基于大语言模型的Web文本输入组件测试方法

张亚东[1],崔展齐[1*],兰文尉[1],徐伟利[1],曹鹤玲[2]
(1. 北京信息科技大学计算机学院,北京 100192;2. 河南工业大学信息科学与工程学院,河南郑州 450001)

摘 要: 文本输入组件是Web应用实现交互功能的重要组成部分,广泛应用于搜索查询、内容创作等操作场景,其输入内容通常受到语法和复杂业务规则的约束。若文本输入组件未能正确处理恶意或非预期的文本输入,可能导致应用崩溃。现有的Web图形用户界面(Graphical User Interface,GUI)测试工具未能充分考虑文本输入组件的约束关系,无法生成具有针对性的文本输入来检测应用中文本输入组件的错误。此外,现有方法通常忽略了多个文本输入组件之间还可能存在复杂的约束关系,难以生成多样化的文本输入组合。为此,本文提出了一种基于大语言模型(Large Language Models,LLMs)的Web应用文本输入组件测试方法LTICT(LLM-based Text Input Component Testing)。首先,LTICT从被测应用的HTML文件中提取文本输入组件的信息,以供LLM推断文本输入组件的约束关系,并据此引导LLM合成程序;然后,LTICT执行该程序来批量生成文本输入,以对文本输入组件进行测试;最后,LTICT将收集所测试文本输入组件的上下文信息和所生成测试数据的执行结果,反馈给LLM以帮助其分析多个文本输入组件间的约束关系,从而生成更多样化的文本输入组合。在4个开源Web应用上进行的实验结果表明,相比于广泛使用的自动化测试工具WebExplor、DBInputs、QTypist,LTICT检测文本输入组件错误的数量分别提升了34.21%、37.84%和8.51%。在检测文本输入组件错误的平均用时方面,LTICT比WebExplor、DBInputs、QTypist分别减少了10.69%、11.87%和6.99%。

关键词: Web GUI测试;文本输入生成;Web应用;提示构建;大语言模型;自动化测试工具

基金项目: 江苏省前沿引领技术基础研究专项(No.BK20202001);北京信息科技大学"勤信人才"培育计划项目(No.QXTCP B202406)

中图分类号: TP311.5 **文献标识码:** A **文章编号:** 0372-2112(2026)01-0276-15

电子学报URL:http://www.ejournal.org.cn **DOI:**10.12263/DZXB.20250880

Testing Text Input Components of Web Applications Based on Large Language Models

ZHANG Yadong[1], CUI Zhanqi[1*], LAN Wenwei[1], XU Weili[1], CAO Heling[2]
(1. *College of Computer Science*, *Beijing Information Science and Technology University*, *Beijing* 100192, *China*;
2. *College of Information Science and Engineering*, *Henan University of Technology*, *Zhengzhou*, *Henan* 450001, *China*)

Abstract: Text input components are essential to Web applications and are widely used in scenarios such as search queries and content creation. Their inputs are typically constrained by syntactic rules and complex business logics. If text input components fail to correctly handle malicious or unexpected input texts, they may cause application crashes. Existing automated graphical user interface (GUI) testing tools for web applications often ignore these constraints. As a result, they cannot generate diverse inputs to effectively detect faults of text input components. Moreover, existing methods often overlook complex constraints among multiple text input components, which makes it difficult to generate diverse input combinations. To address this issue, this paper proposes an approach for testing text input components of web applications based on large language models (LLMs), named LLM-based text input component testing (LTICT). First, LTICT extracts information about text input components from the HTML files of the application under test. It then uses a LLM to infer the constraints of the text input components and to synthesize a text generation program with respect to these constraints. Next, LTICT executes the program to produce input texts in batches to test text input components. Finally, LTICT feeds component contexts and execution outcomes back to the LLM. These feedbacks help the LLM to analyze inter-component constraints and to generate more diverse combinations of inputs. To evaluate the effectiveness of LTICT, comparative experiments are conducted on four open-source web applications with three automated testing tools, which are WebExplor, DBInputs, and QTypist. The experimental results show that LTICT detects more text input component faults, with improvements of 34.21%, 37.84%, and 8.51% over WebExplor, DBInputs, and QTypist, respectively. In addition, LTICT reduces the average time required to detect text input component faults by 10.69%, 11.87%, and 6.99%, respectively.

收稿日期:2025-12-31;录用日期:2026-01-13;责任编委:李姗姗
*通信作者:崔展齐

Keywords: web GUI testing; text input generation; Web applications; prompt construction; large language model; automated testing tool

Foundation Item(s): The Leading-edge Technology Program of Jiangsu Natural Science Foundation (No.BK20202001); Beijing Information Science and Technology University "Qin-Xin Talent" Cultivation Project (No.QXTCP B202406)

0 引言

随着互联网技术的快速发展,Web应用程序被广泛应用于金融、医疗、教育等诸多领域。Web应用功能日益复杂,其可靠性和安全性存在的问题可能导致严重的后果。如2024年5月,Snowflake因身份验证缺陷,导致大量敏感数据泄露,造成严重的经济损失(https://www.theverge.com/2024/5/31/24168984/ticketmaster-santander-data-breach-snowfla-ke-cloud-storage)。因此,需要对Web应用程序进行严格的测试,以确保其面对恶意或非预期的输入时具备足够的鲁棒性和稳定性。

Web应用程序通过文本输入、按钮等组件与后端服务交互,以响应用户操作并动态更新界面。其中,文本输入组件用来实现应用程序的输入功能,常用于搜索查询、内容创作等操作场景。其输入内容通常受到复杂业务规则的约束(即约束关系)[1]。然而,恶意或非预期的文本输入可能会导致应用行为异常,甚至引发安全事故。例如,2021年10月,内容管理网站Craft(https://www.craft.do/)未能有效校验日期输入,导致无效日期被持久化存储,进而影响数据查询功能,使用户无法访问其创建的内容;2023年11月,Group-IB报告显示(https://www.group-ib.com/blog/resumelooters/),黑客组织ResumeLooters通过向65个网站进行恶意输入,非法窃取了超过200万用户的个人信息,导致大规模数据泄露。由此可见,确保Web应用程序正确处理文本输入内容至关重要。因此,需要对文本输入组件进行充分测试。

自动化测试方法可以有效减少人力及时间成本,其中,自动化图形用户界面(Graphical User Interface,GUI)测试技术侧重于为文本输入组件生成符合语法和业务规则的有效文本输入,并尝试探索更多界面[2-4]。例如,QExplore[5]使用自然语言处理技术生成文本输入以探索更多界面,但这种方法无法生成多样化的文本输入,尤其是违反文本输入组件语法和业务规则的异常文本输入,导致无法有效揭示潜在的错误。为此,Liu等人[1]使用大语言模型(Large Language Model,LLM)生成异常文本输入,以检测Android应用程序中文本输入组件引发的错误,有效提升了检测文本输入组件错误的性能。但Android应用程序与Web应用程序在编程语言、业务逻辑实现等方面存在显著差异,使得该项工作难以直接应用于Web应用程序的文本输入组件测试。此外,多个文本输入组件之间还可能存在复杂的约束关系,为充分测试文本输入组件带来了更大挑战。

为解决这些挑战,本文提出了一种基于大语言模型的Web应用文本输入组件测试方法LTICT(LLM-based Text Input Component Testing)。首先,从被测应用的HTML文件中提取文本输入组件的信息及其约束关系来构建提示,以供LLM为其生成有效文本输入,并进一步分析文本输入组件应满足的约束关系(即推断约束关系)[1]。然后,LTICT根据有效文本输入和推断约束关系构建LLM提示,引导其合成程序来批量生成异常文本输入,以对文本输入组件进行测试。在此过程中,LTICT将收集文本输入组件的上下文信息和测试结果反馈给LLM,以帮助其分析同一表单中多个文本输入组件间的约束关系,从而生成更多样化的文本输入组合。为评估LTICT的性能,本文使用LTICT与WebExplor[6]、DBInputs[7]、QTypist[3]在Akaunting、Mantis、Tricentis和Dolibarr4个Web应用上进行了对比实验。实验结果表明,相比于WebExplor、DBInputs和QTypist,LTICT检测文本输入组件错误的数量分别提升了34.21%、37.84%和8.51%。在检测文本输入组件错误平均用时方面,LTICT比WebExplor、DBInputs、QTypist分别减少了10.69%、11.87%、6.99%。

本文研究的主要贡献如下:

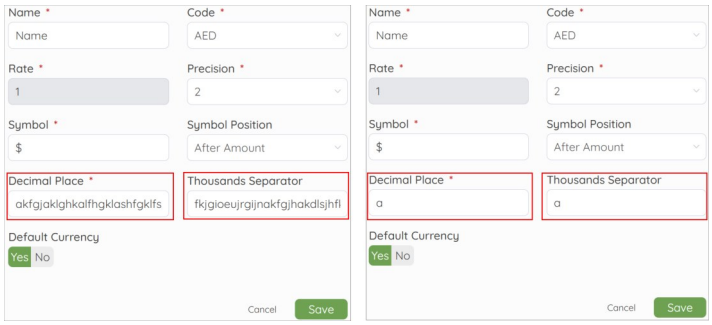
(1)提出了一种基于大语言模型的Web应用文本输入组件测试方法LTICT。该方法分析了文本输入组件的约束关系和测试反馈,并指导LLM生成多样化的文本输入数据。

(2)基于LTICT实现了原型工具,在一组真实的Web应用上进行了实验,并与测试工具WebExplor、DBInputs和QTypist进行对比,以评估LTICT检测Web应用文本输入组件错误的有效性和效率,并讨论了LTICT使用不同LLM的泛用性。

1 动机示例

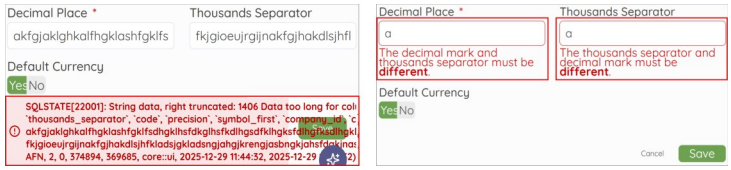
图1展示了Web应用Akaunting(https://github.com/akaunting)中因异常文本输入而导致的文本输入组件错误。Akaunting中新增货币的功能包含4个文本输入组件,分别为Name、Symbol、Decimal Place和Thousands Separator。图2为4个文本输入组件对应的HTML代码,其中红色部分为其需满足的约束关系,

即不能输入空值。此外，分析可发现，4个文本输入组件还应满足隐含的约束关系，即不能输入过长字符，且Decimal Place和Thousands Separator间存在不能输入相同字符的约束关系。在Akaunting中使用新增货币的功能时，用户需要在4个文本输入组件中分别编辑内容，然后点击保存。如图1(a)所示，如果在Decimal Place和Thousands Separator中输入异常值(如长度为200的随机字符串)，或输入相同的字符(如字符a)，会导致应用异常，弹出警告提示(图1(b))，并返回错误响应(图1(c))。



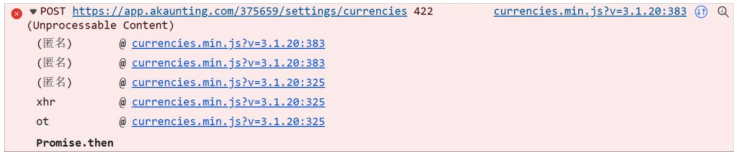
(a) 在Decimal Place和Thousands Separator中输入200个随机字符或字符a并保存

(a) Enter 200 random characters or the character a into the Decimal Place and Thousands Separator fields and save



(b) 应用行为异常

(b) The application exhibits abnormal behavior



(c) 返回错误响应

(c) Return an error response

图1 Akaunting应用文本输入组件错误

Figure 1 Text input component faults in the Akaunting application

以图1所示的Akaunting应用中的文本输入组件Decimal Place和Thousands Separator为例，使用自动化测试工具WebExplor[6]和DBInputs[7]执行30次测试，均不能检测出应用中的文本输入组件错误。分析发现，上述工具在为文本输入组件生成输入时，没有充分考虑文本输入组件间的约束关系，难以生成违反Decimal Place和Thousands Separator间约束关系的异常文本输入。其中，WebExplor通过生成随机字符串对Web应用中的文本输入组件进行测试，这种方式未考虑不同文本输入组件间存在的约束关系，无法产生违反特定组件间约束关系的输入组合，导致WebExplor难以检测出应用中此类约束关系导致的错误。DBInputs可利用文本输入组件HTML标签的语法和语义信息分析其约束关系，并从应用的数据库中检索输入数据。尽管从应用数据库中可能检索到触发文本输入组件错误的数据，但因应用数据库中的数据量有限，且未能分析多个文本输入组件间的约束关系，仍然难以检测出因违反文本输入组件间约束关系而导致的错误。

要成功触发该文本输入组件的错误必须正确分析其约束关系，并生成违反其约束关系的异常文本输入。然而，文本输入组件的HTML代码通常并不显式包含约束关系的描述。LLM具有强大的理解能力和逻辑推理能力[8]，如能利用其分析文本输入组件信息

行号	HTML代码
1	<input type="text" name="name" id="name" value="" placeholder="Enter
2	Name" required="required">
3	<input type="text" name="symbol" id="symbol" value="" placeholder="
4	Enter Symbol" required="required">
5	<input type="text" name="decimal_mark" id="decimal_mark" value=""
6	placeholder="Enter Decimal Place" required="required">
7	<input type="text" name="thousands_separator" id="thousands_separator"
8	value="" placeholder="Enter Thousands Separator" required="required">

图2　文本输入组件Name、Symbol、Decimal Place和Thousands Separator的HTML代码

Figure 2　HTML code of the text input components Name, Symbol, Decimal Place, and Thousands Separator

的同时自动推理其隐含的约束关系,将能生成触发其错误的异常文本输入。

2　LTICT方法

LTICT的框架如图3所示。首先,从被测应用的HTML文件中提取文本输入组件的信息及其约束关系来构建提示,以供LLM为文本输入组件生成有效文本输入,并进一步分析其应满足的推断约束关系。然后,根据有效文本输入和推断约束关系构建LLM提示,引导其合成程序来批量生成异常文本输入,以对文本输入组件进行测试。此外,LTICT将收集所测试文本输入组件的上下文信息和所生成测试数据的执行结果,反馈给LLM以生成更多样化的异常文本输入。

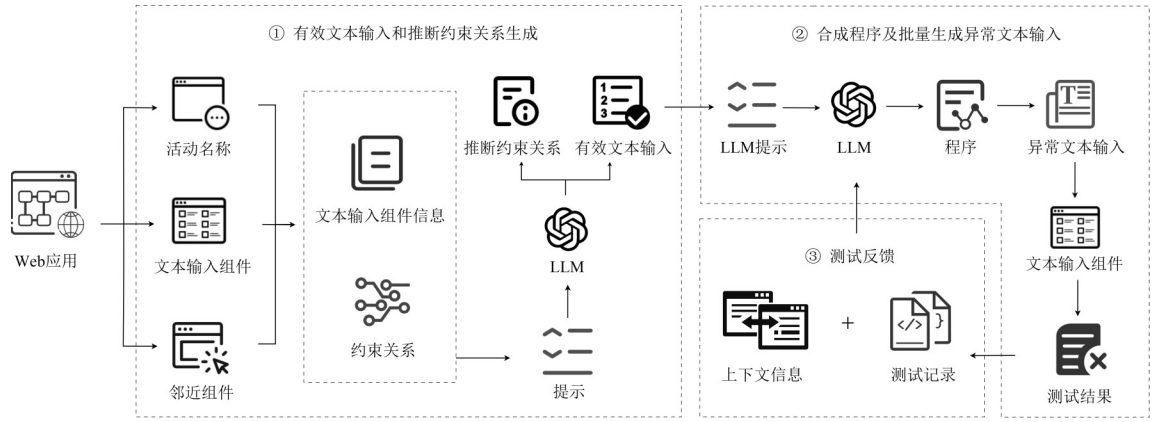


图3　LTICT方法框架

Figure 3　Framework of the LTICT

2.1　有效文本输入和推断约束关系生成

通过生成有效文本输入能够对文本输入组件的基本功能进行测试,并且可为后续生成异常文本输入提供基础数据。此外,直接从应用中提取文本输入组件的约束关系通常不够全面。为解决这些问题,LTICT从被测应用HTML文件中获取文本输入组件的信息及其约束关系,使用LLM生成有效文本输入,并进一步分析文本输入组件应满足的推断约束关系。

为获取文本输入组件的信息,LTICT使用lxml库(https://lxml.de/)解析被测应用的HTML文件,从中提取与文本输入组件相关的HTML标签。首先,LTICT获取应用名称和文本输入组件所在界面的活动名称,以帮助LLM理解文本输入组件的功能。然后,LTICT提取文本输入组件HTML标签中的“name”“value”“text”和“placeholder”属性,并将其属性值作为文本输入组件的描述。此外,LTICT还会获取与文本输入组件同一活动的邻近组件,并提取其HTML标签中相同属性值作为邻近组件的描述,以帮助LLM理解文本输入组件的上下文语境。

约束关系反映了文本输入组件输入内容所受的复杂业务规则限制,我们参考Liu等人[1]对文本输入组件约束关系的分类方法,将约束关系分为组件内部约束(intra-component constraint)和组件间约束(inter-component constraint)。其中,组件内部约束描述了单个文本输入组件对输入内容的要求,又可分为显示约束(explicit constraint)和隐式约束(implicit constraint)。显示约束主要表现在应用界面上呈现的输入规则,例如,应用界面中的号码输入框显示“请输入数字”。隐式约束则主要表现在文本输入时的反馈提示,例如,在设置只包含数字的密码时,应用程序返回提示“请输入至少包含一个英文字符”。组件间约束描述了不同文本输入组件之间对输入内容的要求,例如,应用中要求出发地和目的地的输入地址不能相同。根据以上约束关系的分类,LTICT通过提取文本输入组件HTML标签的属性值及其文本字段,构建文本输入组件的约束关系。

LTICT使用获取到的文本输入组件的信息及其约束关系来构建提示,用于指导LLM生成有效文本输入和推断约束关系,提示模板如表1所示。表1中第1行的系统提示用于定义LLM角色。表1中第2~4行

表1 生成有效文本输入和推断约束关系提示模板

Table 1 Prompt template for generating valid text input and inferred constraint relations

序号	对话角色	类型	提示模板
1	系统提示	系统指令	You are an expert in web text input component testing.
2	用户提示	文本输入组件的信息	We want to test the text input components on <ActivityName> page of <WebName> website which has <NumOfTextInputComponent>. The first text input component is <TextInputComponent>, its description is <TextInputComponent>. The second text input component is … Nearby components include <NearbyComponent>.
3	用户提示	约束关系	There are explicit constraints: <explicit constraint>; Implicit constraints: <implicit constraint>; Inter-component constraints: <inter-component constraint>.
4	用户提示	问题描述	Please generate a valid text input based on the above information and further infer the constraint relations of each text input component. Outputs in the following format: "Valid input: …; Inferred constraint relations: …".

为用户提示,用于为LLM提供具体任务描述。其中,第2行是文本输入组件的信息,包括应用名称、活动名称、文本输入组件描述及其邻近组件描述,以帮助LLM理解文本输入组件在应用程序中的功能。第3行是文本输入组件的约束关系,包括显式约束、隐式约束和组件间约束,以确保LLM生成的有效文本输入充分考虑这些约束关系,并进一步分析文本输入组件应满足的推断约束关系。第4行为问题描述,引导LLM根据提供的信息生成有效文本输入和推断约束关系。

LTICT将提示输入LLM,使其生成有效文本输入和推断约束关系。随后,LTICT通过关键词匹配的方法,提取LLM输出中"Valid input"对应的内容作为有效文本输入,并将"Inferred constraint relations"对应内容作为推断约束关系。以动机示例中的4个文本输入组件为例,图4为LTICT为其生成有效文本输入和推断约束关系过程的示例。其中,图4(a)为LTICT获取文本输入组件的信息和约束关系,包括应用名称、活动名称、文本输入组件描述、邻近组件描述以及约束关系。其中,约束关系包括文本输入组件不能输入空值等显式约束;图4(b)为LTICT根据文本输入组件的信息和约束关系构建的提示;图4(c)为LLM根据提示生成的有效文本输入和推断约束关系。其中,推断约束关系包括文本输入组件不能输入过长字符串等隐式约束,以及文本输入组件Decimal Place和Thousands Separator不能输入相同值等组件间约束。

2.2 合成程序及批量生成异常文本输入

LTICT根据有效文本输入和推断约束关系来生成异常文本输入,以对文本输入组件进行测试。现有研究表明[1],直接通过LLM生成测试数据的效率较低,且频繁与LLM交互需要消耗大量资源。为解决这些问题,LTICT使用LLM合成程序来批量生成异常文本输入。

在LLM生成有效文本输入和推断约束关系后,LTICT根据二者构建用于指导LLM合成程序的提示。提示模板如表2所示,表2中第1行为系统提示,用于定义LLM的角色。表2中第2~4行为用户提示,包括有效文本输入、推断约束关系和问题描述。其中,有效文本输入作为程序生成异常文本输入的基础数据,推断约束关系则用于指导程序来生成异常文本输入,最后添加问题描述,引导LLM根据提供的信息合成可执行程序,并明确该程序的功能是依据有效文本输入和推断约束关系生成异常文本输入。

LTICT将提示输入至LLM,引导其合成程序,并执行该程序来批量生成异常文本输入,以对文本输入组件进行测试。图5为LTICT测试文本输入组件的示例。其中,图5(a)为LTICT构建LLM合成程序的提示,包括有效文本输入、推断约束关系和问题描述三部分,图5(b)为LLM根据提示合成的程序,图5(c)为LTICT执行该程序生成异常文本输入,并对文本输入组件进行测试。

2.3 测试反馈

在2.1节中,LTICT从被测应用的HTML文件中提取了文本输入组件的约束关系。然而,组件间约束通常难以直接从HTML文件中获取。为帮助LLM推断组件间约束,LTICT将收集被测应用中文本输入组件的上下文信息,并将其反馈给LLM。此外,文本输入组件的测试结果能够指导LLM生成更多样化的异常文本输入。为此,LTICT将收集文本输入组件的上下文信息和测试结果,反馈给LLM以对文本输入组件进行迭代测试。

在对文本输入组件C_i的测试过程中,我们将对其测试过程中的反馈表示为四元组$F_i=\left(\mathrm{des}_i,\mathrm{ap}_i,\mathrm{cons}_i,\mathrm{TR}_i\right)$。其中,$\mathrm{des}_i$表示$C_i$对应的描述信息,$\mathrm{ap}_i$表示$C_i$所在界面的活动名称,$\mathrm{cons}_i$表示$C_i$应满足的推断约束关系。$\mathrm{TR}_i$表示$C_i$的测试记录集合$\left\{T_i^1,T_i^2,\cdots,T_i^k,\cdots\right\}$。其中,测试记录$T_i^k$为二元组$\left(u_i^k,r_i^k\right)$,$u_i^k$为测试$C_i$的第$k$个异常文本输入,$r_i^k$为其执行结果。

应用名称	WebName = "Akaunting"
活动名称	ActivityName = "New Currency"
文本输入组件描述	TextInputComponent (1)"name, Enter Name" (2)"symbol, Enter Symbol" (3)"decimal_mark, Enter Decimal Place" (4)"thousands_separator, Enter Thousands Separator"
邻近组件描述	NearbyComponent = "Select Code, Select Precision, Select Symbol Position, default_currency"
约束关系	Explicit constraint: (1) type = "text", required='required' (2) type = "text", required='required' (3) type = "text", required='required' (4) type = "text", required='required' Implicit constraint: N/A; Inter-component constraint: N/A.

(a) 获取文本输入组件信息和约束关系

(a) Retrieve the information and constraint relations of the text input components

文本输入组件的信息	We want to test the text input components on "New Currency" of "Akaunting" website which has 4 text input components. The first text input component is "name", its description is "Enter Name". Nearby components include "Select Code". ···
约束关系	For the first text input component, there are explicit constraints: (1) Type = "text"(2) required='required'; Implicit constraint: N/A; Inter-component constraint: N/A. For the second text input component, ···
问题描述	Please generate a valid text input based on the above information and further infer the constraint relations of each text input component. Outputs in the following format: "Valid input: ···; Inferred constraint relations: ····".

(b) 生成有效文本输入和推断约束关系的提示

(b) Construct prompts for generating valid text input and inferred constraint relations

Valid input:
1.name:'Test Currency' 2.symbol:'$' 3.decimal_mark:'.' 4.thousands_separator:','
Inferred constraint relations:
1.name: - Explicit constraint: (1) Plain text. (2) Input cannot be empty.
- Implicit constraints: (1) Input cannot be too long. (2) Input cannot be repeated.
- Inter-component constraint: N/A.
2.symbol: - Explicit constraint: (1) Plain text. (2) Input cannot be empty.
- Implicit constraints: (1) Should be a valid currency symbol string (commonly 1–5 visible characters). - Inter-component constraint: N/A.
3.decimal_mark: - Explicit constraint: (1) Plain text. (2) Input cannot be empty.
- Implicit constraints: (1) Should be a single character.
- Inter-component constraint: (1) Should be different from the value of 'thousands_separator' to avoid ambiguity in numeric formatting.
4.thousands_separator: - Explicit constraint: (1) Plain text.
(2) Input cannot be empty. - Implicit constraints:(1) Should be a single character.
- Inter-component constraint: (1) Should be different from the value of 'decimal_mark' to ensure unambiguous numeric representation.

(c) LLM输出

(c) LLM output

图4　LTICT生成有效文本输入和推断约束关系示例

Figure 4　Example of LTICT generating valid text input and inferred constraint relations

其中,des_i、ap_i和$cons_i$共同构成C_i的上下文信息。LTICT从文本输入组件的HTML文件中获取C_i对应的描述信息和所在界面的活动名称,用于帮助LLM理解文本输入组件之间的交互语境,并使用LLM生成推断约束关系,用于帮助LLM理解C_i应满足的各类约束。在测试过程中,LTICT将获取ap_i中所有

有效文本输入	We want to test the 4 text input components "name, symbol decimal_mark, thousands_separator" on "New Currency" page of "Akaunting" website. Valid input for the text input component is:"1.name: 'Test Currency' 2.symbol: '$' 3.decimal_mark: '.' 4.thousands_separator: ',' ".
推断约束关系	The inferred constraint relations are as follows: "1.name: - Explicit constraint...".
问题描述	Please generate an executable program, and the role of the program is to generate unusual input data based on the valid input and inferred constraint relations.

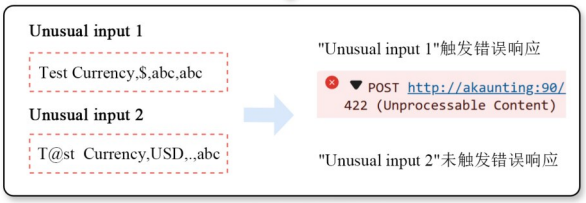
(a) LLM合成程序的提示

(a) Prompt for LLM to synthesize an executable program

```
VALID = ["Test Currency", "$", ".", ","]
SPECIAL = "!@#$%^&*()_+-=[]{}|;:'\",.<>/?\\"
VISIBLE = string.ascii_letters + string.digits + SPECIAL
# Plain text; cannot be empty; not too long; (optionally) cannot be repeated
def gen_text_cases(s: str, *, forbid_repeat: bool):
    cases = ["", " " * 8, "".join(random.choices(SPECIAL, k=10))]
    for L in [200, 300]:
        cases.append(s + "".join(random.choices(VISIBLE, k=L)))
    cases.append(s[:1] + random.choice(SPECIAL) + s[2:] if s else "")
    cases.append(s * 10 if forbid_repeat else s)
    return cases
# Plain text; cannot be empty; typically 1–5 visible chars
def gen_symbol_cases(s: str):
    cases = ["", " " + s + " ", "USD"]
    cases.append(s * 6)  # too long
    cases.append("".join(random.choices(SPECIAL, k=8)))
    return cases
# Plain text; cannot be empty; string inputs
def gen_sep_cases(s: str):
    cases = [s, "",  "abc"]
    return cases
                    ...
```

(b) LLM合成程序

(b) The LLM synthesizes an executable program



(c) 文本输入组件测试

(c) Test the text input components

图5　测试文本输入组件示例

Figure 5　Example of testing text input components

文本输入组件的上下文信息,按其在界面中的位置顺序输入LLM,以帮助LLM从文本输入组件之间的语义和结构层面分析组件间约束。此外,LTICT将异常文本输入数据填充至文本输入组件中,并从浏览器控制台输出日志中获取其触发的错误类型,包括不同的错误状态码和JavaScript异常[6],以判断异常文本输入的执行结果。随后,LTICT根据异常文本输入的执行结果来构建测试记录集合,以帮助LLM分析不同输入数据触发组件错误的效果,从而生成更多违反文本输入组件约束关系的异常文本输入。

算法1描述了构建测试反馈的算法,输入包括文本输入组件C_i所在HTML文件f和异常文本输入集

表 2 LLM 合成程序提示模板

Table 2 Prompt template for LLM to synthesize an executable program

序号	对话角色	类型	提示模板
1	系统提示	系统指令	You are an expert in web text input component testing.
2	用户提示	有效文本输入	We want to test <TextInputComponent> on <ActivityName> page of <WebName> website. Valid input for the text input component is: <ValidInput>.
3	用户提示	推断约束关系	The inferred constraint relations are as follows: <InferredConstraintRelations>.
4	用户提示	问题描述	Please generate an executable program, and the role of the program is to generate unusual input data based on the valid input and inferred constraint relations.

U_i,输出为测试反馈 F_i。首先,初始化测试反馈 F_i 和测试记录集合 TR_i。然后,通过解析 HTML 文件 f,提取文本输入组件 C_i 对应的描述信息 des_i 及其所在界面的活动名称 ap_i(第 2~3 行),并获取 LLM 生成的推断约束关系 cons_i(第 4 行)。之后,遍历异常文本输入集 U_i,获取 U_i 中异常文本输入 u_i 及其执行结果 r_i(第 5~6 行),根据异常文本输入 u_i 及其执行结果 r_i 构建测试记录二元组 T_i^u,并将 T_i^u 加入测试记录集合 TR_i(第 7~8 行)。最后,构建测试反馈 F_i,并返回 F_i(第 10~11 行)。

算法 1 构建测试反馈算法

输入: 文本输入组件 C_i 所在的 HTML 文件 f 和异常文本输入集 U_i

输出: 测试反馈 F_i

1.Initialize F_i,TR_i //初始化 F_i 和 TR_i

2.des_i = GetDescription(f) //获取 C_i 的描述信息

3.ap_i = GetActivityPage(f) //获取 C_i 所在界面的活动名称

4.cons_i = LLMInferConstraint(f) //获取推断约束关系

5.FOR each u_i in U_i do

6. r_i = GetResult(u_i) //获取异常文本输入的执行结果

7. T_i^u = ConstructBinaryGroup(u_i,r_i) //构建测试记录 T_i^u

8. TR_i.add(T_i^u) //将 T_i^u 加入 TR_i

9.END FOR

10.F_i = ConstructTestFeedback(des_i,ap_i,cons_i,TR_i) //构建 F_i

11.RETURN F_i //返回测试反馈 F_i

在构建测试反馈后,LTICT 将其加入到 LLM 合成程序的提示中,使 LLM 持续优化有效文本输入和推断约束关系,从而生成更具针对性的异常文本输入,以对文本输入组件进行迭代测试。图 6 为 LTICT 使用测试反馈的示例。其中,图 6(a)为 LTICT 构建的测试反馈,包括上下文信息和测试结果,其将被加入到 LLM 合成程序的提示中;图 6(b)为 LLM 根据提示迭代优化有效文本输入和推断约束关系,并合成程序;图 6(c)为 LTICT 执行该程序生成异常文本输入,对文本输入组件进行迭代测试,并获取测试结果来更新测试反馈。

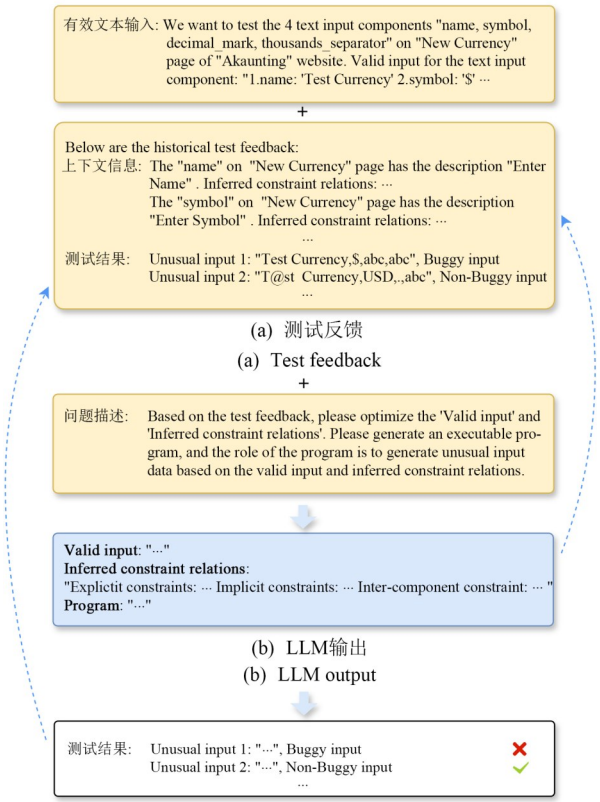


图 6 使用测试反馈示例

Figure 6 Example of using test feedback

3 实验设计

为评估 LTICT 的有效性,我们提出以下 3 个研究问题。

RQ1:与其他 Web 应用自动化测试工具相比,LTICT 检测文本输入组件错误的有效性和效率如何?

LTICT 利用 LLM 生成多样化的异常文本输入,以提高检测文本输入组件错误的有效性和效率。为分析 LTICT 相比其他方法在检测文本输入组件错误效果上的差异,我们将 LTICT 与 Web 应用自动化测试工具 WebExplor、DBInputs 和 QTypist 进行对比。

RQ2:LTICT 的各模块对检测文本输入组件错误

性能的影响如何?

LTICT主要包括有效文本输入生成、推断约束关系生成、批量异常文本输入生成和测试反馈构建等模块,为研究这些模块是否有助于提升LTICT检测文本输入组件错误的效果,我们进行了消融实验。

RQ3:使用LTICT生成的有效文本输入作为输入数据能否提升基线工具探索Web应用界面的性能?

LTICT可生成满足文本输入组件约束关系的有效文本输入。为分析LTICT生成有效文本输入的通用性,以及是否对探索Web应用界面有提升作用,我们将LTICT生成的有效文本输入作为基线工具WebExplor和DBInputs的输入进行测试,以分析其对基线工具的影响。

3.1　实验对象

Clerissi等人[7]的研究中,采用了来自不同领域并且依赖文本输入交互的4个Web应用,其中Budgets应用因保密原因未对外开放。实验采用了其余3个Web应用,分别属于软件开发、保险服务和企业管理领域。此外,实验还加入了与Budgets应用功能相似,且被广泛应用于Web应用测试研究工作[9-10]的开源会计管理应用Akaunting替代Budgets。表3为实验对象的基本信息。其中,括号内数字代表同一表单中包含组件间约束的文本输入组件数量。4个实验对象中,最多含有63个文本输入组件,最少含有12个。可以看出,文本输入组件在Web应用中被广泛使用,这也表明有必要对文本输入组件间约束进行针对性测试。

表3　实验对象基本信息

Table 3　Basic information of the experimental subjects

实验对象	描述	版本	文本输入组件数量	链接
Akaunting	Accounting Management System	3.1.20	46(4)	https://github.com/akaunting
Mantis	Issue Tracking System	2.21.0	48(10)	https://sourceforge.net/projects/mantisbt
Tricentis	Vehicle Insurance Demo	1.01	12(4)	http://sampleapp.tricentis.com
Dolibarr	ERP and CRM System	10.0.0	63(13)	https://sourceforge.net/projects/dolibarr

3.2　基线方法

实验将LTICT和开源工具WebExplor[6]、DBInputs[7]、QTypist[3]进行对比。

其中,WebExplor是一种基于强化学习的Web应用测试方法。该方法将Web应用的URL和HTML标签抽象为应用状态,并利用好奇心驱动的强化学习引导界面探索。在处理文本输入组件时,WebExplor生成随机字符串作为文本输入,用于检测应用中文本输入组件的错误。实验表明,相比于DIG[11]、SUBWEB[12]等自动化测试工具,WebExplor能够探索到更多应用界面,并检测出更多文本输入组件错误。Zheng等人[6]的研究工作未公开WebExplor的实现细节,Sherin等人[5]复现了WebExplor并开源了代码(https://github.com/salmansherin/QExplore/tree/main/Experimental%20results/webExplore),且实验结果与Zheng等人的工作一致。因此,实验采用该开源版本的WebExplor作为基线方法。

DBInputs从被测Web应用数据库中检索文本输入,以生成能够满足文本输入组件约束关系的文本输入。具体来说,DBInputs首先从被测应用的业务数据库中获取数据,以构建测试数据库。随后,DBInputs通过计算HTML标签与测试数据库列名之间语法和语义相似性得分,利用聚类算法筛选相似性得分最高的列,并将测试数据库中该列的数据作为文本输入。实验表明,DBInputs能检测出Link[13]、Monkey[14]等自动化测试工具无法检测到的文本输入组件错误。此外,Clerissi等人根据测试数据库与被测应用的业务数据库中数据重叠程度的不同,将其划分为完全重叠、部分重叠和互不相交三种类型。本文使用其推荐的互不相交的类型作为DBInputs的测试数据库。

QTypist是一种基于大语言模型的文本输入组件测试方法。QTypist首先从Android应用的视图层文件中提取文本输入组件的信息,并通过关键词匹配将其划分为不同的语义类别。随后,根据语义类别并结合预定义的语言模式来构建提示词,使用LLM生成文本输入数据,以对文本输入组件进行测试。此外,QTypist通过构建问答数据集对LLM进行提示微调,从而生成更加多样化的文本输入。实验结果表明,相比于DroidBot[15]、TextExerciser[16]等自动化测试工具,QTypist能够生成更加多样化的文本输入数据,并检测出更多文本输入组件的错误。由于Liu等人[3]未公开其用于提示微调的问答数据集,Alian等人[17]使用OpenAI API调用LLM来生成文本输入数据,并用于检测Web文本输入组件的错误。因此,实验采用Alian等人开源的QTypist(https://anonymous.4open.science/r/webform-testing-56BB/qtypist.py)作为基线方法,并使用GPT-3.5-turbo模型以生成文本输入数据。

3.3　实验设置

在RQ1和RQ2的实验中,我们沿用了Liu等人[1]工作中的两种实验设置,即对每个文本输入组件分别

执行30次文本输入测试,以及持续30 min的文本输入测试,并记录返回错误信息的文本输入组件数量、错误检测率、错误类型数量,计算触发错误的平均尝试次数和平均用时。其中,错误检测率是成功触发文本输入组件错误的输入数据与实验中所有输入数据的比率,被广泛应用于检测Web应用错误的工作[2,5]。在RQ3的实验中,我们沿用了Wang等人[2]和Zheng等人[6]的实验设置,使用URL来统计Web应用界面的数量。在实验中,对4个实验对象分别运行10次测试,每次运行30 min,并记录探索到的Web应用界面平均数量。

在LTICT中,使用Selenium(https://www.selenium.dev/)获取文本输入组件的HTML结构文件并构建自动化测试脚本。对于LLM,我们使用OpenAI API调用GPT-3.5-turbo模型,它被广泛应用于Web应用测试领域[18-19]。实验在一台配备有32 GB内存和Intel(R) Core(TM) Ultra 7处理器的计算机上完成,运行的操作系统和开发环境分别为Windows 11专业版和Python 3.10。

4 实验结果分析

4.1 针对RQ1的实验结果分析

为评价LTICT检测文本输入组件错误的有效性和效率,我们将LTICT和WebExplor、DBInputs、QTypist就检测文本输入组件错误的情况进行对比。

在执行30 min测试时,我们统计了LTICT和基线工具在4个实验对象中检测出的文本输入组件错误数量,结果如表4所示。其中,括号内数字表示检测出的因违反同一表单中组件间约束而导致文本输入组件错误数量,"—"表示未能检测到文本输入组件错误。从表4中可以看出,对于检测出的因违反同一表单中组件间约束而导致文本输入组件错误数量,LTICT比WebExplor和DBInputs分别多检测出4个和6个文本输入组件错误,和QTypist检测出的数量相同。LTICT检测出的所有文本输入组件错误数量,比WebExplor、DBInputs和QTypist分别多检测出13个、14个和4个文本输入组件错误。其中,对于实验对象Dolibarr,当文本输入组件接收到异常文本输入时,其可通过异常处理机制捕获并处理相关错误,而不会将错误信息输出至控制台日志或界面窗口。因此,所有测试工具均未检测到文本输入组件错误。

表5为LTICT和基线工具检测因违反组件间约束而导致文本输入组件错误的用时情况。其中,"—"表示未能检测到因违反组件间约束而导致的文本输入组件错误。从表5中可以看出,LTICT可以检测到WebExplor、DBInputs和QTypist检测出的因违反组件间约束而导致的文本输入组件错误,且用时更少。

表4 不同测试工具检测文本输入组件错误数量

Table 4 Number of text input component faults detected by different testing tools

实验对象	LTICT	WebExplor	DBInputs	QTypist
Akaunting	28(2)	19(—)	21(—)	26(2)
Mantis	20(10)	16(8)	13(6)	18(10)
Tricentis	3(—)	3(—)	3(—)	3(—)
Dolibarr	—	—	—	—
总数	51(12)	38(8)	37(6)	47(12)

表5 不同测试工具检测因违反组件间约束而导致文本输入组件错误的用时情况 单位:min

Table 5 Time required by different testing tools to detect text input component faults caused by violations of inter component constraints unit: min

实验对象	LTICT	WebExplor	DBInputs	QTypist
Akaunting	16.88	—	—	18.25
Mantis	5.61	7.84	13.50	7.69
Tricentis	—	—	—	—
Dolibarr	—	—	—	—
平均用时	20.62	24.46	25.88	21.49

LTICT比WebExplor、DBInputs和QTypist的平均用时分别减少了15.70%、20.32%和4.05%。这是因为,LTICT根据文本输入组件的上下文信息考虑了组件间约束,并使用LLM合成程序来批量生成违反组件间约束的输入组合,使其检测因违反组件间约束而导致文本输入组件错误的效率较高。

表6~表8为LTICT和基线工具检测不同实验对象中的文本输入组件错误情况。其中,表6为实验对象Akaunting的检测情况,在执行30次输入测试时,LTICT生成的异常文本输入错误检测率为56.16%,相比于WebExplor、DBInputs、QTypist分别提升47.63%、35.98%、23.21%。此外,LTICT可以检测到6种错误类型,WebExplor和QTypist分别能够检测到3种和4种错误类型,DBInputs只能检测到2种错误类型。WebExplor、DBInputs和QTypist检测到的文本输入组件错误类型,均能被LTICT检测到,且LTICT用时更少。LTICT比WebExplor检测文本输入组件错误的平均用时减少了26.51%,比DBInputs和QTypist的平均用时分别减少了21.74%和18.26%。表7和表8分别为实验对象Mantis和Tricentis的检测情况。可以看到,LTICT检测文本输入组件的错误检测率高于WebExplor、DBInputs和QTypist,且LTICT能够检测出更多的错误类型,比WebExplor、DBInputs和QTypist分别多检测到3种、2种和1种错误类型。对于实验对象Tricentis,在执行30 min测试时,LTICT比WebExplor检测文本输入组件错误的平均用时增加了2.19%。对Tri-

centis的错误进行分析,发现仅通过随机字符串输入即可触发多种文本输入组件错误,使得WebExplor检测错误的平均用时较少。而LTICT通过分析测试反馈,同样可以在较短时间内生成触发文本输入组件错误的异常文本输入。

表6　文本输入组件错误检测情况(Akaunting)

Table 6　Detection results of text input component faults (Akaunting)

方法	30次测试			30 min测试		
	错误检测率/%	平均尝试次数	错误类型	错误检测率/%	平均用时/s	错误类型
LTICT	56.16	12.76	6.00	58.19	13.61	6.00
WebExplor	38.04	18.07	3.00	38.04	18.52	3.00
DBInputs	41.30	16.98	2.00	41.85	17.39	2.00
QTypist	45.58	14.74	4.00	46.41	16.65	4.00

表7　文本输入组件错误检测情况(Mantis)

Table 7　Detection results of text input component faults (Mantis)

方法	30次测试			30 min测试		
	错误检测率/%	平均尝试次数	错误类型	错误检测率/%	平均用时/s	错误类型
LTICT	40.00	17.98	5.00	40.31	18.46	5.00
WebExplor	33.33	20.33	3.00	33.33	20.67	3.00
DBInputs	28.33	21.75	4.00	28.75	22.21	4.00
QTypist	32.78	19.25	4.00	32.85	19.75	4.00

表8　文本输入组件错误检测情况(Tricentis)

Table 8　Detection results of text input component faults (Tricentis)

方法	30次测试			30 min测试		
	错误检测率/%	平均尝试次数	错误类型	错误检测率/%	平均用时/s	错误类型
LTICT	17.22	22.75	5.00	17.78	23.25	5.00
WebExplor	16.94	22.75	4.00	16.94	22.75	4.00
DBInputs	15.56	22.92	4.00	15.42	23.17	4.00
QTypist	16.67	22.83	5.00	17.22	23.08	5.00

表9展示了LTICT和基线方法在3个实验对象中检测错误类型的情况。可以看到,WebExplor、DBInputs和QTypist检测到的错误类型均能被LTICT检测到,且有3种错误类型仅LTICT能够检测到。我们进一步分析了LTICT能够检测到而基线方法无法检测到的错误类型。以实验对象Mantis为例,LTICT通过生成包含表情符号和脚本语句(如“<script>alert(“XSS”)</script>”)的异常输入,成功触发了应用程序服务器端错误(错误状态码为500),而WebExplor、DBInputs和QTypist均难以生成此类异常输入数据。这是因为WebExplor和DBInputs生成的文本输入类型较为单一,未考虑违反文本输入组件约束关系的异常文本输入,检测出的错误类型较少且检测效率较低。对于基线工具QTypist,它在设计时并未考虑文本输入组件潜在的约束关系,且使用LLM直接生成测试数据的效率较低,无法生成更具针对性和更多样化的异常文本输入,因而检测出的错误类型较少。而LTICT通过分析测试反馈并结合LLM合成程序,可以生成多种违反文本输入组件约束关系的异常文本输入,只需要较短时间就能检测到更多错误类型。

表9　LTICT、DBInputs、WebExplor和QTypist检测错误类型统计

Table 9　Statistics of fault types detected by LTICT, DBInputs, WebExplor, and QTypist

实验对象	错误状态码和JavaScript异常	方法			
		LTICT	WebExplor	DBInputs	QTypist
Akaunting	400	√	×	×	√
	403	√	×	×	×
	422	√	√	√	√
	429	√	√	×	√
	Type Error	√	×	×	×
	Network Error	√	√	√	√
Mantis	400	√	√	√	√
	404	√	√	√	√
	500	√	×	×	×
	Network Error	√	×	√	√
	Script Message Error	√	√	√	√
Tricentis	400	√	√	√	√
	401	√	×	×	√
	403	√	√	√	√
	404	√	√	√	√
	500	√	√	√	√

注:√表示能够检测特定错误类型,×表示不能检测。

针对RQ1的结论:LTICT检测Web应用中文本输入组件错误的能力更强。相比于自动化测试工具WebExplor、DBInputs和QTypist,LTICT检测文本输入组件错误的数量分别提升了34.21%、37.84%和8.51%,生成异常文本输入的平均错误检测率分别提升了30.03%、34.14%和19.92%。此外,在检测文本输入组件错误平均用时方面,LTICT比WebExplor、DBInputs和QTypist分别减少了10.69%、11.87%和6.99%。

4.2　针对RQ2的实验结果分析

为评估LTICT的各模块对检测文本输入组件错误性能的影响,我们分别移除有效文本输入生成、推断约束关系生成、批量异常文本输入生成和测试反馈构建等模块,并与LTICT在文本输入组件错误类型较多的实验对象Akaunting上进行对比。其中,移除有效文本输入生成、推断约束关系生成、批量异常文本输入生成和测试反馈构建的LTICT变体分别表示为LTICT w/o Vaild Input、LTICT w/o Inferred Constraints、LTICT w/o Batch Generation和LTICT w/o Feedback。

从表10中可以看出，LTICT在错误检测率、平均尝试次数以及平均用时上均优于LTICT w/o Vaild Input等4种变体。其中，LTICT w/o Feedback的测试效果最差，相较于LTICT，错误检测率分别降低了27.78%（30次测试）和32.72%（30 min测试），平均尝试次数和平均用时分别增加了9.25%和13.37%。这是由于LTICT w/o Feedback无法根据上下文信息推断文本输入组件之间的约束关系，且未有效利用测试结果的反馈信息，从而生成大量无效文本输入，导致测试性能下降。

表10 LTICT及其变体测试文本输入组件性能对比

Table 10 Comparison of text input component testing performance between LTICT and its variants

方法	30次测试		30 min测试	
	错误检测率/%	平均尝试次数	错误检测率/%	平均用时/s
LTICT	56.16	12.76	58.19	13.61
LTICT w/o Vaild Input	50.91	13.05	55.78	14.02
LTICT w/o Inferred Constraints	45.08	14.33	40.65	15.50
LTICT w/o Batch Generation	49.45	12.87	44.26	17.39
LTICT w/o Feedback	40.56	13.94	39.15	15.43

此外，LTICT w/o Vaild Input使用自定义值（如空值）来生成异常文本输入，仅能触发部分文本输入组件的错误，导致其错误检测率相比LTICT分别降低了9.35%（30次测试）和4.14%（30 min测试）。LTICT w/o Inferred Constraints利用有效文本输入和从HTML标签中提取的约束关系来生成异常文本输入。尽管该方法能够根据测试反馈调整输入数据类型，但由于未使用推断约束关系，生成异常文本输入的针对性较差，检测文本输入组件错误的平均尝试次数和平均用时相比LTICT分别增加了12.30%和13.89%。LTICT w/o Batch Generation检测文本输入组件错误的平均用时相比LTICT增加了27.77%。这是由于其未使用LLM合成程序，生成异常文本输入需要和LLM进行更频繁的交互，导致生成效率较低，消耗更多测试资源。相比之下，LTICT使用LLM合成的程序可快速产生大量异常文本输入，能够持续对文本输入组件进行测试，因此测试效率更高。

针对RQ2的结论：各组件均有助于提升LTICT检测文本输入组件错误的能力。其中，LTICT相比未使用测试反馈的LTICT w/o Feedback错误检测率提升最大，测试30次和测试30 min分别提升了27.78%和32.72%。LTICT相比未使用LLM合成程序批量生成异常文本输入的LTICT w/o Batch Generation检测文本输入组件错误的用时缩短最大，平均用时缩短了27.77%。

4.3 针对RQ3的实验结果分析

为评估使用LTICT生成的有效文本输入作为输入数据能否提升基线工具探索Web应用界面的性能，我们选取LTICT在执行30 min测试后，最后一轮测试反馈所生成的有效文本输入作为WebExplor和DBInputs的输入数据，并将对应的变体分别表示为$\text{WebExplor}_{\text{LTICT}}$和$\text{DBInputs}_{\text{LTICT}}$，与WebExplor和DBInputs进行比较。

表11为WebExplor、DBInputs及其变体探索4个实验对象中界面的情况。可以看出，$\text{WebExplor}_{\text{LTICT}}$探索Web应用界面的平均数量比WebExplor提升了14.23%。其中，对于实验对象Tricentis，$\text{WebExplor}_{\text{LTICT}}$相比WebExplor探索应用界面的平均数量提升了23.02%。这是因为WebExplor使用随机字符串填充文本输入组件，未考虑文本输入组件的约束关系。而$\text{WebExplor}_{\text{LTICT}}$在探索过程中会输入满足文本输入组件约束关系的有效文本输入，成功触发应用的提交操作，从而引起应用的URL发生变化，使其探索Web应用界面的数量高于WebExplor。

表11 WebExplor、DBInputs及其变体探索Web应用界面情况

Table 11 Exploration of web application interfaces by WebExplor, DBInputs, and their variants

实验对象	探索Web应用界面数			
	WebExplor	$\text{WebExplor}_{\text{LTICT}}$ (p-value)	DBInputs	$\text{DBInputs}_{\text{LTICT}}$ (p-value)
Akaunting	42.30	47.00($1.043\,6\times10^{-3}$)	37.00	40.30($4.258\,5\times10^{-3}$)
Mantis	36.50	40.50($4.007\,7\times10^{-3}$)	49.20	52.00($3.678\,7\times10^{-2}$)
Tricentis	27.80	34.20($1.133\,1\times10^{-2}$)	24.10	27.00($3.030\,4\times10^{-3}$)
Dolibarr	30.30	34.70($1.510\,7\times10^{-2}$)	34.20	31.90($4.907\,1\times10^{-2}$)
平均数量	34.23	39.10	36.13	37.80

$\text{DBInputs}_{\text{LTICT}}$探索Web应用界面的平均数量比DBInputs提升了4.62%。其中，对于实验对象Akaunting，$\text{DBInputs}_{\text{LTICT}}$相比DBInputs探索应用界面的平均数量提升了8.92%。这是因为DBInputs使用的测试数据库中存在大量空值，导致在部分文本输入组件中只能生成空值输入，限制了其探索应用界面的能力。而$\text{DBInputs}_{\text{LTICT}}$通过有效文本输入数据补充测试数据库，从而能够探索更多的Web应用界面。对于实验对象Dolibarr，$\text{DBInputs}_{\text{LTICT}}$相比DBInputs少探索到两个界面。分析原因后发现，要到达上述两个界面需要提交多组不同的文本输入数据，而$\text{DBInputs}_{\text{LTICT}}$的测试数据库仅包含一组有效文本输入数据，限制了其探索能力。若使用LTICT生成多组有效文本输入并将

其加入 $DBInputs_{LTICT}$ 的测试数据库中,$DBInputs_{LTICT}$ 将同样可以探索到目标应用界面。

此外,为验证 LTICT 生成的有效文本输入作为输入数据对 WebExplor 和 DBInputs 探索 Web 应用界面的性能是否有显著影响,我们使用 t 检验来比较基线工具探索 Web 应用界面数量的结果。t 检验的原假设为 LTICT 生成的有效文本输入作为输入数据对 WebExplor 和 DBInputs 探索 Web 应用界面的性能没有显著影响。如表 11 所示,$WebExplor_{LTICT}$ 和 $DBInputs_{LTICT}$ 在探索 Web 应用界面数量上的 p-value 均小于 0.05。因此,拒绝原假设,即 LTICT 生成的有效文本输入作为输入数据对 WebExplor 和 DBInputs 探索 Web 应用界面的性能具有显著影响。

针对 RQ3 的结论:使用 LTICT 生成的有效文本输入作为输入数据,能够提升基线工具探索 Web 应用界面的性能。相比未使用有效文本输入的 WebExplor 和 DBInputs,WebExplorLTICT 和 DBInputsLTICT 探索 Web 应用界面的平均数量分别提升了 14.23% 和 4.62%。

5　讨论

本节将分析 LTICT 生成的推断约束关系,以及使用不同 LLM 对 LTICT 的影响。

5.1　LTICT 生成的推断约束关系分析

LTICT 首先从被测应用的 HTML 文件中提取文本输入组件的约束关系,然后使用 LLM 进一步分析其应满足的推断约束关系,最后通过测试反馈对推断约束关系进行迭代优化。为评估 LTICT 生成推断约束关系的数量及合理性,我们人工分析了实验对象中的文本输入组件,统计其中存在的显式约束、隐式约束和组件间约束三类约束关系数,并与 LTICT 生成的推断约束关系进行对比分析。

图 7 展示了 LTICT 为各实验对象生成的推断约束关系情况,包括显式约束、隐式约束和组件间约束三类。图中紫色部分为人工分析发现的文本输入组件约束关系数,浅蓝色部分为 LLM 生成的推断约束关系数(其中,括号内数字代表经验证实际存在的约束关系数)。从图 7 中可以看出,LTICT 生成推断约束关系中,隐式约束占比较高,在四个实验对象中分别占 50.87%、47.27%、55.32% 和 43.27%,这表明 LTICT 能够充分利用 LLM 的语义理解和推理能力,从组件描述、界面语境以及测试反馈中挖掘出文本输入组件潜在的隐式约束。经计算,LTICT 生成推断约束关系的平均 Precision、Recall 和 F_1 分别为 79.83%、96.43% 和 87.21%。分析发现,尽管存在未能识别出某些特定约束的情况(如输入长度上限或重复输入限制),但仅出现在个别样例中,对整体结果影响较小。整体上看,LTICT 生成的推断约束关系较为充分,能够有效反映文本输入组件应满足的各类约束。

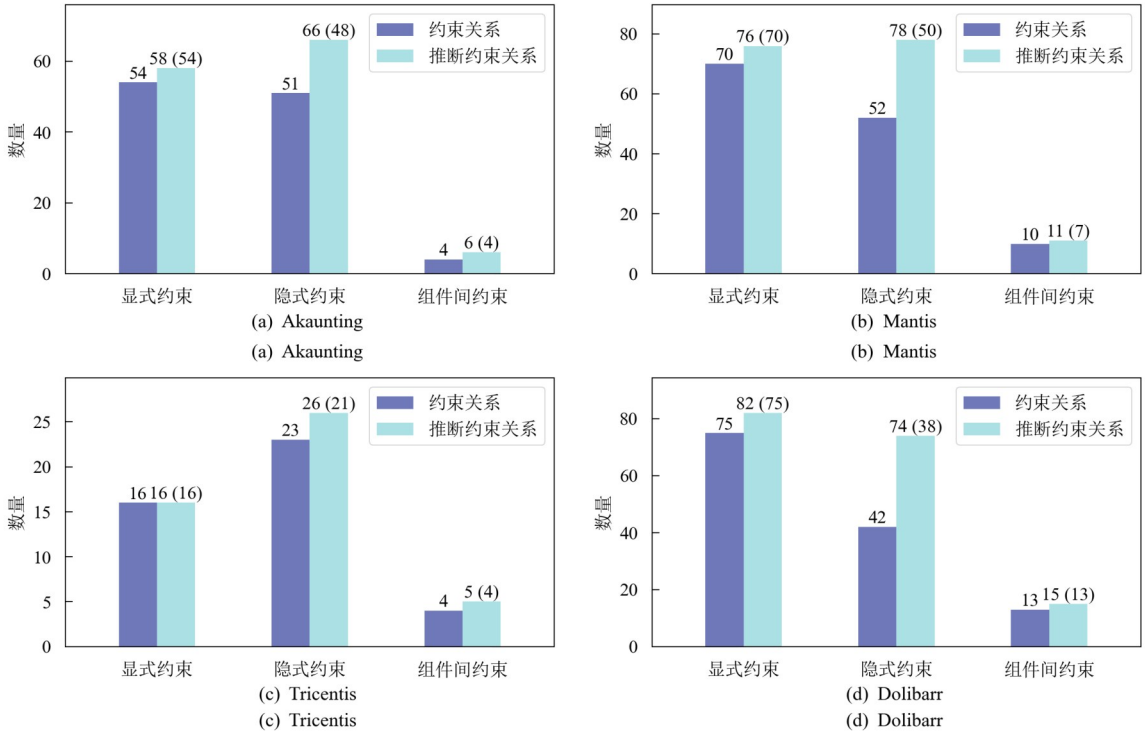


图 7　LTICT 生成推断约束关系情况

Figure 7　Inferred constraint relations generated by LTICT

5.2 使用不同LLM对LTICT的影响

LTICT使用LLM生成有效文本输入和推断约束关系，并用于引导LLM合成程序来批量生成异常文本输入，以对文本输入组件进行测试。为评估LTICT使用不同LLM的性能差异，我们选取在软件工程任务中广泛使用的claude-3-5-haiku和llama3-70b-instruct模型[20-21]，并与GPT-3.5-turbo模型进行对比分析。

表12为LTICT使用不同LLM检测各实验对象中文本输入组件错误的情况。可以看出，LTICT使用GPT-3.5-turbo模型检测文本输入组件错误的性能最优。其中，相较于claude-3-5-haiku模型和llama3-70b-instruct模型，使用GPT-3.5-turbo模型的LTICT在3个实验对象中平均错误检测率分别提升了13.08%和6.91%，检测文本输入组件错误的平均用时分别缩短了5.27%和5.11%，且平均尝试次数分别减少了4.05%和2.16%。因此，LTICT在实验中采用了GPT-3.5-turbo模型。

表12 LTICT使用不同LLM检测各实验对象中文本输入组件错误情况

Table 12 Detection of text input component faults in each experimental subject by LTICT using different LLMs

实验对象	LLM	30次测试		30 min测试	
		错误检测率/%	平均尝试次数	错误检测率/%	平均用时/s
Akaunting	GPT-3.5-turbo	56.16	12.76	58.19	13.61
	claude-3-5-haiku	48.77	14.13	49.46	15.57
	llama3-70b-instruct	52.90	13.59	53.55	15.30
Mantis	GPT-3.5-turbo	40.00	17.98	40.31	18.46
	claude-3-5-haiku	35.90	18.79	36.04	19.50
	llama3-70b-instruct	37.29	18.33	36.22	19.67
Tricentis	GPT-3.5-turbo	17.22	22.75	17.78	23.25
	claude-3-5-haiku	16.39	22.83	16.53	23.33
	llama3-70b-instruct	17.50	22.75	17.36	23.33

此外，为进一步分析LTICT调用LLM所消耗的时间，我们在表13中分别统计了LTICT在执行30 min测试时，调用不同LLM生成有效文本输入和推断约束关系，以及合成程序两个环节的平均用时。其中，LLM生成有效文本输入和推断约束关系的平均用时为59.29 s，合成程序的平均用时为88.11 s，调用LLM用时仅占整个测试过程的8.19%，占比较低。

表13 LTICT调用不同LLM的平均用时情况

Table 13 Average time spent by LTICT in calling different LLMs

LLM	生成有效文本输入和推断约束关系用时/s	合成程序用时/s	调用LLM用时占比/%
GPT-3.5-turbo	62.13	63.93	7.00
claude-3-5-haiku	45.27	63.63	6.05
llama3-70b-instruct	70.47	136.77	11.51
平均	59.29	88.11	8.19

6 结束语

本文提出了一种基于大语言模型的Web应用文本输入组件测试方法LTICT。该方法使用LLM推断文本输入组件的约束关系，并结合测试反馈，自动生成多样化的文本输入，以有效检测出应用中文本输入组件的错误。实验结果表明，LTICT检测文本输入组件错误的有效性和效率显著高于自动化测试工具WebExplor、DBInputs和QTypist。此外，使用LTICT生成的有效文本输入作为输入数据，可帮助WebExplor和DBInputs探索更多Web应用的界面。目前，LTICT仅支持检测应用中同一表单中多个文本输入组件间存在的约束关系。在未来研究工作中，我们计划进一步扩展LTICT所检测组件的范围，针对不同表单中文本输入组件间存在约束的情况进行分析。同时，将使用LTICT生成的输入数据填充AUTOE2E[22]等工具所生成的测试脚本，以自动探索更多应用状态，从而检测到更多类型的组件错误。

参考文献

[1] Liu Zhe, Chen Chunyang, Wang Junjie, et al. Testing the limits: Unusual text inputs generation for mobile app crash detection with large language model[C]//Proceedings of the IEEE/ACM 46th International Conference on Software Engineering. New York: ACM, 2024: 1-12.

[2] Wang Siyi, Wang Sinan, Fan Yujia, et al. Leveraging large vision-language model for better automatic web GUI testing[C]//2024 IEEE International Conference on Software Maintenance and Evolution. Piscataway: IEEE, 2024: 125-137.

[3] Liu Zhe, Chen Chunyang, Wang Junjie, et al. Fill in the blank: Context-aware automated text input generation for mobile GUI testing[C]//2023 IEEE/ACM 45th International Conference on Software Engineering. Piscataway: IEEE, 2023: 1355-1367.

[4] Clerissi D, Denaro G, Mobilio M, et al. Plug the database & play with automatic testing: Improving system testing by exploiting persistent data[C]//Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering. New York: ACM, 2021: 66-77.

[5] Sherin S, Muqeet A, Khan M U, et al. QExplore: An explo-

ration strategy for dynamic web applications using guided search[J]. Journal of Systems and Software, 2023, 195: 111512.

[6] Zheng Yan, Liu Yi, Xie Xiaofei, et al. Automatic web testing using curiosity-driven reinforcement learning[C]//2021 IEEE/ACM 43rd International Conference on Software Engineering. Piscataway: IEEE, 2021: 423-435.

[7] Clerissi D, Denaro G, Mobilio M, et al. DBInputs: Exploiting persistent data to improve automated GUI testing[J]. IEEE Transactions on Software Engineering, 2024, 50(9): 2412-2436.

[8] 许婷, 肖桐, 张圣林, 等. 基于LLM的日志故障诊断[J]. 电子学报, 2025, 53(4): 1123-1141.
Xu Ting, Xiao Tong, Zhang Shenglin, et al. Log fault diagnosis based on large language models[J]. Acta Electronica Sinica, 2025, 53(4): 1123-1141. (in Chinese)

[9] Azzam F, Saies A, Jaber M, et al. Evaluation for web GUI automation testing tool - experiment[C]//2022 International Symposium on Multidisciplinary Studies and Innovative Technologies. Piscataway: IEEE, 2022: 549-554.

[10] Mattiello G R, Endo A T. Model-based testing leveraged for automated web tests[J]. Software Quality Journal, 2022, 30(3): 621-649.

[11] Biagiola M, Stocco A, Ricca F, et al. Diversity-based web test generation[C]//Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. New York: ACM, 2019: 142-153.

[12] Biagiola M, Ricca F, Tonella P. Search based path and input data generation for web application testing[C]//Search Based Software Engineering. Cham: Springer, 2017: 18-32.

[13] Mariani L, Pezzè M, Riganelli O, et al. Link: Exploiting the web of data to generate test inputs[C]//Proceedings of the 2014 International Symposium on Software Testing and Analysis. New York: ACM, 2014: 373-384.

[14] Google. UI/application exerciser monkey[EB/OL]. (2025-07-27)[2025-09-30]. https://developer.android.com/studio/test/monkey.

[15] Li Yuanchun, Yang Ziyue, Guo Yao, et al. DroidBot: A lightweight UI-Guided test input generator for Android[C]//2017 IEEE/ACM 39th International Conference on Software Engineering Companion. Piscataway: IEEE, 2017: 23-26.

[16] He Yuyu, Zhang Lei, Yang Zhemin, et al. TextExerciser: Feedback-driven text input exercising for Android applications[C]//2020 IEEE Symposium on Security and Privacy. Piscataway: IEEE, 2020: 1071-1087.

[17] Alian P, Nashid N, Shahbandeh M, et al. Semantic constraint inference for web form test generation[C]//Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis. New York: ACM, 2024: 932-944.

[18] Liu Zhe, Chen Chunyang, Wang Junjie, et al. Make LLM a testing expert: Bringing human-like interaction to mobile GUI testing via functionality-aware decisions[C]//Proceedings of the IEEE/ACM 46th International Conference on Software Engineering. New York: ACM, 2024: 1-13.

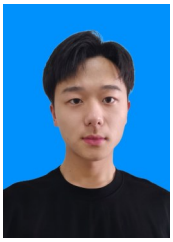
[19] Le T, Tran T, Cao D, et al. KAT: Dependency-aware automated API testing with large language models[C]//2024 IEEE Conference on Software Testing, Verification and Validation. Piscataway: IEEE, 2024: 82-92.

[20] Chen Jizheng, Du Kounianhua, Dai Xinyi, et al. DebateCoder: Towards collective intelligence of LLMs via test case driven LLM debate for code generation[C]//Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics. Stroudsburg: ACL, 2025: 12055-12065.

[21] Guo Yaoqi, Chen Zhenpeng, Zhang J M, et al. Personality-guided code generation using large language models[C]//Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics. Stroudsburg: ACL, 2025: 1068-1080.

[22] Alian P, Nashid N, Shahbandeh M, et al. Feature-driven end-to-end test generation[C]//2025 IEEE/ACM 47th International Conference on Software Engineering. Piscataway: IEEE, 2025: 450-462.

作者简介



张亚东 男,2002 年 2 月生。山东济宁人。现为北京信息科技大学硕士研究生。主要研究方向为软件分析及软件测试技术。

E-mail: zhangyadong@bistu.edu.cn

崔展齐 男,1984 年 2 月生。贵州金沙人。现为北京信息科技大学教授、博士生导师。主要研究方向为软件分析及软件测试技术。

E-mail: czq@bistu.edu.cn

兰文尉 男,1999 年 6 月生。北京人。硕士毕业于北京信息科技大学。主要研究方向为软件分析及软件测试技术。

E-mail: lww_lowry@bistu.edu.cn

徐伟利 男,2002 年 2 月生。天津人。现为北京信息科技大学硕士研究生。主要研究方向为智能化软件工程。

E-mail: weili_xu@bistu.edu.cn

曹鹤玲 女,1980 年 5 月生。河南南阳人。现为河南工业大学教授、博士生导师。主要研究方向为软件缺陷定位、软件缺陷修复和深度学习技术。

E-mail: caohl@haut.edu.cn