湖南大學

HUNAN UNIVERSITY

本科生毕业论文（设计）

论文(设计)题目： 川菜和湘菜在辣度上的对比研究

学 生 姓 名 ： 十三香

学 生 学 号 ： 201212313231

专 业 班 级 ： 食品工程

学 院 名 称 ： 后勤保障部

指 导 老 师 ： 王守义

2025 年 5 月 20 日

湖 南 大 学

毕业论文（设计）原创性声明

本人郑重声明：所呈交的论文（设计）是本人在导师的指导下独立进行研究所取得的研究成果。除了文中特别加以标注引用的内容外，本论文（设计）不包含任何其他个人或集体已经发表或撰写的成果作品。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律后果由本人承担。

学生签名：　　日期：2025 年　月　日

毕业论文（设计）版权使用授权书

本毕业论文（设计）作者完全了解学校有关保留、使用论文（设计）的规定，同意学校保留并向国家有关部门或机构送交论文（设计）的复印件和电子版，允许论文（设计）被查阅和借阅。本人授权湖南大学可以将本论文（设计）的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本论文（设计）。

本论文（设计）属于

1、保　密□，在_____年解密后适用于本授权书。

2、不保密□。

(请在以上相应方框内打“√”)

学生签名：　　日期：2025 年　月　日

导师签名：　　日期：2025 年　月　日

川菜和湘菜在辣度上的对比研究

摘　　要

摘要是论文内容的简要陈述，是一篇具有独立性和完整性的短文。摘要应包括本论文的创造性成果及其理论与实际意义。摘要中不宜使用公式、图表，不标注引用文献编号。避免将摘要写成目录式的内容介绍。比如上面是一段，下面是第二段。

关键词： 关键词1；关键词2；关键词3；关键词4

A Research Conducted on the Spiciness between SiChuan Cusine and Hunan Cusine

Abstract

An abstract is a brief summary of a research article, thesis, review, conference proceeding or any in-depth analysis of a particular subject and is often used to help the reader quickly ascertain the paper's purpose. When used, an abstract always appears at the beginning of a manuscript or typescript, acting as the point-of-entry for any given academic paper or patent application. Abstracting and indexing services for various academic disciplines are aimed at compiling a body of literature for that particular subject. As you can see, this paragraph has the same style as the previous one.

Key Words: Key Word1; Key Word 2; Key Word 3; Key Word 4

目　　录

毕业论文（设计）原创性声明和毕业论文（设计）版权使用授权书 …… I

摘 要 …… II

Abstract …… III

插图索引 …… VI

附表索引 …… VII

1 绪论 …… 1

1.1 动机 …… 1

1.2 贡献 …… 1

1.3 文章结构 …… 1

2 相关工作 …… 1

2.1 高级编程语言 …… 1

3 内容要求 …… 6

3.1 论文题目 …… 6

3.2 摘要与关键词 …… 6

3.3 目录 …… 6

3.4 正文 …… 6

3.5 结论 …… 7

3.6 参考文献 …… 7

3.7 致谢 …… 7

3.8 附录 …… 8

4 书写规范与打印要求 …… 8

4.1 文字和字数 …… 8

4.2 书写 …… 8
4.3 字体和字号 …… 8
4.4 封面 …… 8
4.5 论文页面设置 …… 9
4.6 摘要 …… 9
4.7 目录 …… 10
4.8 论文正文 …… 10
4.9 引用文献 …… 10
4.10 名词术语 …… 10
4.11 物理量名称符号及计量单位 …… 11
4.12 正体斜体用法规定 …… 11
4.13 数字 …… 11
4.14 公式 …… 11
5 讨论 …… 12
6 结语 …… 12
参考文献 …… 13
致谢 …… 26
附录 …… 27
附录 A …… 27
附录 B …… 28
附录 C …… 30

插图索引

图 2.1 JVM各症状的错误原因分布 …… 2

图 A.1 交换图例一 …… 27

图 A.2 交换图例二 …… 27

图 A.3 交换图例三 …… 27

图 A.4 性能统计图表 …… 28

图 B.1 所用代码集 …… 28

附表索引

表 2.1 JVM类型缺陷的收集结果…… 1

表 2.2 2022年JS引擎缺陷在源码文件中的分布 …… 4

表 2.3 Google开发的V8 …… 5

表 2.4 Mozilla开发的SpiderMonkey …… 5

表 B.1 数学字体对照表 …… 28

表 B.2 方法一 干扰抑制结果 …… 29

表 B.3 各组分 lg(B_i) 值 …… 29

表 B.4 电子文献标识类型 …… 29

表 B.5 参考文献标识类型 …… 30

表 C.1 本文所用缩略术语汇总 …… 30

1 绪论

1.1 动机

本文的动机在于……。

1.2 贡献

本文在优化机制上所做出的贡献是……。

1.3 文章结构

本文编排的结构组织如下：第 2 章介绍相关领域的研究工作；第 3 章介绍本文所做的具体工作；第 4 章介绍本文构建实验的方式和测试结果；第 5 章给出本文的讨论；第 6 章总结全文。

2 相关工作

2.1 高级编程语言

2.1.1 Java字节码解释器

Java虚拟机(JVM)转译和执行由Scala，Java，Groovy和Kotlin等高级编程语言编译得到的Java字节码。且当前的JVM系统由不同的组织和公司所开发，如Oracle开发的HotSpot，阿里巴巴开发的DragonWell和IBM开发的OpenJ9。

因JVM功能性复杂且规模巨大[1]，所以潜在的缺陷不可避免[2]。 Haoxiang Jia et al.[2]观察到HotSpot公开的缺陷报告在每年的总数上虽然下降，但其即时编译器部分出现的缺陷却显著增加，有的甚至可以引起JVM的崩溃。

Chaliasos,et al.[3]通过按照表 2.1 所示的缺陷追踪网站，从中收集并整理出 4153 个已被修复且由测试用例确保不会再重现的缺陷；其工作还随机挑选了320个缺陷，结合开发者讨论以及缺陷报告，总结形成了五类的缺陷以及其根因，结果如图 2.1 所示。

表 2.1 JVM类型缺陷的收集结果

语言	追踪网站	问题总数	最早记录时间	最近记录时间	缺陷收集	后-筛选量(post-filtering)
Java	Jira	10872	2004-02-11	2021-03-26	1252	873
Scala2	GitHub	12315	2003-05-22	2021-03-11	1180	1067
Scala3	GitHub	4286	2014-02-01	2021-03-21	429	366
Kotlin	YouTrack	40998	2011-10-28	2021-04-09	2189	1601
Groovy	Jira	9710	2003-09-25	2021-04-09	300	246

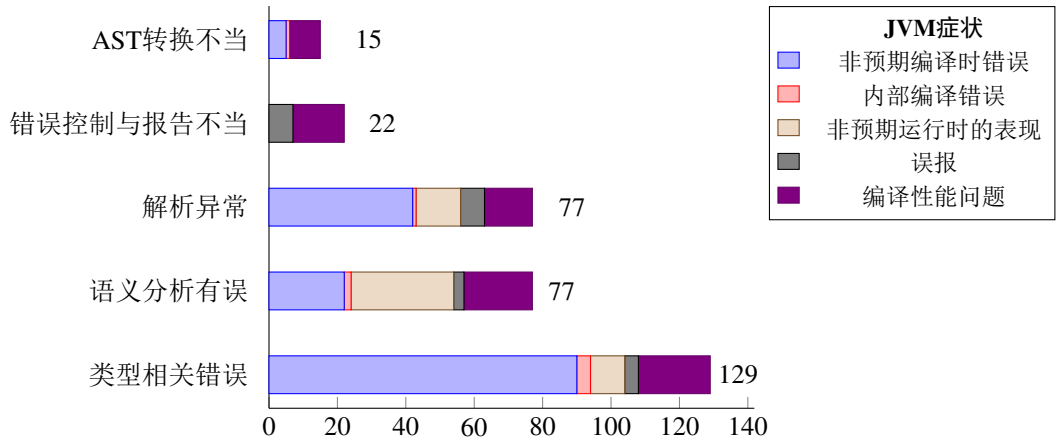


图 2.1　JVM各症状的错误原因分布

其中，AST转换不当是指编译器没有产生和原始输入程序在逻辑上保持一致的转换程序，占JVM缺陷的4.69%，会引发非预期的编译时错误和内部编译错误。错误控制与报告不当是指JVM解释器正确辨识出程序错误，但错误处理和报告机制没有产生预期的响应结果。错误控制与报告不当占JVM缺陷的6.88%，此类缺陷都与崩溃和错误报告有关。

解析异常则是既不能解析一个标识符（即变量名、方法名或类名），也不能正确地检索到先前解析过的标识名。解析异常也占24.06%，由两项原因所致：

(1) 解析算法执行错误，如JDK-7042566是在相同函数名下选择参数更多的一个来执行，从而引发错误；

(2) 编译器错误查询；

(3) 作用域状态错误。

语义分析有误则指的是解释器对特定程序代码生成了错误的分析结果，占JVM缺陷的24.06%，由两方面原因所致：

(1) 没有验证检查语义，如问题Scala2-5878；

(2) 不正确的分析机制，如问题Scala3-4487。

最后的类型相关错误指的是在前端编译器中运用类型系统内定义的规则以及类型中定义的操作应用于输入的源文件之中，但因实现不当而导致的错误。这一部分的实现不当是造成非预期编译时错误的主要原因，且以40.31%(129/320)的比例使之成为JVM最常见的缺陷。且细分类型相关错误则仍有三组：

(1) 不正确的类型推断和类型变量代替，如缺陷报告所报告的KT-10711；

(2) 不正确的类型（强制或非强制）转换，如缺陷报告所报告的KT-9630；

(3) 不正确的类型比较和边界计算，如缺陷报告所报告的JDK-8039214。

2.1.2 JavaScript字节码解释器

JavaScript(JS)是被广泛使用的编程语言之一，其迅速的发展由超过200万个可重用的npm软件库生态所支撑[4]。JS常作为客户端一侧的web应用使用，驱动了当今世界上超过90%的web页面[5]。这种广泛的应用使得保证JS的安全性至关重要[6]。

JS通过自身的执行引擎解释执行且主流浏览器提供商均实现并管理着其编写的js引擎。Google浏览器使用V8引擎，Mozilla的火狐浏览器使用SpiderMonkey引擎，苹果公司的Safari浏览器使用WebKit中的JS引擎，微软公司开发的Edge使用ChakraCore引擎。这些引擎是浏览器的核心[7]，依次将JS源码转变为字节码和机器码。

而Ziyuan Wang et al.[8]汇总分析了V8，SpiderMonkey和Chakra三个主流JS引擎截至2022年已有的19019篇缺陷(bugs)报告，16437次修正和805个测试用例以及随机挑选的540个缺陷的根因，发现：

(1) Compiler和DOM是存在缺陷最多的部件，结果如表 2.2 和表 2.3 和表 2.4 所示；

(2) 小型测试用例足以触发多数漏洞，如V8中70.79%的触发漏洞测试用例代码行数小于10行；

(3) JS引擎漏洞修复通常不复杂，多数仅需修改少量代码行和文件。如V8中76.86%的漏洞修复涉及代码行数少于100行；

(4) 对于修复缺陷的时间，V8中有80.33%的缺陷在半年内修复；SpiderMonkey引擎中有91.9%的缺陷是在半年内修复，且有4.33%是需要超过一年的时间才能修复；

(5) 规范语义缺陷是最常见的根因。除此之外，执行缺陷、特性以及函数调用缺陷均比其它类型的缺陷多。与V8和Chakra相比，SpiderMonkey有更多构建错误和更少内存错误。

研究为确保后续研究的可复现性，还将其成果分别以BSD，MPL，MIT许可证颁布公开于Github上[1,2,3]。

[1]https://github.com/njuptw/Empirical-Study-V8.

[2]https://github.com/njuptw/Empirical-Study-Chakra.

[3]https://github.com/njuptw/Empirical-Study-SpiderMonkey

表 2.2 2022年JS引擎缺陷在源码文件中的分布

V8		SpiderMonkey		Chakra	
源文件	缺陷数量	源码	缺陷数量	源码	缺陷数量
src/compiler	10525	js	3209	lib/Runtime/Library	4629
src/wasm	5070	dom	3082	lib/Runtime/Language	1984
src/builtins	4726	browser	2129	lib/Runtime/Base	1170
src/heap	3630	toolkit	1956	lib/Common/Memory	1004
src/objects	3087	third_party	1933	lib/Runtime/ByteCode	1001
src/runtime	2571	gfx	1807	lib/Runtime/Types	883
src/parsing	2147	devtools	1136	lib/wabt/src	676
src/interpreter	1779	taskcluster	1022	lib/Runtime/Debug	582
src/crankshaft	1178	layout	1013	lib/WasmReader	514
src/ast	1141	intl	962	lib/Jsrt	482
src/full-codegen	1089	netwerk	926	lib/Backend/arm64	264
src/debug	1073	modules	910	lib/Common/DataStructures	242
src/js	1049	js/src/jit	898	lib/common	240
src/ic	1034	mobile	730	lib/Common/Core	238
src/snapshot	1029	security	699	lib/Runtime/PlatformAgnostic	219
src/objects.cc	951	widget	540	lib/Backend/Lower.cpp	211
src/regexp	911	dom/media	534	lib/Backend/amd64	187
src/arm64	872	dom/base	468	lib/Parser/Parse.cpp	184
src/ia32	839	dom/ipc	367	lib/Backend/arm	170
src/arm	811	docshell	364	lib/Common/ConfigFlagsList.h	168

另外，S. Park et al.[9]围绕JS引擎的漏洞问题，实现了CRScope用于分类安全与非安全的缺陷，并评估了Exploitable和AddressSanitizer等工具和模型的性能。其工作所用的训练数据集从Google、Mozilla、Microsoft的Edge与IE浏览器以及GitHub库中收集。内容上包含165个触发安全缺陷的JS Poc以及174个触发非安全缺陷的PoC。同时，S. Park et al.还编译了与PoC对应的JS引擎，构成727个二进制文件以及766个实例，用于训练模型判断崩溃问题是否由安全缺陷所引发。其数据集公开[4]以支撑后续研究。

其训练模型和测试模型的方式采用了四折时间序列交叉验证方式，将所有安全和非安全漏洞按其在目标二进制文件中的提交日期排序，而后按升序排列崩溃实例并均分成5个桶，以保证每个桶中的崩溃实例数量大致相同(约153个)。评估实验结果显示，S. Park et al.所实现的CRScope则在Chakra、V8 和SpiderMonkey的崩溃实例上以0.85、0.89和0.93的准确度，优于Exniffer的0.51和AddressSanitizer的0.63。

[4]https://github.com/WSP-LAB/CRScope.

表 2.3 Google开发的V8

未修复缺陷（5191）		已修复缺陷（2827）	
组件名	数量	组件名称	数量
编译器	932	编译器	565
WebAssembly	855	JS语法	553
JS语法	831	Webassembly	518
运行时	648	运行时	317
基础架构	446	垃圾回收	232
垃圾回收	435	人类语言国际化	227
人类语言国际化	338	基础架构	182
API	253	API	111
构建文件	170	正则匹配	82
正则匹配	166	解释器	65

表 2.4 Mozilla开发的SpiderMonkey

未修复缺陷（4499）		已修复缺陷（3502）	
组件名	数量	组件名称	数量
DOM	519	DOM	428
常规	397	常规	330
JavaScript	294	JavaScript	252
图形	251	图形	211
自动化发布	152	自动化发布	143
页面布局	149	Web平台测试	138
Web平台测试	144	页面布局	127
网络	142	网络	101
音视频 1	14	WebRTC	88
WebRTC	107	音视频	79

2.1.3 Lua字节码解释器

Lua是一个基于栈虚拟机的解释型语言，支持动态类型[10]。Lua被设计为嵌入式语言，主要用于扩展应用程序，而非独立开发程序。它高度依赖宿主程序提供的API，标准库函数的加载也需由宿主程序显式完成[11]。Lua官方在其官网有单独开设一个页面用于记录其解释器存在的缺陷[5]。同时还提供了各个版本的手册[6]以方便编程人员查阅和学习。Lua高度依赖宿主程序提供的API，甚至标准库函数的加载也需由宿主程序显式完成[12]。

[5]https://www.lua.org/bugs.html.

[6]https://www.lua.org/manual/.

3 内容要求

3.1 论文题目

题目应该简短、明确、有概括性。通过题目，能大致了解论文内容、专业特点和学科范畴。但字数要适当，一般不宜超过20字。必要时可加副标题。

3.2 摘要与关键词

3.2.1 论文摘要

摘要应概括反映出毕业论文(设计)的内容、方法、成果和结论。摘要中一般不宜使用公式、图表，不标注引用文献编号。中文摘要以300左右字为宜、外文摘要以250个实词左右为宜。

3.2.2 关键词

关键词是供检索用的主题词条，应采用能覆盖论文（设计）主要内容的通用技术词条（参照相应的技术术语标准），尽量从《汉语主题词表》中选用，未被词表收录的新学科、新技术中的重要术语和地区、人物、文献等名称，也可作为关键词标注。关键词一般为3～8个，按词条的外延层次排列（外延大的排在前面）。关键词应以与正文不同的字体字号编排在摘要下方。多个关键词之间用分号分隔。中英文关键词应一一对应。

3.3 目录

目录按章、节、条三级标题编写，要求标题层次清晰。目录中的标题要与正文中标题一致。目录中应包括绪论、报告（论文）主体、结论、致谢、参考文献、附录等。

3.4 正文

正文是毕业论文(设计)的主体和核心部分，一般应包括绪论、报告（论文）主体及结论等部分。

3.4.1 绪论

绪论一般作为第一章，是毕业论文(设计)主体的开端。绪论应包括：毕业（设计）的背景及目的；国内外研究状况和相关领域中已有的成果；设计和研究方法；设计过程及研究内容等。绪论一般不少于1.5千字。

3.4.2 主体

主体是毕业论文(设计)的主要部分，应该结构合理、层次清楚、重点突出、文字简练、通顺。主体的内容应包括以下各方面：

1. 毕业论文(设计)总体方案设计与选择的论证。
2. 毕业论文(设计)各部分（包括硬件与软件）的设计计算。
3. 试验方案设计的可行性、有效性以及试验数据的处理及分析。
4. 对本研究内容及成果进行较全面、客观的理论阐述，应着重指出本研究内容中的创新、改进与实际应用之处。理论分析中，应将他人研究成果单独书写，并注明出处，不得将其与本人的理论分析混淆在一起。对于将其他领域的理论、结果引用到本研究领域者，应说明该理论的出处，并论述引用的可行性与有效性。
5. 自然科学的论文应推理正确，结论清晰，无科学性错误。
6. 管理和人文学科的论文应包括对所研究问题的论述及系统分析、比较研究，模型或方案设计，案例论证或实证分析，模型运行的结果分析或建议、改进措施等。

3.5 结论

结论是毕业论文(设计)的总结，是整篇设计报告（论文）的归宿。要求精炼、准确地阐述自己的创造性工作或新的见解及其意义和作用，还可进一步提出需要讨论的问题和建议。

3.6 参考文献

按正文中出现的顺序列出直接引用的主要参考文献。毕业论文(设计)的撰写应本着严谨求实的科学态度，凡有引用他人成果之处，均应按论文中所出现的先后次序列于参考文献中。并且只列出正文中以标注形式引用或参考的有关著作和论文。**一篇论著在论文中多处引用时，在参考文献中只能出现一次，序号以第一次出现的位置为准。**

3.7 致谢

致谢中主要感谢导师和对论文工作有直接贡献及帮助的人士和单位。

3.8 附录

对于一些不宜放入正文中、但作为毕业论文(设计)又是不可缺少的部分，或有重要参考价值的内容，可编入毕业论文(设计)的附录中。例如，过长的公式推导、重复性的数据、图表、程序全文及其说明等。

4 书写规范与打印要求

4.1 文字和字数

一般用汉语简化文字书写，字数在1.2万字左右，报告（内容）或软件说明书，字数在1万字左右。

4.2 书写

论文一律由本人在计算机上输入、编排并打印在A4幅面白纸上。毕业论文前置部分（即正文之前）一律用单面印刷，正文部分开始双面印刷。致谢和附录部分应单面起页双面印刷（如正文结束页为单页，则单数页背面不加页眉和页码，致谢单面起页，如致谢为单页，其背面亦不加页眉和页码）。

4.3 字体和字号

论文题目:	小2号黑体
章标题:	3号黑体
节标题:	小4号黑体
条标题:	小4号黑体
正文:	小4号宋体
页码:	5号宋体
数字和字母:	Times New Roman

4.4 封面

论文封面规范见（样张1），论文封皮一律采用白色铜版纸，封皮大小为A4规格。

4.5 论文页面设置

4.5.1 页眉和页脚

毕业论文各页均加页眉，在版心上边线隔一行1.5磅加粗、细双线（粗线在上），其上居中打印页眉。

页脚处居中插入页码，如“1”。

4.5.2 页边距

上边距：30mm；下边距：25mm；左边距：30mm；右边距：20mm；行间距为1.5倍行距。

4.5.3 页码的书写要求

论文页码从绪论部分开始，至附录，用阿拉伯数字连续编排，页码位于页脚居中。封面、摘要和目录不编入论文页码；摘要和目录用罗马数字单独编页码。

4.6 摘要

4.6.1 中文摘要

中文摘要包括：论文题目（小3号黑体）、“摘要”字样（3号黑体）、摘要正文(小4号宋体)和关键词。

摘要正文后下空一行打印“关键词”三字（4号黑体），关键词（小4号黑体）一般为3～5个，每一关键词之间用分号分开，最后一个关键词后不打标点符号，见（样张3）。

4.6.2 英文摘要

英文摘要另起一页，其内容及关键词应与中文摘要一致，并要符合英语语法，语句通顺，文字流畅。

英文题目的字样为：The title（小3号 Times New Roman 加粗）

英文摘要字样为：Abstract(3号 Times New Roman 加粗)

然后隔行书写摘要的文字部分。（字体为小4号Times New Roman）

摘要正文后下空一行打印关键词（4号Times New Roman加粗）：key word1；key word2；（关键词3－5个，小4号Times New Roman加粗）

英文和汉语拼音一律为Times New Roman体，字号与中文摘要相同，见（样张4）。

4.7 目录

专业目录的三级标题，建议按（1……、1.1……、1.1.1……）的格式编写，目录中各章题序的阿拉伯数字用Times New Roman体，第一级标题用小4号黑体，其余用小4号宋体。目录的打印实例见（样张5）。

4.8 论文正文

4.8.1 章节及各级标题

论文正文分章节撰写, 每章应另起一页。各章标题要突出重点、简明扼要。字数一般在15字以内, 不得使用标点符号。标题中尽量不采用英文缩写词，对必须采用者，应使用本行业的通用缩写词。

4.8.2 层次

层次以少为宜，根据实际需要选择。正文层次的编排和代号要求统一，层次为章（如“1”）、节（如“1.1”）、条（如“1.1.1”）、款（如“1.”）、项（如“（1）”）。层次用到哪一层次视需要而定，若节后无需“条”时可直接列“款”、“项”。“节”、“条”的段前、段后各设为0.5行，见（样张6）。

4.9 引用文献

引用文献标示方式应全文统一，并采用所在学科领域内通用的方式，所引文献编号用阿拉伯数字置于方括号中，用上标的形式置于所引内容最末句的右上角，如：“…成果[1]”，引用文献应与文中标注一致。几处地方引用同一个文献时，文中标注按第一次出现的序号。当提及的参考文献为文中直接说明时，其序号应该用阿拉伯数字与正文排齐，如“由文献[8, 10-14]可知”。

不得将引用文献标示置于各级标题处。

4.10 名词术语

科技名词术语及设备、元件的名称，应采用国家标准或部颁标准中规定的术语或名称。标准中未规定的术语要采用行业通用术语或名称。

全文名词术语必须统一。一些特殊名词或新名词应在适当位置加以说明或注解。采

用英语缩写词时, 除本行业广泛应用的通用缩写词外, 文中第一次出现的缩写词应该用括号注明英文全文。如返回导向编程（Return Oriented Programming, ROP）。

4.11 物理量名称符号及计量单位

4.11.1 物理量的名称和符号

物理量的名称和符号应符合GB3100 3102-86的规定。论文中某一量的名称和符号应统一。

4.11.2 物理量计量单位

物理量计量单位及符号应按国务院1984年发布的《中华人民共和国法定计量单位》及GB3100 3102执行, 不得使用非法定计量单位及符号。计量单位符号，除用人名命名的单位第一个字母用大写之外，其它一律用小写字母。

非物理量单位（如件、台、人、元、次等）可以采用汉字与单位符号混写的方式，如“万t·km”。

文稿叙述中不定数字之后允许用中文计量单位符号, 如“几千克至1000kg”。

表达时刻时应采用中文计量单位，如“上午8点3刻”，不能写成“8h45min”。

计量单位符号一律用正体。

4.12 正体斜体用法规定

物理量符号、物理常量、变量符号用**斜体**，计量单位等符号均用**正体**，见(样张7(1))。

4.13 数字

除习惯用中文数字表示的以外, 一般均采用阿拉伯数字。年份一概写全数，如2003年不能写成03年。

4.14 公式

公式应另起一行写在稿纸中央，公式和编号之间不加虚线。公式较长时最好在等号“＝”处转行。如难实现，则可在+、－、×、÷运算符号处转行，运算符号应写在转行后的行首，公式的编号用圆括号括起来放在公式右边行末。

公式序号按章编排，如第一章第一个公式序号为“(1.1)”, 附录A中的第一个公式

为“(A1)”等。

文中引用公式时，一般用“见式(1.1)”或“由公式(1.1)”。公式中用斜线表示“除”的关系时应采用括号, 以免含糊不清, 如$a/(b\cos x)$。通常“乘”的关系在前，如$a\cos x/b$而不写成$(a/b)\cos x$。

5 讨论

本文的提出的设计虽然能较好的完成预期目的，但日后仍应从...出发。

6 结语

本文就当前湘菜和川菜之间目前存在的问题做了汇总研究，并探讨、搭建了相关实验以优化其中的配料比重与锅底温度。

参考文献

[1] Atsuya Sonoyama, Takeshi Kamiyama, Masato Oguchi, et al. Performance Study of Kotlin and Java Program Considering Bytecode Instructions and JVM JIT Compiler[C]. in: 2021 Ninth International Symposium on Computing and Networking Workshops (CANDARW ’21). Matsue, Japan: IEEE, 2021: 127-133.

[2] Haoxiang Jia, Ming Wen, Zifan Xie, et al. Detecting JVM JIT Compiler Bugs via Exploring Two-Dimensional Input Spaces[C]. in: 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE ’23). Melbourne, Australia: IEEE, 2023: 43-55.

[3] Stefanos Chaliasos, Thodoris Sotiropoulos, Georgios-Petros Drosos, et al. Well-Typed Programs Can Go Wrong: A Study of Typing-Related Bugs in JVM Compilers[J]. Proceedings of the ACM on Programming Languages (PACMPL ’21), 2021, 5(OOPSLA): 1-30.

[4] Masudul HasanMasud Bhuiyan, AdithyaSrinivas Parthasarathy, Nikos Vasilakis, et al. SecBench.Js: An Executable Security Benchmark Suite for Server-Side JavaScript[C]. in: 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE ’23). Melbourne, Australia: IEEE, 2023: 1059-1070.

[5] Cristian-Alexandru Staicu, Michael Pradel, Benjamin Livshits. SYNODE: Understanding and Automatically Preventing Injection Attacks on Node.Js[C]. in: Network and Distributed System Security Symposium (NDSS ’18). San Diego, America: The Internet Society, 2018: 1-15.

[6] Jueon Eom, Seyeon Jeong, Taekyoung Kwon. Fuzzing JavaScript Interpreters with Coverage-Guided Reinforcement Learning for LLM-Based Mutation[C]. in: Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA ’24). Vienna, Austria: ACM, 2024: 1656-1668.

[7] Jungwon Lim, Yonghwi Jin, Mansour Alharthi, et al. SOK: On the Analysis of Web Browser Security[Z]. 2021.

[8] Ziyuan Wang, Dexin Bu, Nannan Wang, et al. An Empirical Study on Bugs in JavaScript Engines[J]. Information and Software Technology, 2023, 155(C): 1-16.

[9] Sunnyeo Park, Dohyeok Kim, Sooel Son. An Empirical Study of Prioritizing JavaScript Engine Crashes via Machine Learning[C]. in: Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security (Asia CCS '19). Auckland, New Zealand: ACM, 2019: 646-657.

[10] Roberto Ierusalimschy, Luiz Henrique de Figueiredo, Waldemar Celes. Lua 5.4 Reference Manual[A/OL]. 2020. https://www.lua.org/manual/5.4/.

[11] Petr Adámek. Security of the Lua Sandbox[D]. Prague, Czech Republic: Czech Technical University in Prague, 2022.

[12] Roberto Ierusalimschy. Programming in Lua: 5.3 edition[M]. Roberto Ierusalimschy, 2016.

[13] Marcos A. M. Vieira, MatheusS. Castanho, Racyus D. G. Pacífico, et al. Fast Packet Processing with eBPF and XDP: Concepts, Code, Challenges, and Applications[J]. ACM Computing Surveys (CSUR '20), 2020, 53(1): 1-36.

[14] Atsuya Osaki, Manuel Poisson, Seiki Makino, et al. Dynamic Fixed-Point Values in eBPF: A Case for Fully in-Kernel Anomaly Detection[C]. in: AINTEC '24: Proceedings of the Asian Internet Engineering Conference 2024 (Aintec '24). Sydney, Australia: ACM, 2024: 46-54.

[15] MarcoSpaziani Brunella, Giacomo Belocchi, Marco Bonola, et al. hXDP: Efficient Software Packet Processing on FPGA NICs[C]. in: 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI '20). New York, America: USENIX Association, 2020: 973-990.

[16] Yang Zhou, Zezhou Wang, Sowmya Dharanipragada, et al. Electrode: Accelerating Distributed Protocols with eBPF[C]. in: 20th USENIX Symposium on Networked Systems Design and Implementation (NSDI '23). Boston, America: USENIX Association, 2023: 1391-1407.

[17] Cyril Cassagnes, Lucian Trestioreanu, Clement Joly, et al. The rise of eBPF for non-intrusive performance monitoring[C]. in: 2020 IEEE/IFIP Network Operations and Management Symposium (NOMS '20). Budapest, Hungary: IEEE, 2020: 1-7.

[18] Yoann Ghigoff, Julien Sopena, Kahina Lazri, et al. BMC: Accelerating Memcached Using Safe in-Kernel Caching and Pre-Stack Processing[C]. in: 18th USENIX Symposium on Networked Systems Design and Implementation (NSDI '21). New York, America: USENIX Association, 2021: 487-501.

[19] Michał Zalewski. American Fuzzy Loop Whitepaper[EB/OL]. 2016. https://lcamtuf.coredump.cx/afl/echnical_details.txt.

[20] Arm Limited. Arm A-profile A64 Instruction Set Architecture[A]. 110 Fulbourn Road, Cambridge, England, 2025.

[21] Arm Limited. Armv8.5-A Memory Tagging Extension Whitepaper[Z]. 2018.

[22] Mark Rutland. ARMv8.3 Pointer Authentication[R]. Arm Limited, 2017.

[23] Arm Limited. Arm Architecture Registers Armv8, for Armv8-A architecture profile[A/OL]. 2020. https://developer.arm.com/documentation/ddi0601/2024-12/AArch32-Registers.

[24] 刘骐瑞. 面向 eBPF 字节码的攻防研究[D]. 浙江杭州: 浙江大学, 2024.

[25] Archana, Mayank. Azure Confidential Computing Meets eBPF[R]. Microsoft, 2024.

[26] Victor Bankowski. Dynamic Linking in WebAssembly: Architecture and Performance Evaluation[D]. Helsinki, Finland: University of Helsinki, 2021.

[27] TheophilusA. Benson, Prashanth Kannan, Prankur Gupta, et al. NetEdit: An Orchestration Platform for eBPF Network Functions at Scale[C]. in: Proceedings of the ACM SIGCOMM 2024 Conference (SIGCOMM '24). Sydney, Australia: ACM, 2024: 721-734.

[28] Shrenik Bhansali, Ahmet Aris, Abbas Acar, et al. A First Look at Code Obfuscation for WebAssembly[C]. in: Proceedings of the 15th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec '22). San Antonio, America: ACM, 2022: 140-145.

[29] Luke Nelson, Jacob VanGeffen, Emina Torlak, et al. Specification and Verification in the Field: Applying Formal Methods to BPF Just-in-Time Compilers in the Linux Kernel [C]. in: Proceedings of the 14th USENIX Conference on Operating Systems Design and Implementation (OSDI '20). New York, America: USENIX Association, 2020: 41-61.

[30] Brendan Gregg. BPF Performance Tools: Linux System and Application Observability[M]. 1st. Addison-Wesley Professional, 2019.

[31] RandalE. Bryant, DavidR. O'Hallaron, DavidRichard O'Hallaron. Computer Systems: A Programmer's Perspective[M]. Upper Saddle River, NJ: Prentice-Hall, 2003.

[32] Shangtong Cao, Ningyu He, Yao Guo, et al. WASMixer: Binary Obfuscation for WebAssembly[Z]. 2023. arXiv: 2308.03123.

[33] Intel Corporation. Control-flow Enforcement Technology Specification[A]. 2019.

[34] Milo Craun, Khizar Hussain, Uddhav Gautam, et al. Eliminating eBPF Tracing Overhead on Untraced Processes[C]. in: Proceedings of the SIGCOMM Workshop on eBPF and Kernel Extensions (eBPF '24). Sydney, Australia: ACM, 2024: 16-22.

[35] Craig Disselkoen, John Renner, Conrad Watt, et al. Position Paper: Progressive Memory Safety for WebAssembly[C]. in: Proceedings of the 8th International Workshop on Hardware and Architectural Support for Security and Privacy (HASP '19). Phoenix, America: ACM, 2019: 1-8.

[36] Jules Dejaeghere, Bolaji Gbadamosi, Tobias Pulls, et al. Comparing Security in eBPF and WebAssembly[C]. in: Proceedings of the 1st Workshop on EBPF and Kernel Extensions (eBPF '23). New York, America: ACM, 2023: 35-41.

[37] D. Thaler. RFC 9669: BPF Instruction Set Architecture (ISA)[EB/OL]. RFC Editor. 2024. https://www.rfc-editor.org/info/rfc9405.

[38] AlexanderJ. Gaidis, Joao Moreira, Ke Sun, et al. FineIBT: Fine-grain Control-flow Enforcement with Indirect Branch Tracking[C]. in: Proceedings of the 26th International Symposium on Research in Attacks, Intrusions and Defenses (RAID '23). Hong Kong, China: ACM, 2023: 527-546.

[39] Martin Fink, Dimitrios Stavrakakis, Dennis Sprokholt, et al. Cage: Hardware-Accelerated Safe WebAssembly[Z]. 2024. arXiv: 2408.11456.

[40] Leaning Technologies. An Enterprise-Grade C++ Compiler For The Web[Z]. https://leaningtech.com/cheerp/. 2024.

[41] Emscripten Contributors. emscripten[Z]. https://emscripten.org/. 2015.

[42] 李有霖. 基于模糊测试的eBPF漏洞挖掘技术研究[D]. 四川成都: 电子科技大学, 2023.

[43] Bolaji Gbadamosi, Luigi Leonardi, Tobias Pulls, et al. The eBPF Runtime in the Linux Kernel[Z]. 2024. arXiv: 2410.00026.

[44] Elazar Gershuni, Nadav Amit, Arie Gurfinkel, et al. Simple and Precise Static Analysis of Untrusted Linux Kernel Extensions[C]. in: Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '19). Phoenix, America: ACM, 2019: 1069-1084.

[45] Free Software Foundation, Inc. The GNU C Library (glibc) manual[A]. 2023.

[46] WebAssembly Community, Rossberg Andreas. WebAssembly Specification[S]. 2024.

[47] Andreas Haas, Andreas Rossberg, DerekL. Schuff, et al. Bringing the Web up to Speed with WebAssembly[C]. in: Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '17). Barcelona, Spain: ACM, 2017: 185-200.

[48] Jideng Han, Zhaoxin Zhang, Yuejin Du, et al. ESFuzzer: An Efficient Way to Fuzz WebAssembly Interpreter[J]. Electronics, 2024, 13(8): 1498-1513.

[49] Hao Sun, Zhendong Su. Validating the eBPF Verifier via State Embedding[C]. in: 18th USENIX Symposium on Operating Systems Design and Implementation (OSDI '24). Santa Clara, Canada: USENIX Association, 2024: 615-628.

[50] Peihua Zhang, Chenggang Wu, Xiangyu Meng, et al. HIVE: A Hardware-assisted Isolated Execution Environment for eBPF on AArch64[C]. in: 33rd USENIX Security Symposium (USENIX Security '24). Philadelphia, America: USENIX Association, 2024: 163-180.

[51] Hsin-Wei Hung, Ardalan AmiriSani. BRF: Fuzzing the eBPF Runtime[J]. Proceedings of the ACM on Software Engineering (PACMSE '24), 2024, 1(FSE): 1152-1171.

[52] Iulia Bastys, Maximilian Algehed, Alexander Sjösten, et al. SecWasm: Information Flow Control for WebAssembly[G]. in: Gagandeep Singh, Caterina Urban. Static Analysis: 29th

International Symposium (SAS '22). Auckland, New Zealand: Springer Nature Switzerland, 2022: 74-103.

[53] Intel Corporation. Intel Software Guard Extensions (Intel SGX) SDK for Linux OS[A]. 2023.

[54] Liu Qirui, Shen Wenbo, Zhou Jinmeng, et al. Interp-flow Hijacking: Launching Non-control Data Attack via Hijacking eBPF Interpretation Flow[C]. in: Computer Security – 29th European Symposium on Research in Computer Security, Bydgoszcz, Poland, September 16–20, 2024, Proceedings, Part III (ESORICS '24). Cham: Springer-Verlag, 2024: 194-214.

[55] Isovalent. eBPF Docs[EB/OL]. 2025. https://github.com/isovalent/ebpf-docs.

[56] Abhinav Jangda, Bobby Powers, EmeryD. Berger, et al. Not so Fast: Analyzing the Performance of WebAssembly vs. Native Code[C]. in: 2019 USENIX Annual Technical Conference (USENIX ATC '19). Renton, America: USENIX Association, 2019: 107-120.

[57] Jay Bosamiya, WenShih Lim, Bryan Parno. Provably-Safe Multilingual Software Sandboxing using WebAssembly[C]. in: 31st USENIX Security Symposium (USENIX Security '22). Boston, America: USENIX Association, 2022: 1975-1992.

[58] Jinghao Jia, Raj Sahu, Adam Oswald, et al. Kernel Extension Verification Is Untenable[C]. in: Proceedings of the 19th Workshop on Hot Topics in Operating Systems (HOTOS '23). Providence, America: ACM, 2023: 150-157.

[59] 江佩东. 高级语言程序编译到 WebAssembly 的安全特性研究[D]. 湖北武汉: 武汉大学, 2023.

[60] Evan Johnson, Evan Laufer, Zijie Zhao, et al. WaVe: A Verifiably Secure WebAssembly Sandboxing Runtime[C]. in: 2023 IEEE Symposium on Security and Privacy (S&P). San Francisco, America: IEEE, 2023: 2940-2955.

[61] Yuhong Zhong, Haoyu Li, YuJian Wu, et al. XRP: In-Kernel Storage Functions with eBPF [C]. in: 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI '22). Carlsbad, Canada: USENIX Association, 2022: 375-393.

[62] Eklektix Inc. x86: Kernel IBT[EB/OL]. 2023. https://lwn.net/ml/linux-kernel/20220224145138.952963315@infradead.org/.

[63] Juhee Kim, Jinbum Park, Sihyeon Roh, et al. TikTag: Breaking ARM's Memory Tagging Extension with Speculative Execution[EB/OL]. 2024. https://arxiv.org/abs/2406.08719. arXiv: 2406.08719.

[64] Matthew Kolosick, Shravan Narayan, Evan Johnson, et al. Isolation without Taxation: Near-Zero-Cost Transitions for WebAssembly and SFI[J]. Proceedings of the ACM on Programming Languages (PACMPL '22), 2022, 6(POPL): 1-30.

[65] Hsuan-Chi Kuo, Kai-Hsun Chen, Yicheng Lu, et al. Verified Programs Can Party: Optimizing Kernel Extensions via Post-Verification Merging[C]. in: Proceedings of the Seventeenth European Conference on Computer Systems (EuroSys '22). Rennes, France: ACM, 2022: 283-299.

[66] Chris Lattner. The architecture of open source applications (Volume 1) LLVM[EB/OL]. 2023. https://aosabook.org/en/v1/llvm.html.

[67] Daniel Lehmann, Johannes Kinder, Michael Pradel. Everything Old is New Again: Binary Security of WebAssembly[C]. in: 29th USENIX Security Symposium (USENIX Security '20). New York, America: USENIX Association, 2020: 217-234.

[68] Daniel Lehmann, MartinToldam Torp, Michael Pradel. Fuzzm: Finding Memory Bugs through Binary-Only Instrumentation and Fuzzing of WebAssembly[Z]. 2021. arXiv: 2110.15433.

[69] Daniel Lehmann. Program Analysis of WebAssembly Binaries[D]. Stuttgart Germany: Universität Stuttgart, 2022.

[70] Daniel Lehmann, Michael Pradel. Wasabi: A Framework for Dynamically Analyzing WebAssembly[C]. in: Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '19). Providence, America: ACM, 2019: 1045-1058.

[71] SooYee Lim, Tanya Prasad, Xueyuan Han, et al. SafeBPF: Hardware-assisted Defense-in-depth for eBPF Kernel Extensions[C]. in: Proceedings of the 2024 on Cloud Computing Security Workshop (CCSW '24). Salt Lake City, America: ACM, 2024: 80-94.

[72] Soo Yee Lim, Xueyuan Han, Pasquier Thomas. Unleashing Unprivileged eBPF Potential with Dynamic Sandboxing[C]. in: Proceedings of the 1st Workshop on eBPF and Kernel Extensions (eBPF '23). New York, America: ACM, 2023: 42-48.

[73] 张子君. Linux系统eBPF攻击建模及防护技术研究[D]. 浙江杭州: 浙江大学, 2024.

[74] The kerneldevelopment Community. The Linux Kernel documentation[A/OL]. 2024. https://docs.kernel.org/.

[75] Chris Lattner, Vikram Adve. LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation[C]. in: Proceedings of the International Symposium on Code Generation and Optimization: Feedback-Directed and Runtime Optimization (CGO '04). Palo Alto, America: IEEE Computer Society, 2004: 75-86.

[76] 江秋语. 基于LLVM的Android应用程序加固研究与实现[D]. 四川成都: 四川大学, 2021.

[77] LLVM Project. LLVM User Guides[A/OL]. 2025. https://llvm.org/docs/UserGuides.html.

[78] Jinsong Mao, Hailun Ding, Juan Zhai, et al. Merlin: Multi-tier Optimization of eBPF Code for Performance and Compactness[C]. in: Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3 (ASPLOS '24). La Jolla, America: ACM, 2024: 639-653.

[79] 陈青松. 基于ARM PAuth的内存防护研究[D]. 湖北武汉: 武汉大学, 2021.

[80] James Menetrey, Marcelo Pasin, Pascal Felber, et al. WaTZ: A Trusted WebAssembly Runtime Environment with Remote Attestation for TrustZone[C]. in: 2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS '22). Bologna, Italy: IEEE, 2022: 1177-1189.

[81] MohamedHusain Noor, Xiaoguang Wang, Binoy Ravindran. Understanding the Security of Linux eBPF Subsystem[C]. in: Proceedings of the 14th ACM SIGOPS Asia-Pacific Workshop on Systems (APSys '23). Seoul, Republic of Korea: ACM, 2023: 87-92.

[82] Martin Unterguggenberger, David Schrammel, Pascal Nasahl, et al. Multi-Tag: A Hardware-Software Co-Design for Memory Safety based on Multi-Granular Memory Tagging[C].

in: Proceedings of the 2023 ACM Asia Conference on Computer and Communications Security (AsiaCCS '23). New York, America: ACM, 2023: 177-189.

[83] RemziH. Arpaci-Dusseau, AndreaC. Arpaci-Dusseau. Operating Systems: Three Easy Pieces[M]. North Charleston, SC, USA: CreateSpace Independent Publishing Platform, 2018.

[84] 李浩, 古金宇, 夏虞斌, 等. 基于PKS硬件特性的eBPF内存隔离机制[J]. 软件学报, 2023, 34(12): 5921-5939.

[85] Christian Priebe, Divya Muthukumaran, Joshua Lind, et al. SGX-LKL: Securing the Host OS Interface for Trusted Execution[EB/OL]. 2020. https://arxiv.org/abs/1908.11143. eprint: 1908.11143.

[86] Shenghao Yuan, Frédéric Besson, Jean-Pierre Talpin. End-to-End Mechanized Proof of a JIT-accelerated eBPF Virtual Machine for IoT[C]. in: Arie Gurfinkel, Vijay Ganesh. Computer Aided Verification (CAV '24). Montreal, Canada: Springer Nature Switzerland, 2024: 325-347.

[87] Hanwen Lei, Ziqi Zhang, Shaokun Zhang, et al. Put Your Memory in Order: Efficient Domain-based Memory Isolation for WASM Applications[C]. in: Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS '23). Copenhagen, Denmark: ACM, 2023: 904-918.

[88] ParthaPratim Ray. An Overview of WebAssembly for IoT: Background, Tools, State-of-the-Art, Challenges, and Future Directions[J]. Future Internet, 2023, 15(8): 275-282.

[89] David Cerdeira, José Martins, Nuno Santos, et al. ReZone: Disarming TrustZone with TEE Privilege Reduction[C]. in: 31st USENIX Security Symposium (USENIX Security '22). Boston, America: USENIX Association, 2022: 2261-2279.

[90] Liz Rice. Learning eBPF: Programming the Linux Kernel for Enhanced Observability, Networking, and Security[M]. First edition. Sebastopol, CA: O'Reilly Media, 2023.

[91] Alan Romano, Xinyue Liu, Yonghwi Kwon, et al. An Empirical Study of Bugs in WebAssembly Compilers[C]. in: 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE '21). Melbourne, Australia: IEEE, 2021: 42-54.

[92] Raj Sahu, Dan Williams. Enabling BPF Runtime Policies for Better BPF Management[C]. in: Proceedings of the 1st Workshop on eBPF and Kernel Extensions (eBPF '23). New York, America: ACM, 2023: 49-55.

[93] WebAssembly Community. The Security Model of Web Assembly[A/OL]. 2025. https://webassembly.org/docs/security/.

[94] Ningyu He, Zhehao Zhao, Hanqin Guan, et al. SeeWasm: An Efficient and Fully-Functional Symbolic Execution Engine for WebAssembly Binaries[C]. in: Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '24). Vienna, Austria: ACM, 2024: 1816-1820.

[95] Kostya Serebryany. ARM memory tagging extension and how it improves C/C++ memory safety: 5[EB/OL]. 2019. https://www.usenix.org/system/files/login/articles/login_summer19_03_serebryany.pdf.

[96] Skanehira. Writting a Wasm Runtime in Rust[EB/OL]. 2024. https://github.com/skanehira/writing-a-wasm-runtime-in-rust/.

[97] Hao Sun, Yiru Xu, Jianzhong Liu, et al. Finding Correctness Bugs in eBPF Verifier with Structured and Sanitized Program[C]. in: Proceedings of the Nineteenth European Conference on Computer Systems (EuroSys '24). Athens, Greece: ACM, 2024: 689-703.

[98] Shravan Narayan, Craig Disselkoen, Daniel Moghimi, et al. Swivel: Hardening WebAssembly against Spectre[C]. in: 30th USENIX Security Symposium (USENIX Security '21). New York, America: USENIX Association, 2021: 1433-1450.

[99] BenL. Titzer. A fast in-place interpreter for WebAssembly[J]. Proceedings of the ACM on Programming Languages (PACMPL '22), 2022, 6(OOPSLA2): 1-27.

[100] Jämes Ménétrey, Marcelo Pasin, Pascal Felber, et al. Twine: An Embedded Trusted Runtime for WebAssembly[C]. in: 2021 IEEE 37th International Conference on Data Engineering (ICDE '21). Chania, Greece: IEEE, 2021: 205-216.

[101] Marcos A. M. Vieira, Matheus S. Castanho, Racyus D. G. Pacífico, et al. Fast Packet Processing with eBPF and XDP: Concepts, Code, Challenges, and Applications[J]. ACM Computing Surveys (CSUR '21), 2021, 53(1): 1-36.

[102] 王鹃, 樊成阳, 程越强 等. SGX技术的分析和研究[J]. 软件学报, 2018, 29(9): 2778-2798.

[103] Weili Wang, Honghan Ji, Peixuan He, et al. WAVEN: WebAssembly Memory Virtualization for Enclaves[J]. Network and Distributed System Security Symposium (NDSS '25), 2025, 1(1): 1-18.

[104] Weimin Chen, Zihan Sun, Haoyu Wang, et al. WASAI: uncovering vulnerabilities in Wasm smart contracts[C]. in: Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '22). New York, America: ACM, 2022: 703-715.

[105] Muhammad Waseem, Teerath Das, Aakash Ahmad, et al. Issues and Their Causes in WebAssembly Applications: An Empirical Study[C]. in: Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering (EASE '24). Salerno, Italy: ACM, 2024: 170-180.

[106] Yusheng Zheng, Tong Yu, Yiwei Yang, et al. Wasm-Bpf: Streamlining eBPF Deployment in Cloud Environments with WebAssembly[Z]. 2024.

[107] 张秀宏. WebAssembly原理与核心技术[M]. 北京: 机械工业出版社, 2020.

[108] WebAssembly Community. WebAssembly Dynamic Linking[A/OL]. 2025. https://github.com/WebAssembly/tool-conventions/blob/main/DynamicLinking.md.

[109] Wenlong Zheng, Baojian Hua. WASMDYPA: Effectively Detecting WebAssembly Bugs via Dynamic Program Analysis[C]. in: 2024 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER '24). Rovaniemi, Finland: IEEE, 2024: 296-307.

[110] Jämes Ménétrey, Marcelo Pasin, Pascal Felber, et al. A Comprehensive Trusted Runtime for WebAssembly With Intel SGX[J]. IEEE Transactions on Dependable and Secure Computing (TDSC '24), 2024, 21(4): 3562-3579.

[111] Conrad Watt. Mechanising and verifying the WebAssembly specification[C]. in: Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP '18). New York, America: ACM, 2018: 53-65.

[112] 庄骏杰, 胡霜, 华保健, 等. WebAssembly安全研究综述[J]. 计算机研究与发展, 2024, 61(8): 1-27.

[113] Yutian Yan, Tengfei Tu, Lijian Zhao, et al. Understanding the Performance of Webassembly Applications[C]. in: Proceedings of the 21st ACM Internet Measurement Conference (IMC '21). New York, America: ACM, 2021: 533-549.

[114] Yi He, Roland Guo, Yunlong Xing, et al. Cross Container Attacks: The Bewildered eBPF on Clouds[C]. in: 32nd USENIX Security Symposium (USENIX Security '23). Anaheim, Canada: USENIX Association, 2023: 5971-5988.

[115] Yixuan Zhang, Shangtong Cao, Haoyu Wang, et al. Characterizing and Detecting WebAssembly Runtime Bugs[J]. ACM Transactions on Software Engineering and Methodology (TOSEM '24), 2024, 33(2): 1-29.

[116] Yixuan Zhang, Mugeng Liu, Haoyu Wang, et al. Research on WebAssembly Runtimes: A Survey[J]. ACM Transactions on Software Engineering and Methodology (TOSEM '25), 2025, 1(1): 1-46.

[117] Ziyao Zhang, Wenlong Zheng, Baojian Hua, et al. VMCanary: Effective Memory Protection for WebAssembly via Virtual Machine-assisted Approach[C]. in: 2023 IEEE 23rd International Conference on Software Quality, Reliability, and Security (QRS '23). Chiang Mai, Thailand: IEEE, 2023: 662-671.

[118] Xiangwei Zhang, Junjie Wang, Xiaoning Du, et al. WasmCFuzz: Structure-aware Fuzzing for Wasm Compilers[C]. in: 2024 IEEE/ACM 4th International Workshop on Engineering and Cybersecurity of Critical Systems and 2024 IEEE/ACM Second International Workshop on Software Vulnerability (EnCyCriS/SVM '24). Lisbon, Portugal: ACM, 2024: 1-5.

[119] Yusheng Zheng, Tong Yu, Yiwei Yang, et al. Bpftime: Userspace eBPF Runtime for Uprobe, Syscall and Kernel-User Interactions[Z]. 2023.

[120] Yixuan Zhang, Shuyu Zheng, Haoyu Wang, et al. VM Matters: A Comparison of WASM VMs and EVMs in the Performance of Blockchain Smart Contracts[J]. ACM Transactions

on Modeling and Performance Evaluation of Computing Systems (ToMPECS ’24), 2024, 9(2): 1-24.

致 谢

时光荏苒岁月如梭，转眼自己在HNU的本科生涯就此画上了句号。回想过往的憧憬，愈发发觉有涯的人生实在无法穷尽渴望研习讨论的知识，愈发感受到愿景落空后的久萦心间的遗憾。我深深地感谢我的导师xxx，他最初向我提出了这个问题，他凭着非凡的直觉，坚持认为代数方法将是富有成果的。 I owe a deep dept to my advisor Professor xxx, who originally suggested the topic to me and who, with remarkable intuition, insisted that the algebraic approach would be fruitful.

附录 A

公式如 (A.1) 式所示。所形成的交换图如图 A.1，图 A.2，图 A.3 和图 A.4 所示。

$$\begin{aligned}\int \frac{x+\sin x}{1+\cos x}\mathrm{d}x &= \int \frac{x}{2\cos^2\frac{x}{2}}+\tan\frac{x}{2}\mathrm{d}x \\ &= \int x\mathrm{d}\tan\frac{x}{2}+\tan\frac{x}{2}\mathrm{d}x \\ &= \frac{x}{2}\tan\frac{x}{2}-\int\tan\frac{x}{2}\mathrm{d}x+\int\tan\frac{x}{2}\mathrm{d}x \\ &= x\tan\frac{x}{2}+C\end{aligned} \tag{A.1}$$

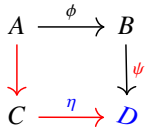
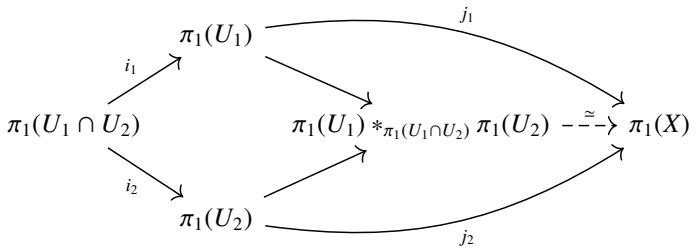
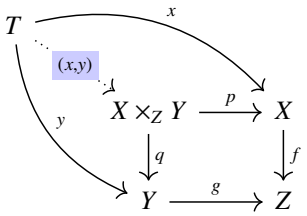


图 A.1　交换图例一



注：关系与图A.1类似，但不多.

图 A.2　交换图例二



注：关系与图A.1，图A.2类似，但不多.

图 A.3　交换图例三

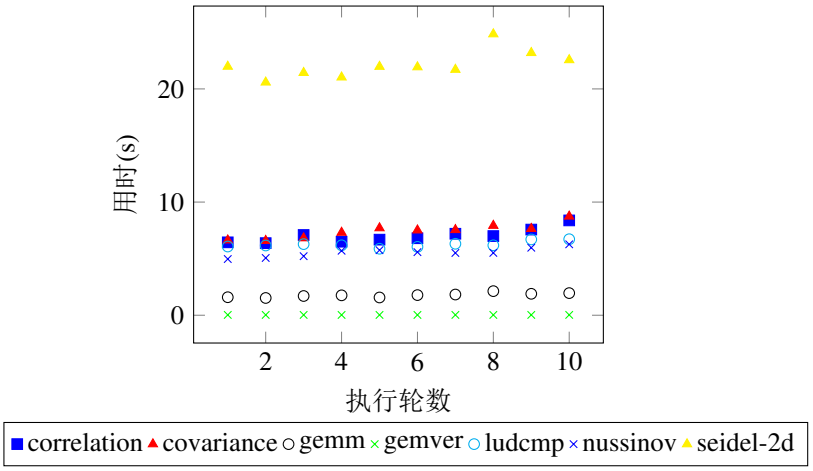
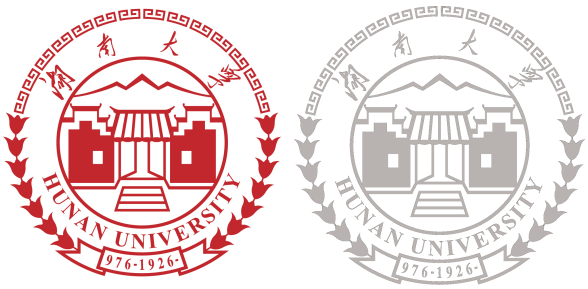


图 A.4　性能统计图表

附录 B



(a) 湖南　　(b) 大学

图 B.1　所用代码集

所用图片为图 B.1。所用表格为表 B.1 和表 B.3。

表 B.1　数学字体对照表

mathrm		mathbf		mathit		mathsf		mathtt		mathcal		mathbb		mathfrak		mathscr	
Aa	Nn	**Aa**	**Nn**	*Aa*	*Nn*	Aa	Nn	Aa	Nn	$\mathcal{A}a$	$\mathcal{N}n$	$\mathbb{A}$	$\mathbb{N}$	$\mathfrak{A}\mathfrak{a}$	$\mathfrak{N}\mathfrak{n}$	$\mathscr{A}a$	$\mathscr{N}n$
Bb	Oo	**Bb**	**Oo**	*Bb*	*Oo*	Bb	Oo	Bb	Oo	$\mathcal{B}b$	$\mathcal{O}o$	$\mathbb{B}$	$\mathbb{O}$	$\mathfrak{B}\mathfrak{b}$	$\mathfrak{O}\mathfrak{o}$	$\mathscr{B}b$	$\mathscr{O}o$
Cc	Pp	**Cc**	**Pp**	*Cc*	*Pp*	Cc	Pp	Cc	Pp	$\mathcal{C}c$	$\mathcal{P}p$	$\mathbb{C}$	$\mathbb{P}$	$\mathfrak{C}\mathfrak{c}$	$\mathfrak{P}\mathfrak{p}$	$\mathscr{C}c$	$\mathscr{P}p$
Dd	Qq	**Dd**	**Qq**	*Dd*	*Qq*	Dd	Qq	Dd	Qq	$\mathcal{D}d$	$\mathcal{Q}q$	$\mathbb{D}$	$\mathbb{Q}$	$\mathfrak{D}\mathfrak{d}$	$\mathfrak{Q}\mathfrak{q}$	$\mathscr{D}d$	$\mathscr{Q}q$
Ee	Rr	**Ee**	**Rr**	*Ee*	*Rr*	Ee	Rr	Ee	Rr	$\mathcal{E}e$	$\mathcal{R}r$	$\mathbb{E}$	$\mathbb{R}$	$\mathfrak{E}\mathfrak{e}$	$\mathfrak{R}\mathfrak{r}$	$\mathscr{E}e$	$\mathscr{R}r$
Ff	Ss	**Ff**	**Ss**	*Ff*	*Ss*	Ff	Ss	Ff	Ss	$\mathcal{F}f$	$\mathcal{S}s$	$\mathbb{F}$	$\mathbb{S}$	$\mathfrak{F}\mathfrak{f}$	$\mathfrak{S}\mathfrak{s}$	$\mathscr{F}f$	$\mathscr{S}s$
Gg	Tt	**Gg**	**Tt**	*Gg*	*Tt*	Gg	Tt	Gg	Tt	$\mathcal{G}g$	$\mathcal{T}t$	$\mathbb{G}$	$\mathbb{T}$	$\mathfrak{G}\mathfrak{g}$	$\mathfrak{T}\mathfrak{t}$	$\mathscr{G}g$	$\mathscr{T}t$
Hh	Uu	**Hh**	**Uu**	*Hh*	*Uu*	Hh	Uu	Hh	Uu	$\mathcal{H}h$	$\mathcal{U}u$	$\mathbb{H}$	$\mathbb{U}$	$\mathfrak{H}\mathfrak{h}$	$\mathfrak{U}\mathfrak{u}$	$\mathscr{H}h$	$\mathscr{U}u$
Ii	Vv	**Ii**	**Vv**	*Ii*	*Vv*	Ii	Vv	Ii	Vv	$\mathcal{I}i$	$\mathcal{V}v$	$\mathbb{I}$	$\mathbb{V}$	$\mathfrak{I}\mathfrak{i}$	$\mathfrak{V}\mathfrak{v}$	$\mathscr{I}i$	$\mathscr{V}v$
Jj	Ww	**Jj**	**Ww**	*Jj*	*Ww*	Jj	Ww	Jj	Ww	$\mathcal{J}j$	$\mathcal{W}w$	$\mathbb{J}$	$\mathbb{W}$	$\mathfrak{J}\mathfrak{j}$	$\mathfrak{W}\mathfrak{w}$	$\mathscr{J}j$	$\mathscr{W}w$
Kk	Xx	**Kk**	**Xx**	*Kk*	*Xx*	Kk	Xx	Kk	Xx	$\mathcal{K}k$	$\mathcal{X}x$	$\mathbb{K}$	$\mathbb{X}$	$\mathfrak{K}\mathfrak{k}$	$\mathfrak{X}\mathfrak{x}$	$\mathscr{K}k$	$\mathscr{X}x$
Ll	Yy	**Ll**	**Yy**	*Ll*	*Yy*	Ll	Yy	Ll	Yy	$\mathcal{L}l$	$\mathcal{Y}y$	$\mathbb{L}$	$\mathbb{Y}$	$\mathfrak{L}\mathfrak{l}$	$\mathfrak{Y}\mathfrak{y}$	$\mathscr{L}l$	$\mathscr{Y}y$

（表B.1接下页）

（续上页表B.1）

mathrm	mathbf	mathit	mathsf	mathtt	mathcal	mathbb	mathfrak	mathscr
Mm Zz	**Mm Zz**	*Mm Zz*	Mm Zz	Mm Zz	$\mathcal{M}m\ \mathcal{Z}z$	$\mathbb{M}\ \mathbb{Z}$	$\mathfrak{M}\mathfrak{m}\ \mathfrak{Z}\mathfrak{z}$	$\mathscr{M}m\ \mathscr{Z}z$

注：此数据仅为示例，实际数据可能有所不同.

表 B.2 方法一 干扰抑制结果

干扰类型	目标信号	阵元数	干扰采样值数	SINR(dB)
第一类干扰	信号1	8	——	30.58
		4	——	21.16
	信号4	8	——	38.28
		4	——	38.28
第二类干扰	信号4	8	30	4.69
			19	4.83
		4	30	-0.42

表 B.3 各组分 $\lg(B_i)$ 值

序号	T=1500K		T=2000K	
	组分	$\lg B_i$	组分	$\lg B_i$
1	O_2^+	5.26	HO_2	6.43
2	HO_2	5.26	O_2^+	6.42
3	H_2O^+	4.76	H_2O^+	6.18
4	N_2^+	3.97	H	6.12
5	H	3.54	H_2^+	6.04
6	OH	3.29	OH	5.91
7	CO^+	3.26	O	5.59
8	H_2^+	2.54	N_2^+	4.87
9	O	2.30	CO^+	3.98
10	H_2O_2	1.62	CO_2^+	3.76
11	CO_2^+	1.40	H_2O_2	3.09
12	HCO^*	-0.47	HCO^*	0.24
13	N^+	-4.85	N^+	-2.81
14	CH_2O^+	-6.91	CH_2O^*	-6.13
15	NO^+	-16.60	NO^+	-11.76

注：“+”表示重要成分，“*”表示冗余组分.

表 B.4 电子文献标识类型

类型	数据库（网上）	计算机程序（磁盘）	电子公告（网上）	光盘图书
文献标识符	DB(DB/OL)	CP(CP/DK)	EB(EB/OL)	M/CD

表 B.5 参考文献标识类型

类型	专著	会议录	（单篇论文）	报纸文章	期刊文章
文献标识符	M	C	A	N	J
类型	学位论文	报告	标准	专利	其它文献
文献标识符	D	R	S	P	Z

附录 C

表 C.1 本文所用缩略术语汇总

缩略名	全称及中文译名	首次出现页码
MMU	Memory Manage Unit, 内存管理单元.	
PC_1	Program Counter, 程序计数器.	
PC_2	Personal Computer, 个人电脑.	
ISA	Instruction Set Architecture, 指令集架构.	
DAG	Directed Acyclic Graph, 有向无环图.	
ROP	Return Oriented Programming, 返回导向编程.	