示的页面布局，共有四个步骤：



(a) 分组效果（蓝线分隔） (b) 分行效果（绿线分隔） (c) 分列效果（红线分隔）

图 4.10: 组行列效果图

1. 初步分组：根据控件中心点的 y 坐标尝试将其归入一个候选组内，只要其中心 y 坐标落入候选组的上下边界内即可将其归入该候选组，并更新该候选组的上下边界。若无法归入一个候选组，则将其看作一个新的候选组。候选组代表了语义相近的控件的集合。

2. 细化分组：候选组之间可能存在着重叠部分，因此需要将其去除。在细化分组过程中，若一个候选组的中心落入另一个候选组内，则认为两个候选组语义接近，需要将其合并为一组。对于中心不落入其它候选组的候选组，若存在重叠，则将重叠部分按照 50%-50% 的规则划分给二者；若互不重叠，则取二者中线作为其分割线，以此更新二者的上下边界，使相离的两组紧贴。除此以外，对于高度过低的候选组，通常认为这样的候选组为非法候选组，因此需将其按 50%-50% 的规则拆分给它的上下候选组。

3. 组内分行：组内存在着高度不同的大小控件，需要给小控件分配一行，而大控件往往可以占用若干行。在组内分行的过程中，根据该组内的所有控件的上下边界，将该组划分为若干行；对于高度过小的控件行，仿照细化分组过程中对高度过小的控件组的做法，将其拆分给上下行即可。

4. 行内分列：对于占用了同一个控件行的所有控件，根据其左右边界即可将该行划分出若干列；类似的，对于宽度过小的控件列，将其等比例拆分给左右列即可。

以上是布局生成的逻辑，接下来介绍布局生成的实现细节。

布局生成的第一步 —— 初步分组的实现代码如图 4.11所示。其中候选组列表用 basic_rows 表示，该方法的输入参数 bounding 为一个控件边界框列表。首先，我们根据控件中心点的 y 坐标对该边界框列表排序，得到一个中心自上而下排列的控件边界框列表。接着遍历该列表，对其中的每一个边界框，都尝试根据其中心点 y 坐标 center_y 将其归入一个候选组中，并更新该候选组的上下边界；若找不到合适的候选组，则为它生成一个新的候选组，候选组从数据结构而言是一个含有 3 个元素的列表，其中第一个元素是从属于该候选组的控件边界框列表，后两个元素分别是该候选组的上边界和下边界，一个候选组由其上下边界唯一确定。

```
def basic_row_generation(bounding):
    basic_rows = []
    for bounding in sorted(bounding, key=lambda b: b[1] + b[3] / 2):
        x, y, w, h = bounding
        center_y = y + h / 2
        found = False
        for row in basic_rows:
            ceiling = row[1]
            ground = row[2]
            if ceiling <= center_y <= ground:
                row[0].append(bounding)
                row[1] = min(ceiling, y)
                row[2] = max(ground, y + h)
                found = True
                break
        if not found:
            basic_rows.append([[bounding], y, y + h])
    return basic_rows
```

图 4.11: 初步分组的实现代码

布局生成的第二步 —— 细化分组的代码如下所示。在具体实现中，我们首先对每个候选组都生成了一个控件组，以此组成控件组列表 groups，然后不断调整该列表。 groups 就数据结构而言是一个包含了 3 个元素的列表，其中第一个元素为该控件组所包含的候选组列表，后两个元素分别为该控件组的上下边界。接下来在调整 groups 时，共分为 4 个小步骤。

```python
def group_generation(basic_rows, resolution):
    groups = [[[row], row[1], row[2]] for row in basic_rows]
    # 合并存在包含关系的控件组
    groups = merge_groups(groups)
    # 分离部分重叠的控件组
    separate_groups(groups)
    # 调整远距离控件组的上下边界
    adjust_sibling_groups(groups)
    # 处理高度过小的控件组
    g_threshold = 1.5 * resolution[1] / 100
    return process_short_groups(groups, g_threshold)
```

图 4.12: 细化分组的实现代码

1. 合并存在包含关系的控件组。其具体实现代码如图 4.13所示。我们使用了一个布尔指示器列表 surviving 来标记 groups 对应控件组的有效性，若 surviving[i] 为 True 则表明 groups[i] 是有效的，需要被保留。对于每一个中心 y 坐标落入其它组 group_j 的控件组 group_i，我们根据 group_i 的上下边界，更新 group_j 的上下边界，接着将 surviving[i] 置为 False，表明该控件组已被归入其它控件组，因此不再为有效控件组。

```python
def merge_groups(groups):
    surviving = [True] * len(groups)
    group_count = 0
    while not len(groups) == group_count:
        group_count = len(groups)
        for i, group_i in enumerate(groups):
            for j, group_j in enumerate(groups):
                if not i == j and surviving[j] and group_j[1] <= (group_i[1] +
                    group_i[2]) / 2 <= group_j[2]:
                    group_j[0] += group_i[0]
                    group_j[1] = min(group_j[1], group_i[1])
                    group_j[2] = max(group_j[2], group_i[2])
                    surviving[i] = False
                    break
    return [group for i, group in enumerate(groups) if surviving[i]]
```

图 4.13: 合并存在包含关系控件组的实现代码

2. 分离部分重叠的控件组。其具体实现代码如图 4.14所示。在将存在包含关系的控件组合并完成后，仍然存在一些相互重叠但无包含关系的控件组，需要将这样的控件组泾渭分明地分隔开。对于下边界落入另一个控件组 group_j 的 group_i 而言，只需将 group_i 的下边界和 group_j 的上边界均设置为二者的中线（即 y 坐标的均值）即可实现分隔。对于上边界落入 group_j 的 group_i 而言类似，将 group_i 的上边界与 group_j 的下边界均设置为二者的中线即可。

```
def separate_groups(groups):
    for i, group_i in enumerate(groups):
        for group_j in groups[i + 1:]:
            if group_j[1] < group_i[2] < group_j[2]:
                group_i[2] = int((group_i[2] + group_j[1]) / 2)
                group_j[1] = group_i[2]
            elif group_j[1] < group_i[1] < group_j[2]:
                group_i[1] = int((group_i[1] + group_j[2]) / 2)
                group_j[2] = group_i[1]
```

图 4.14: 分离部分重叠控件组的实现代码

```
def adjust_sibling_groups(groups):
    if len(groups) > 0:
        # 重置第一个和最后一个的top和bottom
        groups[0][1] = 0 # The first group should be at the top.
        groups[-1][2] = resolution[1]  # The last group should be at the bottom.
    for prev, cur in zip(groups[:-1], groups[1:]):
        # 拼接两个相邻的分组，将其分离的top和bottom统一为二者均值
        if prev[2] < cur[1]:
            cur[1] = int((prev[2] + cur[1]) / 2)
            prev[2] = cur[1]
```

图 4.15: 调整远距离控件组上下边界的实现代码

3. 调整远距离控件组的上下边界。其具体实现如图 4.15所示。这一步是为了使控件组列表能覆盖屏幕中的每一个像素。首先是将第一组的上边界置为 0 以及将最后一组的下边界置为图像高度。接着对每一对相邻控件组 prev

和 cur，若 prev 的下边界在 cur 的上边界的上方，即二者分离，则将二者均设置为它们的中线。

4. 处理高度过小的控件组。其实现代码如图 4.16所示。此处，高度阈值被设置为图像高度的 1.5%。若一个控件组高度小于该阈值，则认为它不是一个合法控件组，需要将其去除。在去除不合法控件组时，我们不考虑其中包含的控件，因为这些控件同样被视为不合法控件。此处的去除不合法控件组的实现同样利用了 surviving 列表，其作用参见第一个小步骤。若不合法控件组位于所有控件组的顶端，则将其并入下一个控件组；若位于所有控件组的底部，则将其并入上一个控件组；若位于中间位置，则将其均分给其上下两个控件组。

```
def process_short_groups(groups, g_threshold):
    surviving = [True] * len(groups)
    for i in range(len(groups)):
        if groups[i][2] - groups[i][1] < g_threshold:
            if i - 1 < 0 and i + 1 < len(groups):
                # 位于顶端，则被下一个分组吞并
                groups[i + 1][0] += groups[i][0]
                groups[i + 1][1] = groups[i][1]
            elif i + 1 >= len(groups) and i - 1 >= 0:
                # 位于底端，则被上一个分组吞并
                groups[i - 1][0] += groups[i][0]
                groups[i - 1][2] = groups[i][2]
            elif i - 1 >= 0 and i + 1 < len(groups):
                # 位于中间，则被上下分组均分
                height_a = groups[i - 1][2] - groups[i - 1][1]
                height_b = groups[i + 1][2] - groups[i + 1][1]
                if height_a < height_b:
                    groups[i - 1][0] += groups[i][0]
                    groups[i - 1][2] = groups[i][2]
                else:
                    groups[i + 1][0] += groups[i][0]
                    groups[i + 1][1] = groups[i][1]
            surviving[i] = False
    return [group for i, group in enumerate(groups) if surviving[i]]
```

图 4.16: 处理高度过小控件组的实现代码

布局生成的第三步 —— 组内分行的实现代码如图 4.17所示。这一步的核心逻辑在于先从该组所包含的控件中整理出它们的上下边界，形成边界线的集合 lines，接着将其转化为一个有序列表，并去除其中距离过近的边界线。这些边界线构成了该组内所有控件行的分割线。利用这些分割线可以实现组内分行，由此生成的控件行从数据结构视角而言是一个包含 3 个元素的列表，其中第一个元素是所有占用了该控件行的控件集合（特别的，由于接下来的行内分列无需 y 坐标的参与，因此仅保留了每一个控件的 x 坐标和宽度 w），后面两个元素分别是该控件行的上下边界。接着对每一个控件行，从该组的控件集合中找到所有与其有重叠区域的控件，加入该控件行的控件集合中。

```
def gen_rows(group, line_merge_threshold):
    # 将分组内所有控件的y坐标为分割线形成行，并去除高度过小的行
    group[0][0][1] = group[1]
    group[0][-1][2] = group[2]

    nodes = [(y, h) for node_row in group[0] for _, y, _, h in node_row[0]]
    lines = sorted(set([y for y, _ in nodes] + [y + h for y, h in nodes] + [group[1],
        group[2]]))
    merge_close_lines(lines, line_merge_threshold)
    lines[0] = group[1]
    lines[-1] = group[2]

    rows = []
    for top, bottom in zip(lines[:-1], lines[1:]):
        # 将与该行有重叠的所有控件的x坐标作为分割线形成列，同样去除宽度过小的列
        filtered_basic_rows = [row for row in group[0] if not (bottom <= row[1] or
            top >= row[2])]
        filtered_nodes = [(x, w) for row in filtered_basic_rows
                          for x, y, w, h in row[0] if not (y + h <= top or y >=
                              bottom)]
        rows.append([filtered_nodes, top, bottom])
    group[0] = rows
```

图 4.17: 组内分行的实现代码

这一步涉及到的近距离分界线的合并逻辑 merge_close_lines(lines, threshold) 函数如图 4.18，其中的距离阈值在具体实现中设置为图像高度的 1.5%。该函数的主要功能是将距离小于阈值的两条分界线合并为其中线。

```
def merge_close_lines(lines, threshold):
    # 将lines中所有距离过近的线两两合并，lines可以是行或列的边界
    i = 0
    if len(lines) < 2:
        return
    first = lines[0]
    last = lines[-1]
    while i + 1 < len(lines):
        if lines[i + 1] - lines[i] < threshold:
            lines[i] = int((lines[i] + lines[i + 1]) / 2)
            lines.pop(i + 1)
        else:
            i += 1
    # 防止所有线都被合并导致不能构成行或列
    if len(lines) < 2:
        lines.clear()
        lines.extend([first, last])
```

图 4.18: merge_close_lines 函数的实现代码

```
def gen_cols(row, img_w, col_merge_threshold):
    filtered_nodes = row[0]
    cols = sorted(set([x for x, _ in filtered_nodes] + [x + w for x, w in
        filtered_nodes]))
    if len(cols) == 0 or not cols[0] == 0:
        cols = [0] + cols
    if not cols[-1] == img_w:
        cols.append(img_w)
    if len(cols) > 0:
        merge_close_lines(cols, col_merge_threshold)
        cols[0] = 0
        cols[-1] = img_w
        cols = [[left, right] for left, right in zip(cols[:-1], cols[1:])]
    row[0] = cols
```

图 4.19: 行内分列的实现代码

布局生成的最后一步 —— 行内分列的实现代码如图 4.19所示。控件列的数

据结构为包含了 2 个元素的列表，其中的 2 个元素分别为该列的左右边界。行内分列与组内分行相似，但更为简单。只需整理出该控件行内所有控件的左右边界形成边界线集合并排序后，添加 0 和图像宽度 img_w 作为控件列的首尾，除去距离过近的边界线，即可以这些边界线为分割线为该行划分出若干列。

至此，页面布局也由以控件组、控件行和控件列组织起来的控件树的形式生成了，可以进一步生成页面状态嵌入向量。

4.1.3 页面状态嵌入向量的生成

页面状态嵌入向量（以下简称为“页面嵌入”）包含两部分的嵌入向量：页面控件嵌入向量（以下简称为“控件嵌入”）和页面布局嵌入向量（以下简称为“布局嵌入”）。

对于**控件嵌入**而言，由于已经提取出了页面上的所有有效控件，只需将这些控件进行单独嵌入，然后整合起来即可。对于一个控件而言，其有效信息包含：控件图像、控件边界框（包含了位置和大小）和控件类型，因此，需要分别对这三类信息进行嵌入，才能得到单个控件的完整嵌入。

- 控件图像的嵌入：对于图像而言，其嵌入向量一般采用提取图像特征组成向量的方式，卷积神经网络（CNN）在提取图像特征方面具有较大的优势，因此可使用 CNN 提取图像特征，最后使用一层全连接层将其转化为嵌入向量。

- 控件边界框的嵌入：边界框只有 4 个要素：左上角的坐标及其宽高。但是考虑到其边界框具有较高的重要性，需要提升其嵌入向量的维数，因此需要先根据边界框生成一张二值图，其中前景色表示控件所占据的区域，其余用背景色表示；然后用与控件图像的嵌入类似的方式，利用 CNN 进行嵌入。

- 控件类型的嵌入：首先需要利用控件分类模型对所有控件进行分类，接着根据类别生成相应的 one-hot 向量作为其类型的嵌入向量。

通过以上方式，可以得到单个控件的三类信息的嵌入。将这三个嵌入向量拼接起来即可形成该控件的嵌入，将这些单个控件的嵌入向量整合起来即可得到页面的控件嵌入。

控件嵌入部分的实现代码较为简单，其中相对而言较为复杂的控件边界框的嵌入代码如图 4.20所示。在处理图像相关嵌入时，本系统使用了 VGG16 网络，该网络结构较为简单，提取特征的效果也较为优秀；本系统去除了 VGG16 网络

用来分类的最后一层全连接层，因此得到的图像嵌入和边界框嵌入是 4096 维的向量。对于边界框嵌入，我们首先将所有的边界框根据 VGG16 所规定的输入图片的大小进行了缩放，接着为每一个边界框生成了一张三通道图，并将由该边界框指示的控件占有的区域置为前景色 1，接着考虑到效率问题，我们将所有向量堆叠起来一起输入 VGG16 网络得到相应嵌入向量。

```
def embed_bbox(bboxes, img_width, img_height, vgg_input_size=(224, 224),
    batch_size=None):
    bboxes_resized = [(x * vgg_input_size[0] // img_width,
                       y * vgg_input_size[1] // img_height,
                       w * vgg_input_size[0] // img_width,
                       h * vgg_input_size[1] // img_height) for x, y, w, h in bboxes]
    hot_maps = [np.zeros((*vgg_input_size, 3)) for _ in range(len(bboxes))]
    for hot_map, bbox in zip(hot_maps, bboxes_resized):
        x, y, w, h = bbox
        hot_map[y:y+h, x:x+w, :] = 1
    embeddings = vgg16.embed(np.stack(hot_maps_resized, axis=0),
        batch_size=batch_size)
    return embeddings
```

图 4.20: 控件边界框嵌入的实现代码

在对所有单个控件的嵌入向量进行整合的具体实现中，考虑到页面嵌入最终需要作为执行策略模块中的神经网络的输入，因此该嵌入的维数必需是固定的，不能随页面的变化而变化。作为页面嵌入的一部分，控件嵌入的维数也必须为固定值，不能受到控件数量的影响。因此，我们对所有单个控件的嵌入向量求了平均，将该平均嵌入向量作为该页面的控件嵌入。

对于**布局嵌入**而言，我们需要从由一个树状结构展现的布局信息中提取其特征并形成特征向量。对树状结构的编码方式有许多种，常见的如先根遍历法、中根遍历法或后根遍历法等，也可以将其转化为一棵二叉树后再应用二叉树的编码方式。考虑到布局嵌入需要作为执行策略模块的输入，必须将不定长度的布局结构转化为一个定长的向量。结合以上考虑，在生成布局嵌入过程中，需要首先将布局编码为一个序列，接着利用循环神经网络对该序列进行特征提取，生成最终的嵌入向量。具体步骤如下所述：

1. 提取布局的纯结构信息。所谓纯结构信息，即将控件树中每一个控件组、控件行和控件列的边界信息忽略，而仅仅考虑共有多少控件组，每一组下

共有多少控件行，每一行内共有多少控件列。其最终生成一个纯结构的控件树，其中控件列为其叶节点。之所以忽略边界信息，是因为边界信息的移动并不会改变布局的结构，唯有结构具有较高的重要性。

2. 编码纯结构控件树。这一步是将上一步生成的包含纯结构信息的控件树编码，形成一个纯结构的序列。该系统采用括号表示法对树进行编码。所谓括号表示法，即对树中的每个节点 n，将其表示为 $nc_1c_2...$ 的形式，其中 c_1 和 c_2 均为节点 n 的子节点的括号表示。例如，图 4.21中表示的树的括号表示为 A{B{E F} C D{G{H}}}。每一棵树与其括号表示均存在一一对应的关系。

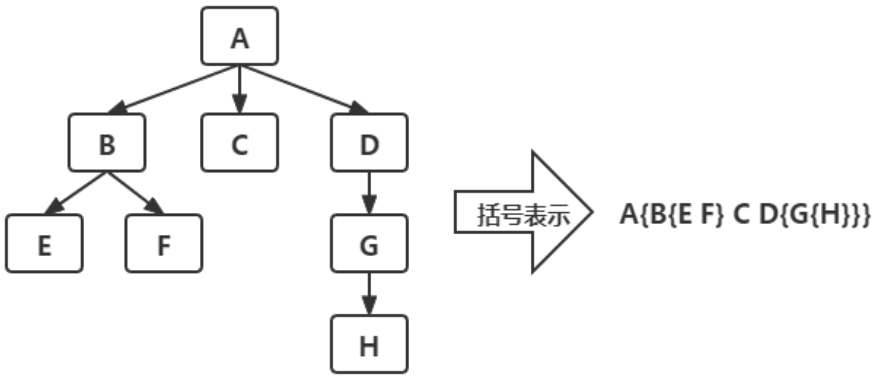


图 4.21: 树的括号表示法示例

3. 嵌入编码序列。利用长短期记忆网络（LSTM）对序列数据的强大的特征提取能力，提取由上一步生成的布局编码序列中的特征，并由此生成一个嵌入向量作为该布局结构的嵌入。

提取布局的纯结构信息的具体实现代码如图 4.22所示。在该实现代码中，主要函数 generate_tree_struct(grcs, label) 递归地为输入的控件树生成纯结构的控件树。其中 grc 表示控件组、控件行或控件列中的任意一种，对于控件组或控件行，其长度为 3，对于控件列其长度为 2。 label 是给最终的纯结构控件树中的节点打上的标签，通常是一个数字，其目的是便于编码；同时为了减少标签对最终编码的影响，必须对所有节点统一使用相同的标签。

在编码纯结构控件树时，会递归地将一个纯结构控件树转化为其对应的括号表示。特别的，此处并非使用原来的字符串的括号表示，而是进一步地用数字代替括号，产生数字序列，以此将字符串转化为数字序列，简化数据格式以便于输入下一步的 LSTM 中。

```
def generate_tree_struct(grcs, label=0):
    tree = [label, []]
    children = tree[1]
    for grc in grcs:
        if len(grc) == 3:
            child = generate_tree_struct(grc[0], label)
        else:
            child = [label, []]
        children.append(child)
    return tree
```

图 4.22: 提取布局纯结构信息的实现代码

```
class TreeEmbeddingNet:
    def build(self):
        with tf.variable_scope('tree_embedding'):
            with tf.variable_scope('embedding', reuse=tf.AUTO_REUSE):
                embedding_dict = tf.get_variable('embedding_dict', [self.vocab_size,
                    self.embed_size])
                embeddings = tf.nn.embedding_lookup(embedding_dict, self.input_seq)
                embeddings = tf.expand_dims(embeddings, axis=0)
            lstm_cell = tf.contrib.rnn.BasicLSTMCell(self.num_lstm_units)
            if self.mode == ModelMode.train:
                lstm_cell = tf.contrib.rnn.DropoutWrapper(lstm_cell,
                    input_keep_prob=self.lstm_dropout_keep_prob,
                    output_keep_prob=self.lstm_dropout_keep_prob)
            with tf.variable_scope('lstm', reuse=tf.AUTO_REUSE) as lstm_scope:
                zero_state = lstm_cell.zero_state(batch_size=1, dtype=tf.float32)
                _, initial_state = lstm_cell(embeddings[:, 0, :], zero_state)
                _, final_state = tf.nn.dynamic_rnn(lstm_cell, embeddings,
                    initial_state=initial_state, dtype=tf.float32, scope=lstm_scope)
            final_state = tf.concat(final_state, axis=1)
            with tf.name_scope('dfs_fc'):
                output = tf.layers.dense(final_state, self.hidden_size,
                    activation=tf.nn.leaky_relu, reuse=tf.AUTO_REUSE)
        self.embedding = output
        return output
```

图 4.23: 布局嵌入核心 LSTM 的实现代码

最后嵌入编码序列的核心 LSTM 的实现如图 4.23所示。该网络主要分为 Embedding 层、LSTM 层、Dropout 层和最后的全连接层，Embedding 层将输入转化为 512 维的张量后由 LSTM 层共 512 个 LSTM 单元来捕获特征，Dropout 层则是正则化的一种方式，仅保留 80% 的激活神经元，以减少过拟合程度，最后全连接层则将 LSTM 层捕获的特征（指输出向量和记忆向量）进一步转化为一个 512 维的张量。该 LSTM 在训练时采用孪生网络的形式，以拟合树编辑距离 [43] 为目标，以输出距离和实际距离的均方误差为损失函数，以随机生成的树对作为数据集进行训练。

当控件和布局的嵌入均完成后，可以对其进行整合，此处整合的方式即为向量的拼接。这便形成了该页面最终的状态嵌入向量。

4.2 动作生成模块

对于软件测试而言，动作是指各类事件的触发，包括用户交互事件和系统事件。这些动作与一般的强化学习应用中的动作不同，对于一个给定页面，一个动作往往是携带参数的，尤其是用户交互事件，例如点击操作，其参数为被点击的控件，随着控件不同，动作也就不同。而不同页面的可交互控件的种类不一、数量不一，也导致了对于每个页面而言，其事件集合不尽相同。另一方面，一个待测应用整体而言有许多控件，不同应用的控件总数亦不相同，很难给出一个事件超集以应对不同应用的不同页面。因此，我们必须单独为每个页面生成动作，并为之编码，将当前的状态嵌入与编码后的该动作嵌入一同输入执行策略模块，以得到该状态下该动作的预期收益。

动作生成模块的类图如图 4.24所示。ActionGenerator 是一个负责生成各类动作的接口，其下有针对不同动作的不同实现类，如 ClickGenerator 类、InputGenerator 类、LongPressGenerator 类、ResizeWindowGenerator 类等，每一个动作类型对应了一个相应的实现类。Action 类为动作的模型类，存储了一个动作所包含的每个要素，其中动作类型为一个枚举值，取值范围由 ActionType 枚举类定义。ActionEmbedder 类则负责对一个动作进行嵌入，在计算动作嵌入时，还需要利用到图像处理模块中的 WidgetEmbedding 类以对动作中的被操作控件进行嵌入。

根据以上论述，动作生成模块总共分为两部分：事件生成和动作编码。接下来分别详细介绍这两个部分的实现细节。

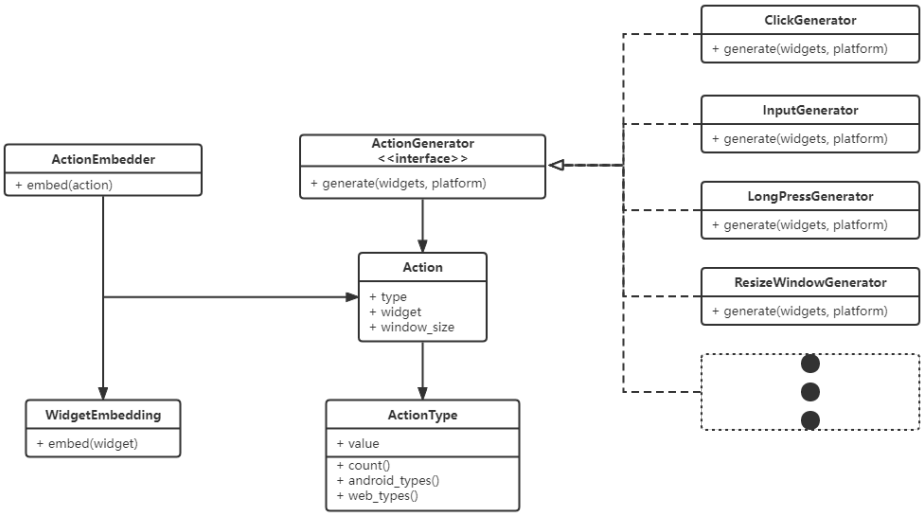


图 4.24: 动作生成模块的类图

4.2.1 事件生成

事件生成部分负责为当前页面生成所有可触发的事件（即可执行的动作，后文中动作与事件含义相同）。一个动作包含类型、参数两个组成部分。对于动作类型而言，不同的平台支持不同类型的动作。根据动作的对象不同，可以将动作分为以下三大类：

1. 对控件的动作：包含各类点击（单击、长按、右击、中键）、输入与滑动。
2. 对窗口的动作：包含对窗口大小的调整，对于移动平台而言，还包含了横竖屏切换。
3. 系统事件：系统事件有许多种类，本工具仅考虑一些较为常见的事件，例如前后台切换、权限的授予与拒绝、网络的切换等。移动设备上还存在电话中断事件，但由于难以实现自动化电话中断，因此不将其纳入本工具的考虑范围内。

以上三类动作的参数各不相同。其中点击操作最为常见，单击尤甚，其参数为被点击的控件边框；输入操作的参数除了需要被输入的控件外，还需要输入的内容文本；而滑动操作则需要滑动的起始控件和终止位置；对窗口大小的调整则需要以目标大小为参数；权限授予与拒绝操作则需要一个被授予或拒绝的权限作为参数；其余操作大多没有参数。在具体实现中，考虑到动作编码的一致性，需要固定动作的参数数量与类型。因此，对于输入操作，我们仅保留其

被操作的控件，内容文本则采用随机化的方式实现；对于滑动操作，我们固定两个滑动方向：向上和向下，滑动距离设置为一个相对屏幕高度的固定百分比；对于窗口大小调整操作，其参数为相对屏幕大小的百分比，且保持窗口原宽高比不变；权限授予与拒绝操作使用随机权限，因此无需参数。由此，参数基本被固定为了被操作的控件和一个调节窗口的百分比。

对于所有这些动作，并非每个平台均能支持它们，或者说并非每个动作都对任意平台有意义。因此，在事件生成的过程中，我们还需要参考目标平台本身支持的操作来生成，以避免产生无用事件。我们收集整理了不同平台各自所支持的操作，以此限制系统生成符合目标平台事件类型的事件集合。

4.2.2 动作编码

对上述事件生成过程中产生的事件集合，还需要对其中的事件分别编码，形成对应的动作嵌入向量才能输入到执行策略模块的神经网络中。根据事件生成部分的描述，每个动作分为动作类型、被操作的控件和调节窗口的百分比，因此动作的编码也只需对这三个部分分别编码，然后将其拼接起来即可。

- 对于动作类型的编码而言，由于动作类型为枚举值，在具体实现中采用了 one-hot 编码，动作类型共 17 个，因而将其编码为了一个维数为 17 的向量。
- 对于被操作的控件而言，我们采用与图像处理模块一致的编码方式对其进行编码，即分别对其边界框、图像内容和控件类型进行编码，最后将其拼接形成控件嵌入向量。
- 对于窗口百分比而言，我们将目标平台的最大窗口大小（即屏幕大小）纳入考虑范围，以此计算出目标窗口的实际大小，并将其与该百分比值一同形成一个向量，作为最后一部分的嵌入。

以上就是动作生成模块的具体实现，在该模块，我们为当前页面生成所有可执行的动作，并对这些动作进行编码，最终形成若干动作嵌入向量以供后续处理。这里生成的动作集合与其嵌入向量集合即为本模块对外的输出，其中动作集合记录了每个动作的细节，而动作嵌入集合则输入给下文中的执行策略模块作为它的决策范围。

4.3 执行策略模块

在得到当前页面的状态嵌入和所有可执行动作的嵌入后，执行策略模块会根据这些信息计算出这些动作在当前状态下的价值，从而引导系统选择合适的动作去执行，以尽可能多地探索未知状态。

所谓执行策略，即是指一个根据当前状态给出所有动作的执行概率分布的函数。对于遍历测试而言，每一个可访问的页面或弹窗、页面或弹窗上每一个控件的内容变化都意味着当前应用状态的变化，因此其需要探索的状态空间往往较为庞大；又由于每个状态下可执行的动作集合各不相同，整个应用的动作空间往往也较大。在这种情况下，传统的利用 Q-Table 的强化学习处理起来会较为费力；而考虑到深度学习有着较强的适应性，不论对庞大的探索空间还是对较小的探索空间均有较好的拟合能力，因此，执行策略模块的核心策略采用深度神经网络的方式实现。

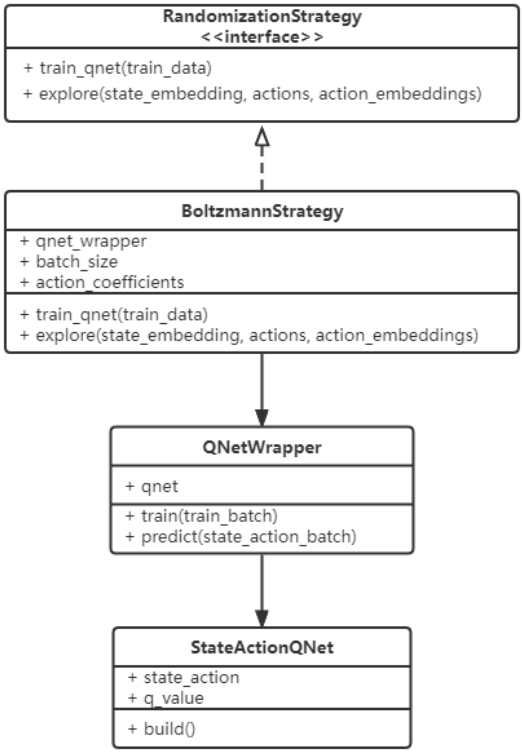


图 4.25: 执行策略模块的类图

执行策略模块的类图如图 4.25所示。RandomizationStrategy 是一个随机化策略的接口，其实现为 BoltzmannStrategy 类，该类利用 Boltzmann 策略 [44] 来实现探索随机化。StateActionQNet 类为核心神经网络，负责根据当前状态，计算每个给定动作的 Q 值（即状态-动作价值）；QNetWrapper 类封装了 StateActionQNet 类，在其基础上提供了对网络参数的训练和利用网络批量预测的功能，为 BoltzmannStrategy 类提供了核心探索策略和相应神经网络的训练方法。

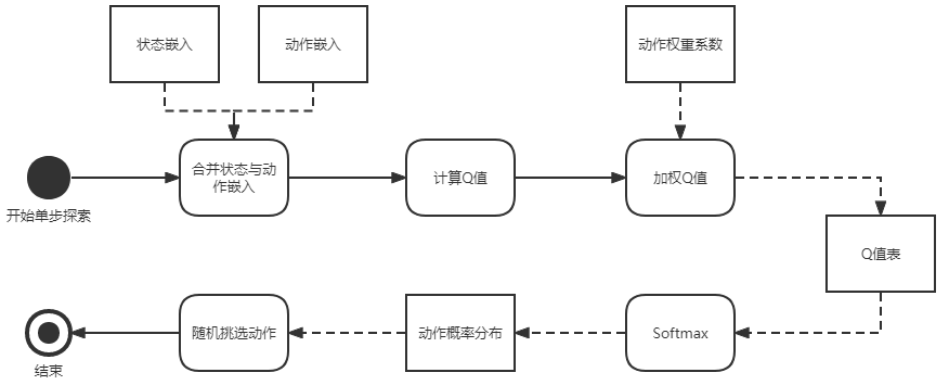


图 4.26: 执行策略模块的活动图

执行策略模块的活动图如图 4.26所示。对于一次单步探索，首先将当前的状态嵌入和每一个动作嵌入结合，接着应用 QNet 计算每一个动作的 Q 值，然后应用预先定义的动作权重系数对所有动作的 Q 值根据类型进行加权，得到加权后的动作 Q 值表后，利用 Softmax 函数根据 Q 值表计算每个动作的执行概率，最后根据该执行概率随机挑选一个动作作为下一个准备执行的动作返回。

4.3.1 核心神经网络

执行策略模块的核心神经网络实现如图 4.27所示。其本质为若干层全连接网络。

此处采用全连接网络而非更为复杂的卷积神经网络或循环神经网络，其原因在于全连接网络拟合速度较快，其它网络结构拟合速度较慢，在强化学习中，该神经网络需要随着对未知空间的探索过程不断读取新的训练数据并不断训练，若拟合速度过慢，会严重影响到整个系统的运行效率。全连接网络在保证一定的拟合能力的同时，其较高的拟合速度能使系统运行效率维持在良好的水平。

在具体实现的代码中：

- 变量 state_action_size 表示状态嵌入与一个动作嵌入拼接形成的张量，是该网络的输入张量；
- 该网络共包含 4 层输出维度为 512 的隐藏层，这些参数均可调节；
- 最后输出层输出一个浮点类型的常量，其意义为该 < 状态，动作 > 对的价值；
- 为提高拟合能力，网络中并未加入正则化操作；

- 损失函数的定义为该输出数值与预期数值（即 ground truth）的均方误差。

该网络的训练数据由回放记录模块提供；考虑到运行效率，随着遍历测试的进行该网络会不断进行小批量训练，而非进行大批量训练；关于该网络训练所需预期数值的计算，在下文的回放记录模块中会做详细介绍，因此不在本节加以介绍。

```
class StateActionQNet:
    def __init__(self, state_action_size, hidden_size=512, num_layers=4,
        activation_fn=tf.nn.leaky_relu, mode=ModelMode.train, max_value=5):
        state_action = tf.placeholder(dtype=tf.float32, shape=[None,
            state_action_size])
        expected_value = tf.placeholder(dtype=tf.float32, shape=[None])
        hidden_layer = state_action
        for i in range(num_layers):
            with tf.variable_scope(f'hidden_{i}'):
                hidden_layer = tf.layers.dense(hidden_layer, hidden_size,
                    activation=activation_fn)
        value = tf.layers.dense(hidden_layer, 1, activation=None)
        value = tf.squeeze(value, axis=1)
        if mode == ModelMode.train:
            batch_size = tf.cast(tf.shape(state_action)[0], dtype=tf.float32)
            self.loss = tf.reduce_sum(tf.square(expected_value - value)) / batch_size
        self.state_action = state_action
        self.expected_value = expected_value
        self.value = value
```

图 4.27: 策略模块核心神经网络的实现代码

4.3.2 随机化策略

在得到每个 < 状态，动作 > 对的价值后，我们就得到了在当前状态下的动作执行概率分布。理论而言，执行最高价值对应的动作就是系统当前的最优解，但是若总是执行最优解，则该系统总是在利用已有的知识进行探索，即总是会在已知范围内探索，而难以探索到未知空间，这就是强化学习中的利用-探索困境。为了解决该问题，此处采用 Boltzmann 策略，在已知动作的概率分布后，根据这个概率分布随机挑选一个动作来执行。通过 Boltzmann 策略，系统在每个状态下都有一定概率去探索当前未知的空间，从而获得更多的对探索空间的知识，

以更好地计算接下来的动作执行概率分布来引导探索。

Boltzmann 探索策略的具体实现代码如图 4.28所示。此处省略了部分初始化的代码。其整体流程如下：首先将当前状态嵌入与所有动作嵌入拼接，形成所有的状态-动作嵌入，接着将其输入该模块的核心神经网络中得到每个动作的价值；此处我们引入了动作权重，权重越高的动作被执行的概率越大，这些权重可以手动调整，通常，不同动作类型的权重均相等；对动作价值进行加权操作后，使用公式 $P(x|X) = \frac{e^x}{\sum_{a \in X} e^a}$ 得到每个动作的概率分布；最后根据该分布随机挑选动作返回即可。

```
class BoltzmannStrategy:
    def explore(self, state_embedding, actions, action_embeddings):
        num_actions = len(actions)
        state_embeddings = np.tile(state_embedding, (num_actions, 1))
        state_actions = np.concatenate([state_embeddings, action_embeddings],
            axis=1)
        scores = qnet_wrapper.predict(state_actions, batch_size=batch_size)

        scores -= scores.min()
        action_weights = np.array(self.action_type_coefficients)[[a[0] for a in
            actions]]
        if action_weights is not None:
            scores *= action_weights

        exps = np.exp(scores)
        probs = exps / np.sum(exps)
        r = random.random()
        k = 0
        for i, p in enumerate(probs):
            k += p
            if r < k:
                return actions[i], action_embeddings[i]
        logger.warn(f'Random number >= 1 not supposed to happen: {r}')
        return actions[-1], action_embeddings[-1]
```

图 4.28: Boltzmann 策略的实现代码

4.4 即时奖励模块

即时奖励模块涉及到奖励函数的设计、页面相似度计算和最终即时奖励的计算。下面首先介绍奖励函数的设计，然后给出类图和活动图以对页面相似度和即时奖励计算时需要使用到的类及相关计算流程做出概述，最后详细介绍这两个数值计算的具体实现。

4.4.1 奖励函数的设计

在每次状态转移（即 < 旧状态, 动作, 新状态 > 三元组）发生后，系统需要根据本次状态转移，给出一个合适的奖励值，以此表示本次转移的意义，奖励值越高表明本次转移意义越大，通过强化学习的过程，系统便能学到正确的状态价值与动作价值。遍历测试的目标通常是提高其对待测应用各种不同状态的覆盖率，为了提高覆盖率，需要设计一个良好的即时奖励函数以引导系统探索未知空间。除此以外，还要应对系统陷于局部最优解而不继续探索的问题。因此，我们的奖励函数设计如下：

假设时刻 t 的状态转移对应的页面转移为 $< p_t, a_t, p_{t+1} >$，其中 p_t 指代 t 时刻的页面，a_t 指代 t 时刻执行的动作；记该页面转移发生的次数为 N_t，记页面 p_{t+1} 中所有可执行的动作数量为 m_{t+1}，所有已执行的动作数量为 n_{t+1}，则定义页面 p_{t+1} 的探索度为：

$$e_{t+1} = \frac{n_{t+1}}{m_{t+1}}$$

而基于该探索度，我们给出对本次状态转移的即时奖励为：

$$r_t = \frac{1 - e_{t+1}}{\sqrt{N_t}} = \frac{1 - \frac{n_{t+1}}{m_{t+1}}}{\sqrt{N_t}}$$

在具体实现中，我们以上述奖励为核心奖励，其取值范围为 $[0, 1]$；考虑到避免让系统总是在相同页面中做无效操作，需要对不发生页面变化的状态转移添加惩罚因子；为了让系统能更快的从奖励中学到探索空间的知识，对惩罚后的奖励添加了激活因子，该因子能使奖励值突破 $[0, 1]$ 的范围，从而一定程度上提高系统的学习效率。

4.4.2 类图

接下来介绍本模块的详细设计。即时奖励模块的类图如图 4.29所示。该模块整体分为以下几个类：

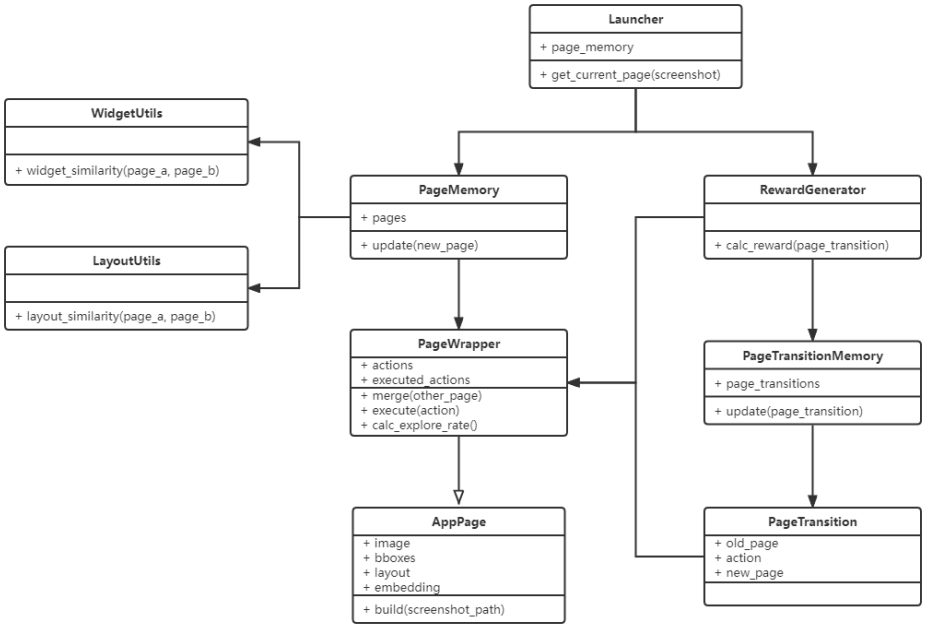


图 4.29: 即时奖励模块的类图

- PageWrapper 类。该类是对图像处理模块中 AppPage 类的包装，在其基础上添加了相似页面的融合功能和页面探索度的计算方法。相似页面是指被下面的 PageMemory 类识别为同一页面的两个页面，因为二者相似度较高，因此需要加以合并，将二者的控件集合、已执行动作集合和可执行动作集合分别进行合并。通过相似页面的融合，页面探索度的计算才能更准确。

- PageMemory 类。该类负责区分不同的页面，并利用 PageWrapper 提供的功能将相似的页面加以融合。其区分不同页面的依据主要是两个页面的控件相似度和布局相似度，若相似度较低则将其看作不同页面；反之则为相同页面，需要加以融合。区分不同页面的功能主要由算法端启动类 Launcher 使用，以维护一个待测应用在测试过程中遇到的所有页面集合。

- PageTransitionMemory 类。该类本质上是一个计数器，负责记录每个页面转移的次数，每个页面转移都是一个 PageTransition 对象，包含旧页面、动作和新页面三个要素。

- Reward 类。该类负责计算核心奖励、惩罚因子和激活因子，其具体算法会在后文详细介绍。

- WidgetUtils 类和 LayoutUtils 类。这两个工具类负责计算页面的控件相似度和布局相似度，作为 PageMemory 类区别不同页面的依据。

4.4.3 活动图

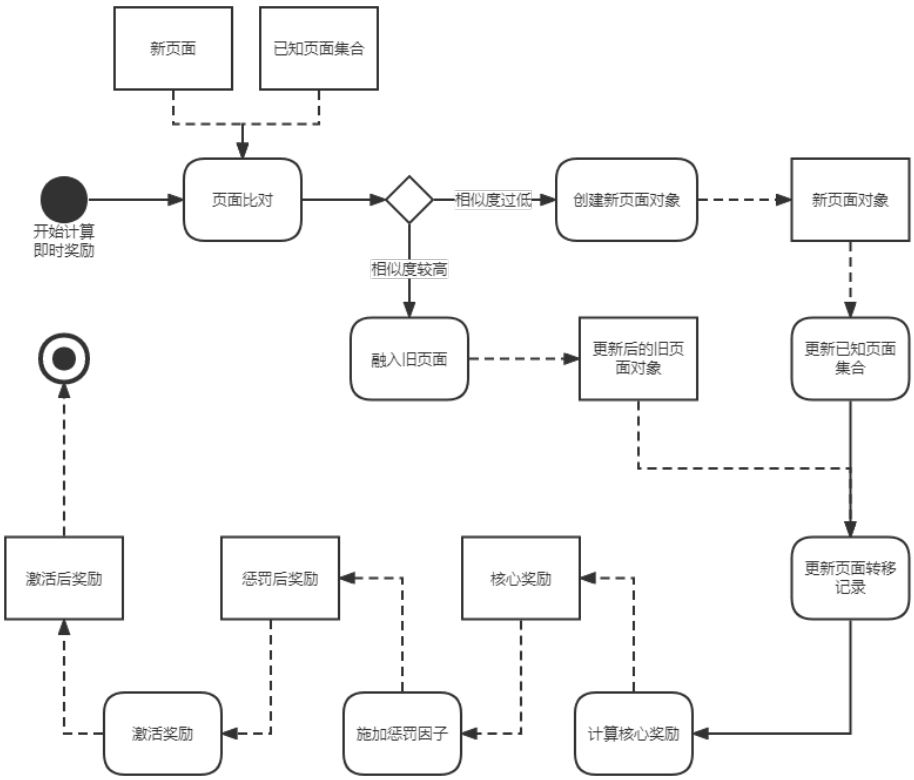


图 4.30: 即时奖励模块的活动图

即时奖励模块的活动图如图 4.30所示。在计算一次状态转移时，首先会根据新页面和当前系统已知的页面集合（PageMemory 中记录的集合）尝试进行逐个页面比对，其目的是找到一个与新页面最为相似的页面以确认新页面属于已知页面，若能找到这样的已知页面，则将新页面的数据融入找到的已知页面；反之则构建一个新的页面对象，并将其加入已知页面集合中。以上过程结束后即可构建出一个页面转移 < 旧页面，动作，新页面 >，其中“新页面”指的是融合后的已知页面或新创建的页面对象。将该页面转移更新到页面转移记录中后，便可开始计算本次状态转移的即时奖励了。该即时奖励计算分三个部分，分别是核心奖励、惩罚因子和激活因子，通过前面的奖励函数计算出核心奖励后，乘以惩罚因子，再应用激活因子，最终得到本次状态转移的即时奖励。

4.4.4 页面相似度的计算

核心奖励部分的两个重要数据为新页面探索度和页面转移次数，而“页面”是需要基于图像信息来加以界定的。在图像处理模块中，我们已经介绍了一个截屏中的布局信息和控件信息，而正是这两个信息决定了一个页面。因此，在界定页面时，我们同样利用这两个信息，根据两个页面的布局相似度和控件相似度的加权平均数来判断这两个页面是否为同一个页面，若相似度较高，则可以认为这两个页面为相同的页面。下面分别介绍布局相似度和控件相似度的计算。

布局相似度。在图像处理模块中，布局是由一个树状结构表示的，与其中的布局嵌入类似，此处我们同样使用其纯结构的控件树，忽略每个控件的内容，因为控件内容在控件相似度的计算中考虑。两个页面的布局相似度可以由这两个页面的纯结构控件树的编辑距离决定。与字符串的编辑距离类似，树的编辑距离也是根据从一棵树转变到另一棵树所需要的节点添加、节点修改与节点删除这三类操作的加权次数和决定的。记页面 p_a 的纯结构控件树为 T_a，页面 p_b 的纯结构控件树为 T_b，T_a 与 T_b 的编辑距离为 $d(a,b)$，记树 T_a 的节点数量为 n_a，T_b 的节点数量为 n_b，则可以将 p_a 与 p_b 的布局相似度定义为：

$$s_l(p_a, p_b) = 1 - \frac{d(a,b)}{max(n_a, n_b)}$$

其中的分母 $max(n_a, n_b)$ 为 $d(a,b)$ 的理论最大值，其目的是将相似度的取值范围限定至 $[0,1]$ 区间内。

控件相似度。在图像处理模块中，当前页面的所有控件均已被提取出来，控件相似度的计算就利用两个页面 p_a 和 p_b 各自的控件边界框集合进行匹配，并计算这些匹配控件对的平均距离，再基于该平均距离定义二者的控件相似度。具体流程如下：

1. 匹配控件。对两个页面的控件边界框利用 IOU 指标进行一一匹配，允许存在一些控件找不到合适的匹配。

2. 计算匹配控件对的距离。对第一步中每一对匹配好的控件对，利用卷积神经网络分别加以嵌入，求出嵌入向量的距离作为这一对匹配的距离。

3. 计算不匹配控件的长度。对第一步中每一个 p_a 中找不到匹配的控件，同样利用卷积神经网络加以嵌入，直接将该嵌入向量作为距离（即与原点的距离）。

4. 求平均距离。上述第二、第三步的每一个距离均需要被截断到 $[0,1]$ 之间，然后求出这些距离的平均距离 $d(a,b)$。

5. 求控件相似度。可以直接将 p_a 对 p_b 的相似度定义为 $s_w(a,b) = 1 - d(a,b)$。

```
def widget_similarity(src_image, src_bboxes, tgt_image, tgt_bboxes,
    bbox_threshold=.7):
    matches = widget_bbox_match(src_bboxes, tgt_bboxes, bbox_threshold)
    if matches:
        matched_src_bboxes = [src_bboxes[i] for i, _ in matches]
        matched_src_widgets = [src_image[y:y+h, x:x+w, :] for x, y, w, h in
            matched_src_bboxes]
        matched_src_embeddings = embed_img(matched_src_widgets)
        matched_tgt_bboxes = [tgt_bboxes[j] for _, j in matches]
        matched_tgt_widgets = [tgt_image[y:y+h, x:x+w, :] for x, y, w, h in
            matched_tgt_bboxes]
        matched_tgt_embeddings = embed_img(matched_tgt_widgets)
        matching_distance =
            np.sum(np.clip(np.sqrt(np.sum((matched_src_embeddings -
            matched_tgt_embeddings) ** 2, axis=1)), 0, 1))
    else:
        matching_distance = 0

    remaining_bboxes = [src_bboxes[i] for i in set(j for j in range(len(src_bboxes)))
        - set(j for j, _ in matches)]
    if remaining_bboxes:
        remaining_widgets = [src_image[y:y+h, x:x+w, :] for x, y, w, h in
            remaining_bboxes]
        remaining_embeddings = embed_img(remaining_widgets)
        remaining_distance = np.sum(np.clip(np.sqrt(np.sum(remaining_embeddings
            ** 2, axis=1)), 0, 1))
    else:
        remaining_distance = 0
    sim = 1 - (matching_distance + remaining_distance) / len(src_bboxes)
    return sim, matches
```

图 4.31: 计算控件相似度的实现代码

控件相似度计算的具体实现代码如图 4.31所示。其中变量 matches 是一个

序号对的列表，其中的每个序号对 (i, j) 表示 src_bboxes[i] 与 tgt_bboxes[j] 相匹配。代码中 embed_img(widgets) 函数是一个对给定控件的图像集合进行批量嵌入的函数，其底层利用了 VGG16 网络来实现对每一个图像的嵌入。

对两个页面的布局相似度和控件相似度进行加权平均即可得到这两个页面的相似度，接着利用预先指定的相似度阈值即可判断这两个页面是否为同一个页面。若为相同页面，则会利用 PageWrapper 类提供的方法对二者加以合并，反之则将其作为一个新页面加入 PageMemory 中。更新了 PageWrapper 与 PageMemory 后，再把本次页面转移加入 PageTransitionMemory，更新其次数。接下来，所有数据均已更新完成，就可以计算本次状态转移的即时奖励了。

4.4.5 即时奖励的计算

即时奖励的计算主要分为核心奖励、惩罚因子和激活因子三个部分，由核心奖励与惩罚因子相乘得到惩罚后奖励，接着应用激活因子得到最终的激活后奖励作为状态转移的奖励。

```
def generate_reward(old_page, action, new_page, new_page_confidence, ptm,
                    punishment_coefficient=3):
    # Core reward.
    er = new_page.calc_explore_rate()
    ptc = ptm.count(old_page, action, new_page)
    core_reward = (1 - er) / math.sqrt(ptc)
    # Punishment.
    if old_page == new_page:
        punished_reward = (1 - new_page_confidence) * punishment_coefficient *
            core_reward
    else:
        punished_reward = core_reward
    # Activation.
    if punished_reward > .5:
        reward = 5 * (math.exp(punished_reward - .5) - 1)
    else:
        reward = 5 * (1 - math.exp(.5 - punished_reward))
    return reward
```

图 4.32: 计算即时奖励的实现代码

即时奖励的具体实现代码如图 4.32所示。代码中 ptm 为一个 PageTransition-

Memory 的实例对象，即一个页面转移计数器。代码中 new_page_confidence 为新页面的置信度，该置信度即为新页面与 PageMemory 中最相似的页面的相似度，若无最相似页面，则置信度为-1。存在最相似页面意味着该页面是曾经遍历过的页面，因此在计算惩罚因子这一步中，当新旧页面相同时，必定存在最相似页面，即新页面置信度必为正值，不会出现-1；当新旧页面不同时，无需加以惩罚。具体实现中的激活因子采用如下函数：

$$f(x) = 5 \times \frac{x - 0.5}{|x - 0.5|}(e^{x-0.5} - 1)$$

该函数的图像是一个以 $(0.5, 0)$ 为中心的中心对称图形，即当惩罚后奖励在 0.5 以上时，激活后奖励为正，反之则为负，且激活后奖励的绝对值与距离惩罚后奖励与 0.5 的差呈指数增长关系，这样可以使系统更快地针对相同页面与不同页面进行学习探索。该激活因子中的系数可以自由调整，系数越大则学习的越快，系统的策略变化的也越剧烈。

4.5 回放记录模块

回放记录模块负责记录每一次状态转移及其奖励，并向执行策略模块的神经网络提供训练数据。经过精心设计的即时奖励需要通过该模块去作用到执行策略模块，以引导系统更新优化其探索策略，以实现更高的覆盖率。

回放记录模块主要记录如下数据：每一次状态转移 $< s_t, a_t, s_{t+1} >$、每一个即时奖励 r_t 和新状态下的所有动作 A_{t+1}，其中 s_t、a_t 和 r_t 分别为 t 时刻的状态嵌入、动作嵌入和奖励，A_{t+1} 为状态 s_{t+1} 下的可执行动作集合。对上述数据做记录的目的是为了为执行策略模块的神经网络提供训练数据，包括 s_t、a_t 和标准 Q 值 Q^{target}。在执行策略模块中已经介绍过其神经网络采用预测值 Q 与标准值 Q^{target} 的均方误差作为损失函数，而其中标准值 Q^{target} 的计算则由回放记录模块负责，其计算公式如下：

$$Q_t^{target} = r_t + \gamma max_{a \in A_{t+1}} Q(s_{t+1}, a)$$

其中，γ 为衰减率，衰减率越大表明时间越近的奖励越重要，即系统会更多地学习近期的状态转移而更少地从早期的状态转移中学习知识。

由于随着遍历测试的不断运行，回放记录中存储的数据数量会不断增多，考虑到训练效率的问题，不能将回放记录中的所有数据作为训练数据提供给执行策略模块，而只能从中选出一部分数据来训练。为了使系统更多地从最近的状

态转移中进行学习，在挑选 n 条训练数据时，首先从最近的记录中选择 $\frac{n}{2}$ 条数据，再从剩余的记录中选择 $\frac{n}{2}$ 条数据。

提供训练数据的具体实现代码如图 4.33所示。我们并非先计算状态转移的奖励再将奖励存入回放记录中，而是直接将计算奖励所需要的页面转移存入回放记录中，在向策略模块提供训练数据时，再实时地计算奖励，进而计算 Q^{target} 值。之所以延后计算奖励是因为随着探索不断深入，许多页面的探索度都会发生变化，同时同一个页面转移的次数也会与过去不同，若提前计算即时奖励则使用的是过去的探索度和页面转移次数，从而无法很好的使系统学习到当前的探索状态。最后做的批量化操作是为了提高训练速度，加速神经网络的拟合。

```
class ReplayMemory:
    def get_train_data(self, qnet_wrapper, pred_kwargs=dict(), n=16, batch_size=4):
        data = self[self._select_train_ids(n)]
        expected_qs = []
        for d in data:
            new_action_embeddings = d[3]
            num_new_actions = new_action_embeddings.shape[0]
            new_state_embeddings = np.tile(d[2], (num_new_actions, 1))
            new_state_actions = np.concatenate([new_state_embeddings,
                new_action_embeddings], axis=1)
            scores = qnet_wrapper.predict(new_state_actions, **pred_kwargs)
            reward = generate_reward(*d[1], self.ptm)
            expected_qs.append(reward + self.gamma * np.max(scores))
        batches = []
        for i in range(0, min(n, len(data)), batch_size):
            state_actions = np.stack([d[0] for d in data[i:(i + batch_size)]])
            expected_values = np.array(expected_qs[i:(i + batch_size)])
            batches.append((state_actions, expected_values))
        return batches
```

图 4.33: 回放记录模块获取训练数据的实现代码

4.6 页面截屏模块、动作执行模块与前端界面

页面截屏模块与动作执行模块属于该系统的服务端，负责与各类平台的设备交互，主要实现截屏功能和动作生成模块中提及的各类交互事件与系统事件的触发。

在具体实现中，与安卓设备的交互完全基于 ADB 命令，通过 Python 的 subprocess 类库实现对服务器上 ADB 命令的调用；与 Web 平台的交互则基于 ChromeDriver 和 Selenium，Selenium 已经实现了一套与 Chromium 内核的浏览器交互的接口，因此在实现与 Web 平台交互时只需要将其接口进一步封装即可。

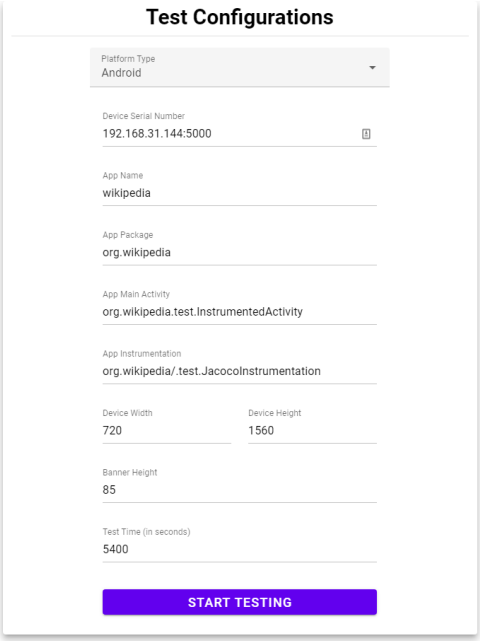


图 4.34: 前端测试配置界面

图 4.35: 前端测试监控界面

下面展示本系统的前端界面。本系统的前端主要分为测试配置界面和测试监控界面，其中测试配置界面如图 4.34所示。顶端的选择框可选安卓和 Web 平台，该平台指待测应用运行所需的平台；下面根据不同平台会有不同的配置项，

例如对于安卓平台而言，需要配置安卓设备的序列号、屏幕大小、待测应用的名称、包名、主活动名、instrumentation 名、设备顶部导航栏高度和测试时间。对于 Web 应用，则需要填写待测应用名称、url、电脑分辨率和测试时间。该页面底部为开始测试按钮，点击后就会根据相应地配置开始遍历测试。

遍历测试开始后，系统自动跳转到测试监控界面，该界面如图 4.35所示。顶部会显示当前测试状态是否结束；左侧为测试设备屏幕监控，能实时查看当前设备的屏幕；右侧为测试日志，能实时查看日志信息；当遍历测试结束后，右下方的日志下载按钮会被激活，点击该按钮可下载本次测试的完整日志。

4.7 本章小结

本章主要介绍了基于图像理解强化学习的通用遍历测试技术的详细设计及其系统的具体实现。首先给出了系统的整体架构，接着结合设计图与实现代码依次介绍系统中每个重要模块的详细设计与具体实现，包括了算法端的图像处理模块、动作生成模块、执行策略模块、即时奖励模块和回放记录模块，以及服务端的页面截屏模块和动作执行模块，最后展示了本系统的前端界面，包括测试配置界面和测试监控界面。

第五章　系统测试与实验分析

本章的主要介绍基于图像理解强化学习的通用遍历测试技术的系统测试与相关实验分析。本章主要包含功能测试和效果测试，其目的是为了验证系统的可用性和有效性。

5.1　测试准备

测试之前首先需要将系统的前中算法端分别进行部署。部署时，算法端被部署于一台 Linux 服务器上，而前端和服务端均被部署于同一台 Windows 系统的计算机上。

算法端服务器环境信息如下表 5.1所示：

表 5.1: 算法端服务器环境信息

信息字段	具体信息
服务器 CPU	Intel(R) Core(TM) i7-6850K CPU @ 3.60GHz * 12
服务器内存大小	62GB
服务器硬盘空间	915GB
服务器 GPU	Nvidia GeForce GTX 1080Ti * 2
操作系统	Ubuntu 20.04.2 LTS
Python 库	python==3.7.0 tensorflow-gpu==1.13.1 opencv-python==4.2.0.32 keras==2.3.1 Flask==2.0.2 baidu-aip==2.2.18.0 apted==1.0.3

前端和服务端所部署于的计算机环境如下表 5.2所示：

5.2　功能测试

接下来介绍对本系统的功能测试。功能测试基于需求分析得到的功能性需求列表，针对其中的每个功能性需求检测本系统的完整性和可用性。下面依次针对每个功能性需求编写相应地测试用例，最后给出测试执行结果以证明本系统的完整可用性。

表 5.2: 前、服务端计算机环境信息

信息字段	具体信息
CPU 信息	Intel(R) Core(TM) i5-5200 CPU @ 2.20GHz * 2
内存大小	8GB
硬盘空间	157G
操作系统	Windows10
服务端依赖的 Python 库	python==3.7.0 Flask==2.0.3 Flask-Cors==3.0.10 selenium==4.1.0
Chrome 版本	99.0.4844.74 (Official Build) (64-bit)
ChromeDriver 版本	98.0.4758.80
前端依赖的 node 版本	16.13.0
前端依赖的 npm 库	vue@3.2.31 vue-router@4.0.13 vuetify@3.0.0-alpha.0 vuex@4.0.2 axios@0.26.0

表 5.3是安卓应用测试配置功能的测试用例设计表，主要测试系统能否正确地进行安卓应用遍历测试的配置，以及能否对用户输入的配置信息进行检查。

表 5.3: 安卓应用测试配置功能的测试用例设计

测试用例 ID	**TC1**
测试名称	安卓应用测试配置功能测试
测试功能	系统是否能配置及检查安卓平台的测试参数
测试步骤	1. 用户在平台选择栏中选择安卓平台 2. 填写安卓各项配置，包括设备序列号、设备屏幕大小、设备顶部横幅高度、待测应用名称、包名、instrumentation 名和测试时间 3. 点击提交 4. 用户少填任意一项配置项并点击提交 5. 用户填错任意一项配置项的格式并点击提交
预期结果	1. 界面的配置项显示齐全，没有缺漏 2. 正确的配置信息可以提交成功 3. 缺少配置项或错误配置项均会提交失败

表 5.4是 Web 应用测试配置功能的测试用例设计表，主要测试系统能否正确地进行 Web 应用遍历测试的配置，以及能否对用户输入的配置信息进行检查。该用例与 TC1 的主要区别在于其配置项种类与安卓平台的不同，除此以外，整

体流程相似。

表 5.4: Web 应用测试配置功能的测试用例设计

测试用例 ID	TC2
测试名称	Web 应用测试配置功能测试
测试功能	系统是否能配置及检查 Web 平台的测试参数
测试步骤	1. 用户在平台选择栏中选择 Web 平台 2. 填写 Web 各项配置，包括设备屏幕大小、待测应用名称、待测应用主页 url、测试时间 3. 点击提交 4. 用户少填任意一项配置项并点击提交 5. 用户填错任意一项配置项的格式并点击提交
预期结果	1. 界面的配置项显示齐全，没有缺漏 2. 正确的配置信息可以提交成功 3. 缺少配置项或错误配置项均会提交失败

表 5.5是安卓应用遍历测试功能的测试用例设计表，主要测试系统能够根据配置信息找到相应的安卓设备并启动对待测安卓应用的遍历测试，同时不会对其它无关设备造成任何影响。在开始测试前，需要在测试设备上提前安装好待测应用。

表 5.5: 安卓应用遍历测试功能的测试用例设计

测试用例 ID	TC3
测试名称	安卓应用遍历测试功能测试
测试功能	系统能否根据配置信息启动安卓应用的遍历测试
测试步骤	1. 用户在平台选择栏中选择安卓平台并填入正确的配置信息 2. 用户将相关设备连入计算机 3. 用户点击开始遍历测试 4. 用户查看相应的安卓设备，检查是否已开始对相关应用的遍历测试 5. 用户查看其它安卓设备，检查是否受到影响
预期结果	1. 相应的安卓设备上，相关应用的遍历测试已经开始 2. 其它安卓设备并未受到影响

表 5.6是 Web 应用遍历测试功能的测试用例设计表，主要测试系统能够根据配置信息在计算机上启动对待测 Web 应用的遍历测试。该测试用例与 TC3 的主要区别在于 Web 应用无需连接设备，直接在服务端所部署的计算机上测试。该用例同样需要在该计算机上提前将待测应用安装完成。

表 5.7是设备屏幕监控功能的测试用例设计表，主要测试系统对测试设备屏幕的监控功能是否具有实时性，以及是否能持续监控到本次遍历测试的结束。

表 5.6: Web 应用遍历测试功能的测试用例设计

测试用例 ID	TC4
测试名称	Web 应用遍历测试功能测试
测试功能	系统能否根据配置信息启动 Web 应用的遍历测试
测试步骤	1. 用户在平台选择栏中选择 Web 平台并填入正确的配置信息 2. 用户点击开始遍历测试 3. 用户查看服务端所部署的计算机，检查是否已开始对相关应用的遍历测试
预期结果	1. 服务端所部署的计算机上对待测 Web 应用的遍历测试已开始

表 5.7: 设备屏幕监控功能的测试用例设计

测试用例 ID	TC5
测试名称	设备屏幕监控功能测试
测试功能	系统能否在测试进行过程中持续监控测试设备的屏幕直至测试结束
测试步骤	1. 用户开始任意平台的遍历测试 2. 用户查看测试详细信息页面的屏幕监控部分
预期结果	1. 测试详细信息页面的屏幕监控部分能不断显示当前最新的测试设备屏幕图像 2. 该监控功能持续到遍历测试结束，不会中途断开

表 5.8是测试日志监控功能的测试用例设计表，主要测试系统对测试日志的监控功能是否具有实时性，以及能否持续监控到本次遍历测试的结束。

表 5.8: 测试日志监控功能的测试用例设计

测试用例 ID	TC6
测试名称	测试日志监控功能测试
测试功能	系统能否在测试进行过程中持续监控测试日志信息直至测试结束
测试步骤	1. 用户开始任意平台的遍历测试 2. 用户查看测试详细信息页面的测试日志部分
预期结果	1. 测试详细信息页面的测试日志部分能不断显示当前最新的测试测试日志 2. 该监控功能持续到遍历测试结束，不会中途断开

表 5.9是下载日志文件功能的测试用例设计表，主要测试系统能否在测试结束后下载本次测试产生的服务端的日志文件，该日志文件可以协助测试人员复盘本次遍历测试。

严格按照上述功能测试用例描述执行测试后，最终的执行结果如表 5.10所示，每个测试用例均通过。由此得证，本系统实现了每一个功能性需求，覆盖了每一个功能点，具有一定的可用性。

表 5.9: 下载日志文件功能的测试用例设计

测试用例 ID	**TC7**
测试名称	下载日志文件功能测试
测试功能	系统能否在测试结束后下载本次测试产生的日志文件
测试步骤	1. 遍历测试结束后用户点击下载日志按钮 2. 用户选择本地的合适位置进行日志下载
预期结果	1. 遍历测试结束后下载日志按钮可操作 2. 日志能完整地被下载到本地

表 5.10: 功能测试结果

测试用例 ID	**对应功能性需求 ID**	**测试结果**
TC1	R1	通过
TC2	R2	通过
TC3	R3	通过
TC4	R4	通过
TC5	R5	通过
TC6	R6	通过
TC7	R7	通过

5.3 实验评估

本节主要介绍针对本技术的效果评估而设计的实验及其结果。下面分别从实验设计、实验执行和实验结果三个方面介绍实验评估。

5.3.1 实验设计

实验设计部分主要由待测应用集、对比工具、评估指标和实验方法四个部分组成。

待测应用集。由于本技术属于平台通用的遍历测试技术，因此待测应用集是由安卓和 Web 两个不同平台的应用组成，其中安卓应用有 15 个，Web 应用有 5 个，共计 20 个待测应用。

安卓待测应用如下表5.11所示，其中应用均为开源应用，其源代码均在 Github 上，因此其 Github 上的 star 数量（数据截至本章编写为止）可表示该应用的受欢迎程度；这些待测应用共覆盖了笔记、电池管理、理财等 12 个不同的类别，其总有效代码行数从数百到上万均有覆盖，总分支数同样如此，受欢迎程度从 0 到上千的 star 数量均有覆盖。

Web 待测应用如下表5.12所示，覆盖了团队管理、博客、理财、时间管理和

表 5.11: 安卓应用列表

应用名称	应用类型	总有效代码行数	总分支数	Github star 数量
AnkiDroid	笔记	35655	14206	4.8k
BetterBatteryStats	电池管理	9921	3437	486
BudgetWatch	理财	2097	676	0
ConnectBot	远程连接	8909	3689	1.8k
DumbPhoneAssistant	通讯录	524	116	24
KeepassDroid	密码	9182	3579	1.3k
Man-Man	命令手册	2440	2011	36
OmniNotes	笔记	9563	3444	7
OpenBikeSharing	交通	1202	383	66
PasswordMaker	密码	1900	453	16
PrivacyFriendlyNotes	笔记	1993	510	62
RedReader	阅读	17746	6901	1.3k
RunnerUp	运动	13761	5522	601
Vanilla	音乐	10598	4827	890
Wikipedia	百科	34353	17295	1.6k

社交 5 个类别，有效代码行数同样从数十到上万，总分支数和受欢迎程度也多寡兼俱。

表 5.12: Web 应用列表

应用名称	应用类型	总有效代码行数	总分支数	Github star 数量
Gadael	团队管理	25468	20867	52
Mean-Blog	博客	39	8	156
Splittypie	理财	2008	477	118
Timeoff-Management	时间管理	3729	5092	725
Webogram	社交	89596	36793	7.6k

对比工具。工业界最常用的遍历测试工具为 Monkey，即基于随机探索策略的遍历测试工具，这是本实验的一个重要对比工具。除此以外，本实验还考虑了其它基于强化学习的遍历测试工具，包括：适用于安卓应用遍历测试的 Q-testing，以及适用于 Web 应用遍历测试的 WebExplor。由于 Q-testing 和 WebExplor 均未开放源码，因此，本文所使用的这两个工具是基于二者论文所述方法而复现的工具，这可能会在一定程度上影响本实验的有效性。

评估指标。遍历测试的目标是尽可能覆盖待测应用的更多功能场景，因此常见的指标是有效代码行覆盖率和分支覆盖率，因为对源代码的覆盖率能较好地反映对功能场景的覆盖程度。除此以外，在运行效率方面，常见的指标是事件

生成效率，即单位时间内生成并执行的事件数量，由于遍历测试的过程是不停地生成一个又一个的事件并执行，所以事件生成效率越高，一定时间内生成的事件数量就越多，能覆盖到的功能场景也可能越多。

对于有效代码行覆盖率和分支覆盖率，需要考虑两个不同的角度：时间角度和事件角度，前者指固定时间内能达到的覆盖率，后者指固定事件数量的条件下能达到的覆盖率。时间角度的覆盖率反映了当前实验条件下能到达的实际效果，而事件角度的覆盖率则反映了在不考虑每个事件消耗时间的条件下，该方法的理论效果。实际效果总是会受到客观运行条件的制约、具体技术实现的限制，而理论效果则无关这些因素，随着运行条件和实现方式的优化升级，实际效果会逐步接近理论效果。

综上所述，本实验采取的评估指标包含：

1. 时间角度的有效代码行覆盖率和分支覆盖率。在 90 分钟内各不同遍历测试工具在各个应用上能达到的有效代码行覆盖率和分支覆盖率。
2. 事件角度的有效代码行覆盖率和分支覆盖率。在 500 个事件的限制内各不同遍历测试工具在各个应用上能达到的有效代码行覆盖率和分支覆盖率。
3. 事件生成效率。单位时间内生成并执行的事件数量。

实验方法。对所有待测安卓应用，分别使用 Monkey、UniRLTest 和 Q-testing 进行 90 分钟和 500 个事件的遍历测试，并统计每次测试的 90 分钟后和 500 个事件后的有效代码行覆盖率和分支覆盖率；对所有待测 Web 应用，分别使用 Monkey、UniRLTest 和 WebExplor 进行 90 分钟和 500 个事件的遍历测试，并统计每次测试的 90 分钟后和 500 个事件后的有效代码行覆盖率和分支覆盖率。事件生成效率可以根据上述 90 分钟固定时间测试的事件总数经过简单计算得到。每次实验后，都要删除本次实验测试设备中待测应用的所有数据，使下次在该设备上对同一个应用的遍历测试与本次测试的初始状态相同。

5.3.2 实验执行与结果分析

将 UniRLTest 的后端正确部署到服务器上，将其服务端也正确部署到对应的计算机上，然后将其它对比工具也部署到同一台计算机上后，即可按照预定的实验方法开始执行。

安卓应用的 90 分钟覆盖率数据如下表 5.13所示，其中待测应用按有效代码行数升序排序，M 指代 Monkey 方法，Q 指代 Q-testing 方法，U 指代本文提出的 UniRLTest 方法，加粗的数据为三个工具中最优的结果。从表中数据可以看

出，不论有效代码行的多少，UniRLTest 方法在多数情况下都能跑出优于其它两个对比工具的覆盖率效果。当有效代码行数在一万行以内时，该方法的效果明显优于其它方法。

表 5.13: 安卓应用 90min 覆盖率

应用名称	有效代码行数	有效代码行覆盖率 (%)			分支数	分支覆盖率 (%)		
		M	Q	U		M	Q	U
DumbPhoneAssistant	524	37.79	36.45	**40.84**	116	18.97	18.97	**20.69**
OpenBikeSharing	1202	29.37	38.94	**69.13**	383	14.36	25.59	**44.91**
PasswordMaker	1900	31.68	26.16	**36.26**	453	**26.71**	18.54	24.94
PrivacyFriendlyNotes	1993	46.86	22.63	**52.03**	510	37.45	11.96	**40**
BudgetWatch	2097	**44.02**	13.59	41.39	676	**22.78**	5.62	21.15
Man-Man	2440	33.98	15.08	**37.54**	2011	17.21	5.67	**19**
ConnectBot	8909	10.67	11.81	**13.77**	3689	4.74	5.53	**5.56**
KeepassDroid	9182	2.78	1.23	**10.01**	3579	1.15	0.45	**4.47**
OmniNotes	9563	21.82	26.54	**30.59**	3444	14.23	19.51	**23.03**
BetterBatteryStats	9921	28.64	27.96	**29.05**	3437	21.94	21.79	**23.63**
Vanilla	10598	37.14	29.88	**41.21**	4827	25.19	18.98	**30.21**
RunnerUp	13761	13.98	10.17	**18.87**	5522	6.94	4.93	**9.24**
RedReader	17746	**20.56**	14.89	19.51	6901	**13.75**	9.4	12.97
Wikipedia	34353	6.1	13.43	**29.15**	17295	2.86	7.12	**12.68**
AnkiDroid	35655	**26.29**	8.67	23.3	14206	**17.58**	5.05	12.87

安卓应用 500 个事件的覆盖率如下表 5.14所示，从表中数据可知，该方法在事件角度的效果依然与 Monkey 和 Q-testing 相比较优。即便对于效果较差的应用，其与最佳覆盖的差距依旧不大；而对于部分应用，如 OpenBikeSharing、PrivacyFriendlyNotes 等，该方法能达到的事件角度的覆盖率大幅超过了其它方法。整体来看，UniRLTest 方法对于移动应用效果较优。

Web 应用 90 分钟的覆盖率如表 5.15所示，500 个事件的覆盖率如表 5.16所示。从表中可以看出，对于 Mean-Blog 这类应用，由于其主要代码逻辑十分容易被覆盖，其余代码逻辑则入口较深，难以被覆盖，因此不同方法的效果基本一致（表 5.16中 Monkey 较好的效果是不稳定的）；而对于其它应用，从有效代码行覆盖率的角度看，UniRLTest 仍然具有一定优势，但与其它方法相比差距并不大，从分支覆盖率的角度看，该方法效果略逊一筹，但同样差距不大。

表 5.14: 安卓应用 500 个事件的覆盖率

应用名称	有效代码行数	有效代码行覆盖率 (%)			分支数	分支覆盖率 (%)		
		M	Q	U		M	Q	U
DumbPhoneAssistant	524	**37.79**	33.59	36.64	116	18.97	**21.55**	18.1
OpenBikeSharing	1202	17.55	10.23	**68.14**	383	8.09	4.7	**44.13**
PasswordMaker	1900	**28.11**	25.84	25.37	453	**22.08**	16.56	17.44
PrivacyFriendlyNotes	1993	20.62	16.81	**51.53**	510	13.73	8.63	**39.02**
BudgetWatch	2097	23.41	26.37	**38.48**	676	9.91	11.69	**18.64**
Man-Man	2440	**32.13**	29.75	26.84	2011	**16.21**	11.98	11.14
ConnectBot	8909	8.83	**11.47**	8.82	3689	3.82	**5.18**	3.82
KeepassDroid	9182	2.59	2.79	**10.01**	3579	1.06	1.56	**4.47**
OmniNotes	9563	19.72	19.06	**30.27**	3444	12.6	13.24	**22.88**
BetterBatteryStats	9921	**19.25**	13.4	15.49	3437	**13.79**	9.98	11.23
Vanilla	10598	**34.35**	29.34	34.07	4827	22.06	18.62	**22.73**
RunnerUp	13761	13.92	12.75	**18.87**	5522	6.88	6.41	**9.24**
RedReader	17746	12	10.92	**14.44**	6901	7.6	7.25	**9.11**
Wikipedia	34353	5.98	24.24	**26.87**	17295	2.73	9.29	**11.29**
AnkiDroid	35655	6.74	**21.78**	15.75	14206	3.93	**13.91**	9.04

表 5.15: Web 应用 90min 覆盖率

应用名称	有效代码行数	有效代码行覆盖率 (%)			分支数	分支覆盖率 (%)		
		M	W	U		M	W	U
Mean-Blog	39	41.02	41.02	41.02	8	0	0	0
Splittypie	2008	36.25	26.89	**37.5**	477	**15.09**	7.75	9.88
Timeoff-Management	3729	22.76	**26.46**	26.31	5092	16.59	**20.07**	19.85
Gadael	25468	**29.77**	19.61	24.62	20867	**16.92**	9.79	13.51
Webogram	89596	11.83	13.3	**17.57**	36793	14.5	16.38	**23**

表 5.16: Web 应用 500 个事件的覆盖率

应用名称	有效代码行数	有效代码行覆盖率 (%)			分支数	分支覆盖率 (%)		
		M	W	U		M	W	U
Mean-Blog	39	**61.54**	41.02	41.02	8	**25**	0	0
Splittypie	2008	35	26.89	**36.25**	477	14.46	7.75	**15.09**
Timeoff-Management	3729	**26.44**	23.41	26.3	5092	**20.01**	17.55	19.85
Gadael	25468	**24.5**	6.47	23.99	20867	**13.42**	1.78	13.03
Webogram	89596	15.93	9.88	**17.62**	36793	20.39	11.4	**22.93**

事件生成效率的对比如下表 5.17所示，从表中可知，Monkey 方法执行速度最快，WebExplor 次之，本文提出的方法 UniRLTest 较慢，而 Q-testing 方法最慢。

经分析，UniRLTest 的时间开销主要在图像处理的具体实现方面，因此，仍有可提高效率的空间。

表 5.17: 事件生成效率对比表（单位：个/min）

Monkey	Q-testing	WebExplor	UniRLTest
169	1.71	31.29	8.20

综合上述安卓应用遍历测试与 Web 应用遍历测试实验效果来看，本文提出的遍历测试方法 UniRLTest 在具备了平台通用性的同时，相比 Monkey、Q-testing 和 WebExplor 这些方法，还能达到较好的代码覆盖率效果。其有效性至此得以验证。

5.4 本章小结

本章介绍了对基于图像理解强化学习的通用遍历测试技术的功能性测试与实验，并通过这些测试与实验，验证了工具的可用性和方法的有效性。本章首先从工具的需求出发，设计了用于验证每一个需求是否得到满足的测试用例套件，接着使用这些测试用例套件进行功能性测试，以验证该工具的可用性；接着从实验待测应用集、对比工具、实验指标和实验方法四个方面介绍了实验设计，并给出最终的实验结果，以验证该方法的有效性。

第六章　总结与展望

6.1　总结

遍历测试是一种重要的界面测试方法，其核心在于对待测应用所代表的状态空间的充分探索。当前的遍历测试方法大多基于平台底层提供的页面布局结构信息作为页面状态，这种做法虽然一定程度上保证了状态的准确性，但受限于平台提供的信息，从而不同平台的状态信息结构不同，使探索策略无法实现平台通用。另一方面，针对 H5 页面、Canvas 组件这一类具体内容不在平台底层布局信息中出现的、或不易提取其具体内容的页面与组件，现在的探索方法所提取的状态自然是不准确的，丢失了许多重要信息，从而影响对相关状态空间的探索。本文提出了一种基于深度截屏理解强化学习的平台通用遍历测试方法 UniRLTest，通过对每一个页面截屏的深度理解实现状态信息提取，并应用强化学习实现对待测应用的遍历测试。该方法利用 Canny 边缘检测和光学字符识别技术实现控件识别；利用卷积神经网络和长短期记忆实现对截屏中的控件信息和布局信息的嵌入，由此整合得到页面状态信息；利用深度强化学习处理由此带来的状态空间过大的难题。由于该方法的状态提取基于截屏图像，所以不受平台的限制，能够识别难以通过平台提取的页面信息。由于以上优势，根据对该方法有效性的实验可知，该方法相比当前常见的遍历测试方法效果较优。

本文除了提出了上述平台通用的遍历测试方法以外，还应用该方法，基于 Tensorflow、Flask 和 WebSocket 等框架和技术实现了一个遍历测试工具，该工具分为前端、服务端和算法端，测试人员可以在前端界面中配置遍历测试的参数，开始一次遍历测试，实时监控待测设备的屏幕和测试日志，并在测试结束后下载测试日志以复盘本次遍历测试；服务端则负责操作测试设备，具体的遍历测试逻辑（即探索逻辑）由算法端实现。本文还通过对每个需求设计并执行相应的测试用例套件，验证了该工具的可用性。

6.2　展望

本文提出的基于深度截屏理解强化学习的平台通用遍历测试方法主要利用截屏信息识别当前状态，利用深度强化学习的方法实现遍历测试。相关工具实现方面则实现了安卓和 Web 两大常见平台的适配。考虑到更多平台的更多应用，本方法和工具还需不断优化，主要有以下若干方面的优化方向：

- 截屏理解技术的优化。当前的深度截屏理解技术能做到控件的识别和布局的提取，但在控件意图的提取方面仍然有所欠缺，若能将精确的控件意图融入状态信息中，能更完整地表达页面状态，从而提高探索效率和效果。

- 新的平台支持。当前支持的为安卓和 Web 平台，未来还可以考虑对 PC、iOS、各类小程序所在平台等的支持。对新平台的支持对底层探索逻辑没有影响，但是需要添加相关平台的交互接口的实现，包括捕获截屏和进行各类交互操作。

- 图像理解算法效率的提升。当前的实现中，时间开销主要在图像理解算法，包括图像比对、控件提取、控件比对和布局比对等，强化学习流程的时间开销相比并不高，未来可以考虑优化图像理解相关算法，以提升其运行效率，使该工具能在给定时间内生成并执行更多事件，从而优化其效果。

参考文献

[1] HAMLET R. Random Testing[C] // Encyclopedia of Software Engineering. [S.l.] : Wiley, 1994 : 970 – 978.

[2] PAN M, HUANG A, WANG G, et al. Reinforcement Learning Based Curiosity-Driven Testing of Android Applications[C/OL] // ISSTA 2020 : Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis. New York, NY, USA : Association for Computing Machinery, 2020 : 153–164.
https://doi.org/10.1145/3395363.3397354.

[3] PATHAK D, AGRAWAL P, EFROS A A, et al. Curiosity-driven Exploration by Self-supervised Prediction[J/OL]. CoRR, 2017, abs/1705.05363.
http://arxiv.org/abs/1705.05363.

[4] CHICCO D. Siamese Neural Networks: An Overview[M/OL] // CARTWRIGHT H. Artificial Neural Networks. New York, NY : Springer US, 2021 : 73 – 94.
https://doi.org/10.1007/978-1-0716-0826-5_3.

[5] ZHENG Y, LIU Y, XIE X, et al. Automatic Web Testing using Curiosity-Driven Reinforcement Learning[C/OL] // . [S.l.] : arXiv, 2021.
https://arxiv.org/abs/2103.06018.

[6] CANNY J. A Computational Approach to Edge Detection[J/OL]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 1986, PAMI-8(6) : 679 – 698.
http://dx.doi.org/10.1109/TPAMI.1986.4767851.

[7] GOODFELLOW I, BENGIO Y, COURVILLE A. Deep Learning[M]. [S.l.] : MIT Press, 2016.

[8] LOWE D. Object recognition from local scale-invariant features[C/OL] // Proceedings of the Seventh IEEE International Conference on Computer Vision : Vol 2. 1999 : 1150 – 1157 vol.2.
http://dx.doi.org/10.1109/ICCV.1999.790410.

[9] BAY H, TUYTELAARS T, VAN GOOL L. SURF: Speeded Up Robust Features[C] // LEONARDIS A, BISCHOF H, PINZ A. Computer Vision – ECCV 2006. Berlin, Heidelberg : Springer Berlin Heidelberg, 2006 : 404 – 417.

[10] LAZEBNIK S, SCHMID C, PONCE J. Beyond Bags of Features: Spatial Pyramid Matching for Recognizing Natural Scene Categories[C/OL] // 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06) : Vol 2. 2006 : 2169 – 2178. http://dx.doi.org/10.1109/CVPR.2006.68.

[11] CIRESAN D C, MEIER U, SCHMIDHUBER J. Multi-column Deep Neural Networks for Image Classification[J/OL]. CoRR, 2012, abs/1202.2745. http://arxiv.org/abs/1202.2745.

[12] MOUSA A. Canny edge-detection based vehicle plate recognition[J]. International Journal of signal processing, Image processing and pattern recognition, 2012, 5(3) : 1 – 8.

[13] ELARABY W S, MEDIAN A H, ASHOUR M A, et al. Fractional canny edge detection for biomedical applications[C] // 2016 28th International Conference on Microelectronics (ICM). 2016 : 265 – 268.

[14] ALI M, CLAUSI D. Using the Canny edge detector for feature extraction and enhancement of remote sensing images[C] // IGARSS 2001. Scanning the Present and Resolving the Future. Proceedings. IEEE 2001 International Geoscience and Remote Sensing Symposium (Cat. No. 01CH37217) : Vol 5. 2001 : 2298 – 2300.

[15] 岳晓峰, 焦圣喜, 韩立强, et al. 模式识别中的光字符识别技术及应用综述 [J]. 河北工业科技, 2006, 23(5) : 312 – 316.

[16] ISLAM N, ISLAM Z, NOOR N. A Survey on Optical Character Recognition System[J/OL]. CoRR, 2017, abs/1710.05703. http://arxiv.org/abs/1710.05703.

[17] SRIVASTAVA S, PRIYADARSHINI J, GOPAL S, et al. Optical Character Recognition on Bank Cheques Using 2D Convolution Neural Network[C] // MALIK H, SRIVASTAVA S, SOOD Y R, et al. Applications of Artificial Intelligence Techniques in Engineering. Singapore : Springer Singapore, 2019 : 589 – 596.

[18] ZAHRA S B, MOAEN S, MUNIR S, et al. Optical Handwritten with Character Recognition[C/OL] // 2021 International Conference on Innovative Computing (ICIC). 2021 : 1 – 8.
http://dx.doi.org/10.1109/ICIC53490.2021.9692927.

[19] HA H T. Recognition of Invoices from Scanned Documents[C] // RASLAN. 2017.

[20] O'SHEA K, NASH R. An Introduction to Convolutional Neural Networks[J/OL]. CoRR, 2015, abs/1511.08458.
http://arxiv.org/abs/1511.08458.

[21] 周飞燕, 金林鹏, 董军, et al. 卷积神经网络研究综述 [J]. 计算机学报, 2017, 40(6) : 1229 – 1251.

[22] ALBAWI S, MOHAMMED T A, AL-ZAWI S. Understanding of a convolutional neural network[C/OL] // 2017 International Conference on Engineering and Technology (ICET). 2017 : 1 – 6.
http://dx.doi.org/10.1109/ICEngTechnol.2017.8308186.

[23] KRIZHEVSKY A, SUTSKEVER I, HINTON G E. ImageNet Classification with Deep Convolutional Neural Networks[J/OL]. Commun. ACM, 2017, 60(6) : 84–90.
https://doi.org/10.1145/3065386.

[24] SIMONYAN K, ZISSERMAN A. Very Deep Convolutional Networks for Large-Scale Image Recognition[J/OL], 2014.
https://arxiv.org/abs/1409.1556.

[25] SZEGEDY C, LIU W, JIA Y, et al. Going Deeper with Convolutions[J/OL]. CoRR, 2014, abs/1409.4842.
http://arxiv.org/abs/1409.4842.

[26] SZEGEDY C, VANHOUCKE V, IOFFE S, et al. Rethinking the Inception Architecture for Computer Vision[J/OL]. CoRR, 2015, abs/1512.00567.
http://arxiv.org/abs/1512.00567.

[27] SZEGEDY C, IOFFE S, VANHOUCKE V. Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning[J/OL]. CoRR, 2016,

abs/1602.07261.
http://arxiv.org/abs/1602.07261.

[28] HE K, ZHANG X, REN S, et al. Deep Residual Learning for Image Recognition[J/OL]. CoRR, 2015, abs/1512.03385.
http://arxiv.org/abs/1512.03385.

[29] LI Z, LIU F, YANG W, et al. A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects[J/OL]. IEEE Transactions on Neural Networks and Learning Systems, 2021 : 1 – 21.
http://dx.doi.org/10.1109/TNNLS.2021.3084827.

[30] 卢宏涛, 张秦川. 深度卷积神经网络在计算机视觉中的应用研究综述 [J]. 数据采集与处理, 2016, 31(1) : 1 – 17.

[31] 杨丽, 吴雨茜, 王俊丽, et al. 循环神经网络研究综述 [J]. 计算机应用, 2018, 38(A02) : 1 – 6.

[32] HOCHREITER S, SCHMIDHUBER J. Long Short-Term Memory[J/OL]. Neural Computation, 1997, 9(8) : 1735 – 1780.
http://dx.doi.org/10.1162/neco.1997.9.8.1735.

[33] MAYER H, GOMEZ F, WIERSTRA D, et al. A System for Robotic Heart Surgery that Learns to Tie Knots Using Recurrent Neural Networks[C/OL] // 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems. 2006 : 543 – 548.
http://dx.doi.org/10.1109/IROS.2006.282190.

[34] GERS F, SCHMIDHUBER E. LSTM recurrent networks learn simple context-free and context-sensitive languages[J/OL]. IEEE Transactions on Neural Networks, 2001, 12(6) : 1333 – 1340.
http://dx.doi.org/10.1109/72.963769.

[35] LI X, WU X. Constructing Long Short-Term Memory based Deep Recurrent Neural Networks for Large Vocabulary Speech Recognition[J/OL]. CoRR, 2014, abs/1410.4281.
http://arxiv.org/abs/1410.4281.

[36] KAELBLING L P, LITTMAN M L, MOORE A W. Reinforcement Learning: A Survey[J/OL]. CoRR, 1996, cs.AI/9605103.
https://arxiv.org/abs/cs/9605103.

[37] WHITE III C C, WHITE D J. Markov decision processes[J]. European Journal of Operational Research, 1989, 39(1): 1 – 16.

[38] JAMES F. Monte Carlo theory and practice[J]. Reports on progress in Physics, 1980, 43(9): 1145.

[39] WATKINS C J, DAYAN P. Q-learning[J]. Machine learning, 1992, 8(3): 279 – 292.

[40] LI Y. Deep reinforcement learning: An overview[J]. arXiv preprint arXiv:1701.07274, 2017.

[41] 刘全, 翟建伟, 章宗长, et al. 深度强化学习综述 [J]. 计算机学报, 2018, 41(1): 1 – 27.

[42] MNIH V, KAVUKCUOGLU K, SILVER D, et al. Playing Atari with Deep Reinforcement Learning[J/OL]. CoRR, 2013, abs/1312.5602.
http://arxiv.org/abs/1312.5602.

[43] PAWLIK M, AUGSTEN N. Tree edit distance: Robust and memory-efficient[J/OL]. Information Systems, 2016, 56: 157 – 173.
https://www.sciencedirect.com/science/article/pii/S0306437915001611.

[44] CESA-BIANCHI N, GENTILE C, LUGOSI G, et al. Boltzmann Exploration Done Right[J/OL]. CoRR, 2017, abs/1705.10257.
http://arxiv.org/abs/1705.10257.

简历与科研成果

基本情况

恽叶霄，男，汉族

教育背景

2020 ~ 2022	南京大学软件学院	硕士
2016 ~ 2020	南京大学软件学院	本科

读研期间的成果（包括发表的论文及参与的专利）

1. Shengcheng Yu, Chunrong Fang*, Kai Mei, Yexiao Yun, Zhenfei Cao, Zhihao Cao, Zhenyu Chen. Detecting Crowdsourced Test Report Consistency for Mobile Apps with Deep Image Understanding and Text Analysis. Preprint.
2. Shengcheng Yu, Chunrong Fang*, Tongyu Li, Mingzhe Du, Xuan Li, Jing Zhang, Yexiao Yun, Xu Wang, Zhenyu Chen. Automated Mobile App Test Script Intent Generation via Image and Code Understanding. Preprint.
3. Shengcheng Yu, Chunrong Fang*, Yexiao Yun, Yang Feng. Layout and Image Recognition Driving Cross-platform Automated Mobile Testing. ICSE 2021.
4. Zhenfei Cao, Xu Wang, Shengcheng Yu, Yexiao Yun, Chunrong Fang*. STIFA: Crowdsourced Mobile Testing Report Selection Based on Text and Image Fusion Analysis. ASE 2020 Demo.
5. 一种基于深度图像理解的移动应用跨平台强化学习遍历测试技术(202111471771.5), 房春荣、刘昱磊、张子谦、虞圣呈、恽叶霄、陈振宇
6. 一种基于图像理解技术的移动应用众包测试缺陷报告自动生成的方法(2020104872052), 房春荣, 虞圣呈, 恽叶霄, 王旭, 曹振飞, 李彤宇.
7. 一种基于自然语言处理的众包测试报告相似度检测的方法（2020104872029）, 房春荣, 王旭, 曹振飞, 虞圣呈, 恽叶霄, 李彤宇.

致　　谢

白驹过隙，两年的研究生生活至此已进入尾声。这两年中，陈振宇老师在研究方向和论文题目与内容方面给了我重要指导，令我有信心顺利毕业。陈老师不论工作日还是休息日均在认认真真地工作，一手忙于公司与学院的诸多事项，一手还要管理整个实验室从博士到本科生大大小小的事情，在每个学期伊始便为每个学生制定学期、学年的研究方向与计划，案牍劳形。因此，我首先要感谢陈老师在这两年间给我的所有帮助。

在平日的研究中，房春荣老师和虞圣呈博士也给了我很多悉心的指导，除了毕设，还带着我做了许多其它项目，投了一些顶级会议的学术论文，帮助我对学术研究的整体过程和细节建立了一定程度的认知，锻炼了我的创新思维、利用最新技术解决问题的能力和代码能力；在毕设项目上，他们也帮助我把握毕设的总体进度，给予我需要的人力物力支持，协助我更好地完成本毕设题目。因此，我也要感谢房老师和虞博对我提供的悉心帮助。