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 1 月　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。

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.

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 …… V

附表索引 …… VI

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

2.1.1 Java 字节码解释器 …… 1

2.1.2 JavaScript 字节码解释器 …… 2

2.1.3 Lua 字节码解释器 …… 5

3 eBPF 和 Web Assembly 的优化机制 …… 5

3.1 ebpf 的优化策略 …… 6

3.2 WebAssembly 的优化策略 …… 6

4 实验 …… 6

4.1 实验环境 …… 6

4.2 实验构建 …… 6

5 讨论 …… 6

6 结语 …… 6

参考文献 …… 7

致谢 …… 14

附录 …… 15

附录 A …… 15

附录 B …… 16

插图索引

图 2.1 JVM 各症状的错误原因分布[7]…… 2
图 A.1 交换图例一 …… 15
图 A.2 交换图例二 …… 15
图 A.3 交换图例三 …… 15
图 A.4 性能统计图表 …… 16
图 B.1 所用代码集 …… 16

附表索引

表 2.1 JVM 类型缺陷的收集结果[7] …… 1
表 2.2 2022 年 JS 引擎缺陷在源码文件中的分布[16] …… 4
表 2.3 2022 年 JS 引擎中各组件缺陷分布情况[16] …… 5
表 B.1 数学字体对照表 …… 17
表 B.2 各组分 lg(B_i) 值 …… 17

1 绪论

你好世界。

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 eBPF 和 wasm 等相关工作；第 3 章介绍本文在两者优化机制上所做的贡献；第 4 章介绍本文构建实验的方式和测试结果；第 5 章将给出本文的讨论；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[1]，阿里巴巴开发的 DragonWell[2]和 IBM 开发的 OpenJ9[3]。

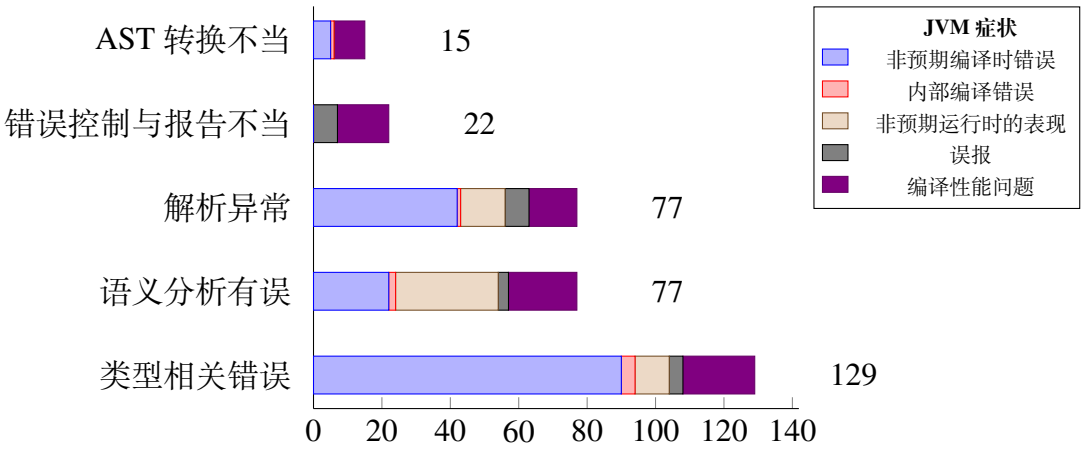
因 JVM 功能性复杂且规模巨大[4]，所以潜在的缺陷不可避免[5]。Jia,et al.[5]观察到 HotSpot 公开的缺陷报告在每年的总数上虽然下降，但其即时编译器部分出现的缺陷却显著增加，有的甚至可以引起 JVM 的崩溃[6]。

Chaliasos,et al.[7]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[7]

语言	缺陷追踪网站	链接	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	https://bugs.openjdk.java.net/rest/api/latest/search	10872	2004-02-11	2021-03-26	1252	873
Scala2	GitHub	https://api.github.com/repos/scala/bug	12315	2003-05-22	2021-03-11	1180	1067
Scala3	GitHub	https://api.github.com/repos/lampepfl/dotty	4286	2014-02-01	2021-03-21	429	366
Kotlin	YouTrack	https://youtrack.jetbrains.com/api/issues	40998	2011-10-28	2021-04-09	2189	1601
Groovy	Jira	https://issues.apache.org/jira/rest/api/2/search	9710	2003-09-25	2021-04-09	300	246

图 2.1 JVM 各症状的错误原因分布[7]



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[8]。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[9]。这种广泛的应用使得保证 JS 的安全性至关重要[10]。

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[11]，Mozilla 的火狐浏览器使用 SpiderMonkey 引擎[12]，苹果公司的 Safari 浏览器使用 WebKit[13]中的 JS 引擎，微软公司开发的 Edge 使用 ChakraCore 引擎[14]。这些引擎是浏览器的核心[15]，依次将 JS 源码转变为字节码和机器码。

而 Ziyuan Wang et al.[16]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 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[①②③]。

另外，基于 JavaScript 的 Node.js 在服务器端和桌面端也得到大量的应用[17]。因缺乏浏览器的安全沙箱机制，eval 和 exec 注入 API 可能会引发严重的危害[18]，同时因开发者对分析工具接受度低且修复意愿不足，模块间还存在责任推诿问题[9]。

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

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

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

表 2.2 2022 年 JS 引擎缺陷在源码文件中的分布[16]

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.[19]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[20]，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。

④https://github.com/WSP-LAB/CRScope.

表 2.3 2022 年 JS 引擎中各组件缺陷分布情况[16]

(a) 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

(b) 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[21]。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[22]。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[23]。

3 eBPF 和 Web Assembly 的优化机制

针对第 2 节提出的问题，本文从前端编译器、沙盒运行环境以及解释器出发，结合了 wasm

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

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

3.1 ebpf 的优化策略

本文对于 ebpf 采用了页表标记的手段，是从……。

3.2 WebAssembly 的优化策略

本文对于 wasm 的改进手段是……

4 实验

4.1 实验环境

本文用于搭建、采集的计算机配置为 Intel 11th Gen Intel(R) Core(TM) i5-1135G7@2.40 GHz，内存 16GB，操作系统为 Linux-6.5.1。

4.2 实验构建

管理学和人文社会学科的论文主体应包括对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。

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。

参考文献

[1] Oracle. Hotspot[Z]. https://openjdk.org/. 2024.

[2] Alibaba. DragonWell[Z]. https://github.com/dragonwell-project/dragonwell11. 2025.

[3] IBM. Openj9[Z]. https://eclipse.dev/openj9/. 2024.

[4] A. Sonoyama, 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). Matsue, Japan: IEEE, 2021: 127-133.

[5] H. Jia, 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). Melbourne, Australia: IEEE, 2023: 43-55.

[6] Webbug Group. Assert(false) failed: infinite loop in PhaseIterGVN::optimize[Z]. https://bugs.openjdk.org/browse/JDK-8290711. 2022.

[7] S. Chaliasos, 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, 2021, 5(OOPSLA): 1-30.

[8] M. H. M. Bhuiyan, 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). Melbourne, Australia: IEEE, 2023: 1059-1070.

[9] C.-A. Staicu, et al. SYNODE: Understanding and Automatically Preventing Injection Attacks on Node.Js[C]. in: Network and Distributed System Security Symposium. 2018.

[10] J. Eom, et al. 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. Vienna Austria: ACM, 2024: 1656-1668.

[11] Google. V8 javascript engine[Z]. https://chromium.googlesource.com/v8/. 2024.

[12] Mozilla. SpiderMonkey[Z]. https://spidermonkey.dev/. 2024.

[13] Apple. WebKit[Z]. https://webkit.org/. 2024.

[14] Microsoft. ChakraCore[Z]. https://github.com/chakra-core/ChakraCore. 2024.

[15] J. Lim, et al. SOK: On the Analysis of Web Browser Security[Z]. 2021. arXiv: 2112.15561 [cs].

[16] Z. Wang, et al. An Empirical Study on Bugs in JavaScript Engines[J]. Information and Software Technology, 2023, 155: 107105.

[17] F. Xiao, et al. Abusing Hidden Properties to Attack the Node.js Ecosystem[C]. in: 30th USENIX Security Symposium (USENIX Security 21). USENIX Association, 2021: 2951-2968.

[18] S. Holtskog. Security in Node.Js from Theory to Practice[D]. OSLO: UNIVERSITY OF OSLO, 2023.

[19] S. Park, et al. 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. Auckland New Zealand: ACM, 2019: 646-657.

[20] N. Le, et al. K-Fold Cross-Validation: An Effective Hyperparameter Tuning Technique in Machine Learning on GNSS Time Series for Movement Forecast[C]. in: A. Çiner, et al. Recent Research on Geotechnical Engineering, Remote Sensing, Geophysics and Earthquake Seismology. Cham: Springer Nature Switzerland, 2024: 377-382.

[21] R. Ierusalimschy, et al. Lua 5.4 Reference Manual[A/OL]. 2020. https://www.lua.org/manual/5.4/.

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

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

[24] C. Kim, et al. Short-Circuit Dispatch: Accelerating Virtual Machine Interpreters on Embedded Processors[C]. in: 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA). Seoul, South Korea: IEEE, 2016: 291-303.

[25] B. L. Titzer. A Fast In-Place Interpreter for WebAssembly[Z]. 2022. arXiv: 2205.01183.

[26] T. Bach, et al. The Impact of Coverage on Bug Density in a Large Industrial Software Project[C]. in: 2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM). Toronto, ON: IEEE, 2017: 307-313.

[27] S. Bhansali, 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. San Antonio TX USA: ACM, 2022: 140-145.

[28] T. Usui, et al. Bytecode Jiu-Jitsu Choking Interpreters to Force Execution of Malicious Bytecode[R]. PPT. United States of America: Black Hat, 2024.

[29] S. Cao, et al. WASMixer: Binary Obfuscation for WebAssembly: arXiv:2308.03123 [EB/OL]. arXiv. 2023 [2024-12-11]. https://doi.org/10.48550/arXiv.2308.03123. arXiv: 2308.03123.

[30] A. Celik, et al. Design, Implementation, and Application of GPU-based Java Bytecode Interpreters[J]. Proceedings of the ACM on Programming Languages, 2019, 3(OOPSLA): 1-28.

[31] E. D. Berger, et al. Triangulating Python Performance Issues with SCALENE[C]. in: 17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23). Boston, MA: USENIX Association, 2023: 51-64.

[32] R. Nystrom. Crafting Interpreters[M]. Erscheinungsort nicht ermittelbar: Verlag nicht ermittelbar, 2021.

[33] H. Barbosa, et al. cvc5: A Versatile and Industrial-Strength SMT Solver[C]. in: D. Fisman, et al. Tools and Algorithms for the Construction and Analysis of Systems. Cham: Springer International Publishing, 2022: 415-442.

[34] P. Daniel, et al. Discovering Vulnerabilities in WebAssembly with Code Property Graphs [C]. in: 2019.

[35] Dxo, et al. Hevm, a Fast Symbolic Execution Framework for EVM Bytecode[G]. in: A. Gurfinkel, et al. Computer Aided Verification: vol. 14681. Cham: Springer Nature Switzerland, 2024: 453-465.

[36] E. Güler, et al. Atropos: Effective Fuzzing of Web Applications for Server-Side Vulnerabilities[C]. in: 33rd USENIX Security Symposium (USENIX Security 24). Philadelphia, PA: USENIX Association, 2024: 4765-4782.

[37] Z. Feng, et al. A Bytecode-based Integrated Detection and Repair Method for Reentrancy Vulnerabilities in Smart Contracts[J]. IET Blockchain, 2024, 4(3): 235-251.

[38] R. Fang, et al. A Performance Survey on Stack-based and Register-based Virtual Machines [Z]. 2016. arXiv: 1611.00467 [cs].

[39] Z. Feng, et al. A Bytecode-based Integrated Detection and Repair Method for Reentrancy Vulnerabilities in Smart Contracts[J]. IET Blockchain, 2024, 4(3): 235-251.

[40] D. P. Friedman, et al. Essentials of Programming Languages[M]. 3rd ed. Cambridge, MA: MIT Press, 2008.

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

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

[43] E. Gershuni, 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. Phoenix AZ USA: ACM, 2019: 1069-1084.

[44] Google. ClusterFuzz[Z]. https://google.github.io/clusterfuzz/. 2024.

[45] WebAssembly Community, et al. WebAssembly Specification[A/OL]. 2024. https://github.com/WebAssembly/spec/.

[46] A. Haas, 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. Barcelona Spain: ACM, 2017: 185-200.

[47] D. Park. Halmos, a symbolic testing tool for EVM smart contracts[EB/OL]. 2024. https://github.com/a16z/halmos.

[48] Intel. Intelő 64 and IA-32 Architectures Software Developer's Manual Combined Volumes: 1, 2A, 2B, 2C, 2D, 3A, 3B, 3C, 3D, and 4[A/OL]. 2024. https://cdrdv2.intel.com/v1/dl/getContent/843820?fileName=325462-sdm-vol-1-2abcd-3abcd-4-1.pdf.

[49] Q. Liu, et al. Interp-Flow Hijacking: Launching Non-control Data Attack via Hijacking eBPF Interpretation Flow[G]. in: J. Garcia-Alfaro, et al. Computer Security – ESORICS 2024: vol. 14984. Cham: Springer Nature Switzerland, 2024: 194-214.

[50] Y. Klimiankou. Interpretizer:ăA Compiler-Independent Conversion of Switch-Based Dispatch into Threaded Code[C]. in: M. Mazzara, et al. Software Technology: Methods and Tools. Cham: Springer International Publishing, 2019: 59-72.

[51] D. Evans. Introduction to Computing: Explorations in Language, Logic, and Machines [M]. Version: August 15, 2011. Lexington, Ky., 2011.

[52] Y. Jiang, et al. Fuzzing the PHP Interpreter via Dataflow Fusion[Z]. 2024. arXiv: 2410.21713 [cs].

[53] E. Hildenbrandt, et al. KEVM: A Complete Formal Semantics of the Ethereum Virtual Machine[C]. in: 2018 IEEE 31st Computer Security Foundations Symposium (CSF). 2018: 204-217.

[54] O. Larose, et al. AST vs. Bytecode: Interpreters in the Age of Meta-Compilation[J]. Proceedings of the ACM on Programming Languages, 2023, 7(OOPSLA2): 318-346.

[55] D. Lehmann, et al. Everything Old is New Again: Binary Security of WebAssembly[C]. in: 29th USENIX Security Symposium (USENIX Security 20). USENIX Association, 2020: 217-234.

[56] D. Lehmann, et al. 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. Providence RI USA: ACM, 2019: 1045-1058.

[57] M. H. N. Mohamed, et al. Understanding the Security of Linux eBPF Subsystem[C]. in: Proceedings of the 14th ACM SIGOPS Asia-Pacific Workshop on Systems. Seoul Republic of Korea: ACM, 2023: 87-92.

[58] D. Liu, et al. Towards Understanding Bugs in Python Interpreters[J]. Empirical Software Engineering, 2023, 28(1): 19.

[59] CVE. A vulnerability in the implementation of the Lua interpreter[Z]. https://www.cve.org/CVERecord?id=CVE-2020-3423. 2020.

[60] CVE. Stack overflow in Lua Interpreter 5.1.0 5.4.4[Z]. https://www.cve.org/CVERecord?id=CVE-2021-43519. 2021.

[61] CVE. Use after free in garbage collector and finalizer of Lua interpreter 5.4.0 5.4.3[Z]. https://www.cve.org/CVERecord?id=CVE-2021-44964. 2021.

[62] C. Luo, et al. Reverse Engineering of Obfuscated Lua Bytecode via Interpreter Semantics Testing[J]. IEEE Transactions on Information Forensics and Security, 2023, 18: 3891-3905.

[63] A. Niemetz, et al. Bitwuzla at the SMT-COMP 2020[EB/OL]. 2020. https://arxiv.org/abs/2006.01621. arXiv: 2006.01621 [cs.LO].

[64] N. Popov. Optimizing PHP Bytecode Using Type-Inferred SSA Form[D]. Berlin, Germany: Technische Universität Berlin, 2016.

[65] S. Yuan, et al. End-to-End Mechanized Proof of a JIT-accelerated eBPF Virtual Machine for IoT[C]. in: A. Gurfinkel, et al. Computer Aided Verification. Cham: Springer Nature Switzerland, 2024: 325-347.

[66] J. Bosamiya, et al. Provably-Safe Multilingual Software Sandboxing using WebAssembly [C]. in: 31st USENIX Security Symposium (USENIX Security 22). Boston, MA: USENIX Association, 2022: 1975-1992.

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

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

[69] A. Romano, et al. An Empirical Study of Bugs in WebAssembly Compilers[C]. in: 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE). Melbourne, Australia: IEEE, 2021: 42-54.

[70] D. WeiSSer, et al. Security Analysis of PHP Bytecode Protection Mechanisms[G]. in: H. Bos, et al. Research in Attacks, Intrusions, and Defenses: vol. 9404. Cham: Springer International Publishing, 2015: 493-514.

[71] H. Chen, et al. Security Bugs in Embedded Interpreters[C]. in: Proceedings of the 4th Asia-Pacific Workshop on Systems. Singapore Singapore: ACM, 2013: 1-7.

[72] X. Lin, et al. On the Security of Python Virtual Machines: An Empirical Study[C]. in: 2022 IEEE International Conference on Software Maintenance and Evolution (ICSME). Limassol, Cyprus: IEEE, 2022: 223-234.

[73] M. Seitzer, et al. A Bytecode Interpreter for Secure Program Execution in Untrusted Main Memory[G]. in: G. Pernul, et al. Computer Security – ESORICS 2015: vol. 9327. Cham: Springer International Publishing, 2015: 376-395.

[74] C.-A. Staicu, et al. Bilingual Problems: Studying the Security Risks Incurred by Native Extensions in Scripting Languages[Z]. 2023. arXiv: 2111.11169 [cs].

[75] H. Sun, et al. Finding Correctness Bugs in eBPF Verifier with Structured and Sanitized Program[C]. in: Proceedings of the Nineteenth European Conference on Computer Systems. Athens Greece: ACM, 2024: 689-703.

[76] 李成扬, 等. 虚拟机软件保护技术综述[J]. 信息安全研究, 2022, 8(7), 675: 675-.

[77] Yixuan Zhang, et al. Research on WebAssembly Runtimes: A Survey[Z]. 2024. arXiv: 2404.12621 [cs].

[78] S. Narayan, et al. Swivel: Hardening WebAssembly against Spectre[C]. in: 30th USENIX Security Symposium (USENIX Security 21). USENIX Association, 2021: 1433-1450.

[79] Z. Wang, et al. An Empirical Study on Bugs in Php[J]. International Journal of Software Engineering and Knowledge Engineering, 2022, 32(06): 845-870.

[80] Z. Wang, et al. An Empirical Study on Bugs in Python Interpreters[J]. IEEE Transactions on Reliability, 2022, 71(2): 716-734.

[81] M. Waseem, et al. Issues and Their Causes in WebAssembly Applications: An Empirical Study: arXiv:2311.00646[EB/OL]. arXiv. 2024 [2024-11-28]. http://doi.org/10.48550/arXiv.2311.00646. arXiv: 2311.00646.

[82] CVE. Heap Overflow Vulnerability in wasm-micro-runtime[Z]. https://www.cve.org/CVERecord?id=CVE-2023-48105. 2023.

[83] CVE. Out-of-bound memory read vulnerability in wasm-micro-runtime[Z]. https://www.cve.org/CVERecord?id=CVE-2024-34250. 2024.

[84] CVE. Heap buffer overflow vulnerability in wasm-micro-runtime[Z]. https://www.cve.org/CVERecord?id=CVE-2024-34251. 2024.

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

[86] Yutian Yan, et al. Understanding the Performance of Webassembly Applications[C]. in: Proceedings of the 21st ACM Internet Measurement Conference. Virtual Event: ACM, 2021: 533-549.

[87] L. De Moura, et al. Z3: An Efficient SMT Solver[G]. in: D. Hutchison, et al. Tools and Algorithms for the Construction and Analysis of Systems: vol. 4963. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008: 337-340.

[88] Yixuan Zhang, et al. Characterizing and Detecting WebAssembly Runtime Bugs[J]. ACM Transactions on Software Engineering and Methodology, 2024, 33(2): 1-29.

[89] Q. Zhang, et al. RegCPython: A Register-based Python Interpreter for Better Performance [J]. ACM Transactions on Architecture and Code Optimization, 2023, 20(1): 1-25.

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

公式推导如 (A.1) 式所示。

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

所形成的交换图如图 A.1，图 A.2，图 A.3 和图 A.4 所示。

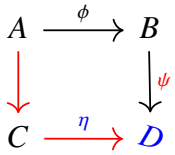
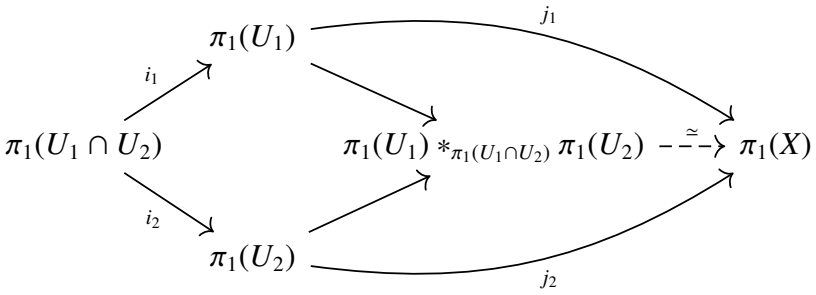
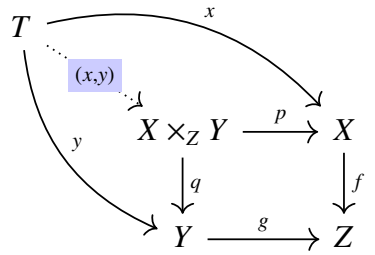


图 A.1　交换图例一



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

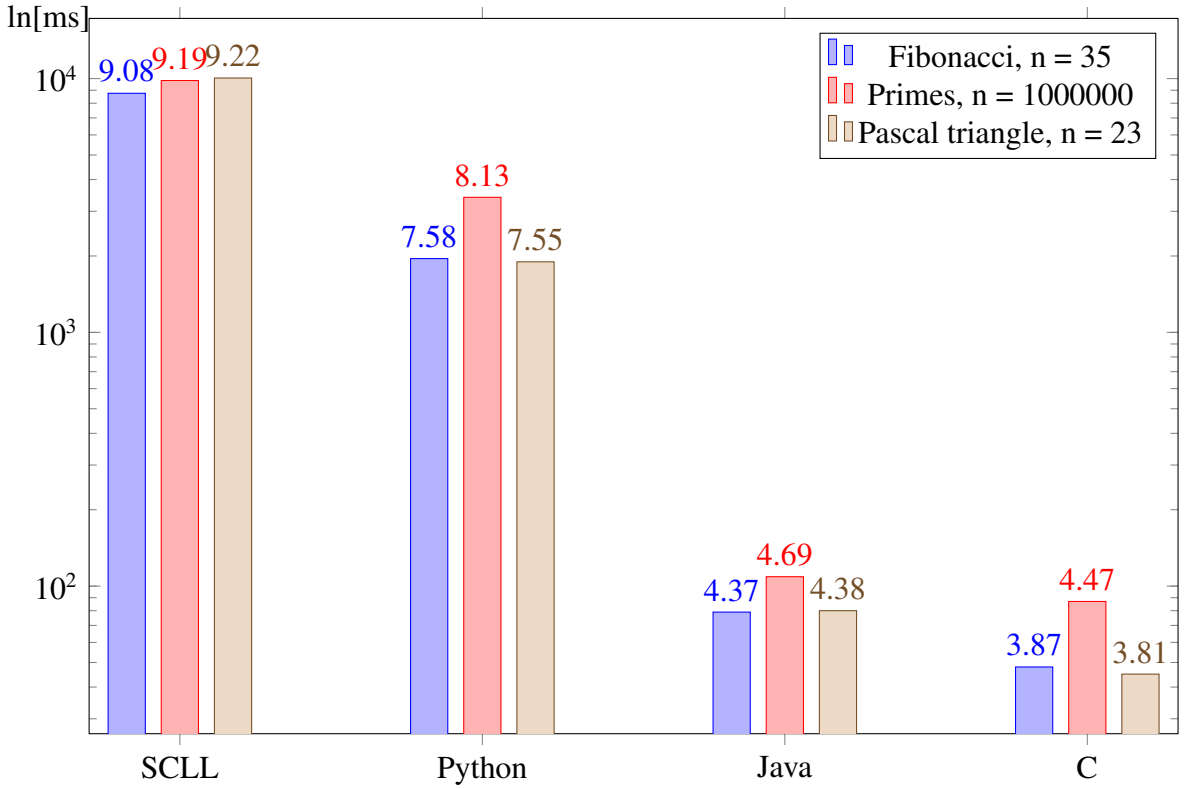


图 A.4　性能统计图表

附录 B

所用图片为图 B.1。

```
#!/usr/bin/python
from bcc import BPF

program = r"""
int hello(void *ctx) {
    bpf_trace_printk("Hello World!");
    return 0;
}
"""

b = BPF(text=program)
syscall = b.get_syscall_fnname("execve")
b.attach_kprobe(event=syscall, fn_name="hello")

b.trace_print()
```

(a) 测试代码一

```
(module $a.out.wasm
  (type (;0;) (func))
  (type (;1;) (func (param i32)))
  (type (;2;) (func (param i32) (result i32)))
  (type (;3;) (func (result i32)))
  (func $__wasm_call_ctors (type 0)
    nop)
  (func $_emscripten_stack_restore (type 1) (param i32)
    local.get 0
    global.set $__stack_pointer)
  (func $_emscripten_stack_alloc (type 2) (param i32) (result i32)
    (local i32)
    global.get $__stack_pointer
    local.get 0
    i32.sub
    i32.const -16
    i32.and
    local.tee 1
    global.set $__stack_pointer
    local.get 1)
  (func $emscripten_stack_get_current (type 3) (result i32)
    global.get $__stack_pointer)
  (table (;0;) 1 1 funcref)
  (memory (;0;) 258 258)
  (global $__stack_pointer (mut i32) (i32.const 66560))
  (export "memory" (memory 0))
  (export "__wasm_call_ctors" (func $__wasm_call_ctors))
  (export "__indirect_function_table" (table 0))
  (export "_emscripten_stack_restore" (func $_emscripten_stack_restore))
  (export "_emscripten_stack_alloc" (func $_emscripten_stack_alloc))
  (export "emscripten_stack_get_current" (func $emscripten_stack_get_current)))
```

(b) 测试代码二

图 B.1　所用代码集

所用表格为表 B.1 和表 B.2。

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	$\mathcal{L}l$	$\mathcal{Y}y$	$\mathbb{L}$	$\mathbb{Y}$	$\mathfrak{L}\mathfrak{l}$	$\mathfrak{Y}\mathfrak{y}$	$\mathscr{L}l$	$\mathscr{Y}y$
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 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.