基于国产架构的 UnixBench 测试基准分析与优化

吴登勇[1,2] 于英杰[1,2] 许 瑞[1,2] 金庆哲[1,2] 张 帅[1,2] 付文文[1,2]
WU Dengyong YU Yingjie XU Rui JIN Qingzhe ZHANG Shuai FU Wenwen

摘 要 计算平台性能直接反映一个企业在技术方面的综合实力与创新力，同时也是目标客户评估、选择硬件平台的重要依据，开展基准性能优化技术研究，提升产品性能、价值势在必行。基于此，文章深入剖析 UnixBench 测试基准的运行机理及性能度量算法，系统梳理影响系统性能的关键因素，并从编译器层面出发，针对不同架构平台开展性能优化策略研究。以国产 ARM 架构平台为依托，全面探究不同编译优化等级、静态编译及链接时优化对系统性能产生的影响。通过对比分析各类优化策略下的性能数据，明确优化参数与系统性能间的具体关联，并提出了针对性的性能调优建议，为平台性能评估与优化构建了系统性的认知框架。

关键词 国产 ARM 架构平台；性能测试；编译器优化；编译优化等级；静态编译；链接时优化

doi：10.3969/j.issn.1672-9528.2025.05.026

0 引言

在计算体系中，性能评估是衡量计算机系统运行效率的核心指标，对系统设计、优化迭代以及硬件选型决策具有至关重要的意义。在各类项目竞标与实物评测活动中，通过测试系统性能，可以快速定位产品性能短板，并据此评估厂商的技术实力[1]。UnixBench 作为一款综合基准测试应用程序，不仅广泛应用于系统性能评估与调优实践，还在操作系统性能对比分析及学术研究领域发挥着重要作用。虽然 UnixBench 提供了一种量化系统性能的方法，但测试结果仍受软件环境、系统负载等因素影响。为此，本研究聚焦于基于国产平台的 UnixBench 的性能优化，探讨了 UnixBench 测试结果与编译器之间的关系，为未来的性能优化研究提供理论基础和实践指导。

1 UnixBench 基准设计

1.1 测试项分析

UnixBench 测试分为 System 系统测试和 2D/3D Graphic 图形测试两大部分，其中 System 性能测试含 9 个测试类，12 个测试案例[2]。鉴于本研究的核心目标在于全面评估与优化系统的整体性能，因此将 System 系统测试作为研究的主要关注点，旨在通过该测试对系统的核心功能展开全面评估与优化。System 系统测试中各测试案例的具体详情如表 1 所示。通过分析测试项可以发现，UnixBench 测试结果与系统和编译器关系较大。

表 1 测试项介绍

测试项目	测试项说明
Dhrystone	测试字符串处理。因不涉及浮点操作，深受软硬件设计、编译和链接、代码优化、内存缓存、等待状态、整数数据类型的影响。
Whetstone	测试浮点运算速度与效率。这一测试的每个模块都包括一组用于科学计算的操作。
Execl Throughput	测试 execl 吞吐。此测试考察每秒钟可以执行的 execl 系统调用的次数。
File copy	测试文件数据从一个文件向另外一个文件传输的速率。即规定时间（默认 10 s）内被读、写、复制的字符数量。
Pipe Throughput	测试管道吞吐。考察 1 s 内一个进程可以向一个管道写 512 Byte 数据然后再读回的次数。
Pipe-based Context Switching	基于管道的上下文交互，测试衡量两个进程每秒钟通过一个管道交换和整数倍增加吞吐的次数。
Process Creation	进程创建测试。衡量一个进程每秒钟可以创建子进程然后收回子进程的次数（子进程一定立即退出）。关注点是新进程控制块的创建和内存分配，即测试程序直接使用了内存带宽。
System Call Overhead	系统调用消耗测试。衡量进入和离开操作系统内核的消耗，即一次系统调用的代价。消耗的指标是调用进入和离开内核的执行时间。
Shell Scripts	Shell 脚本测试。衡量 1 s 内一个进程可以启动并停止 Shell 脚本的次数，通常测试 1、2、3、4、8 个 Shell 脚本的共同副本。

1. 超越科技股份有限公司 山东济南 250104
2. 山东省自主可靠计算技术与装备重点实验室 山东济南 250104

1.2 测试运行流程

Run 脚本是 UnixBench 运行的入口脚本，也是该工具的核心文件之一，管理、串联了整个测试工具。Run 脚本的执行从 main 函数开始，通过解析命令行参数，循环调用 runTests 函数以执行测试流程。在 runTests 函数内部，调用 runBenchmark 函数来执行 System 套件的各测试项，并通过 indexResults 函数计算总体性能得分。runBenchmark 函数确定执行命令以及测试项的重复次数 repeats 后，遍历执行 repeats 次 runOnePass 函数，最终由 combinePassResults 函数计算单项分数。Run 脚本执行的调用关系如图 1 所示。

图 1 RUN 脚本执行的调用关系

1.3 度量方法

UnixBench 的 System Benchmarks Index Score 通过对各个子测试项的单项性能分数进行加权平均计算得出。每个子测试项的单项性能分数又通过对原始测试结果进行指数运算得到。

1.3.1 单测试项计分算法

单项测试结果处理通过 Run 中的 combinePassResults 函数实现，单个测试项测试次数通过变量 testParams 中的 repeat 参数可调。根据设定的单项测试次数将每一遍的结果进行排序，去掉最差的 1/3 的结果，每一项原始结果形如：COUNT|x|y|z，其中 x 为分数，y 为时间单位，若 y 为 0，则 x 为比率，z 为标签符号。

当 y 为时间单位时的计算公式为：

$$\text{score} = e^{\left(\sum_{i=1}^{n} \log(\text{count})\right)/n} \tag{1}$$

当 y 为 0 时的计算公式为：

$$\text{score} = e^{\left(\sum_{i=1}^{n} \log\left(\frac{\text{count}\cdot\text{timebase}}{\text{time}}\right)\right)/n} \tag{2}$$

式中：score 表示单项性能分数；n 表示剩余有效结果个数；count 表示每个有效结果的值；timebase 表示时间基本单位；time 为运行总时间。

1.3.2 总分计分算法

总分计分算法通过 Run 中的 indexResults 函数实现，公式表示为：

$$\text{sum\_score} = e^{\left(\sum_{i=1}^{\text{tests\_num}} \log\left(\frac{\text{score}}{\text{baseline}}\right)\right)/\text{tests\_num}} \times 10 \tag{3}$$

式中：score 表示单项性能分数；tests_num 表示 System 系统测试的子项个数。此外，baseline 表示文件 pgms/index.base 中所记录的各项负载的基准性能数据，这些数据均基于 George SPARCstation 20-61 工作站测试得到。该工作站的详细配置信息如表 2 所示。

表 2 System 系统测试基准平台配置

项目	详细信息
型号	Sun Microsystems SPARCstation 20-61
CPU 型号	Superscalar SPARC+ 60 MHz
CPU 数量	1
内存	128 MB RAM
操作系统	Solaris 2.3
UnixBench index 值	10

2 基于 UnixBench 的性能调优测试分析

2.1 测试平台环境

UnixBench 工具在不同 CPU 架构及操作系统的部署安装通用性较强，适配 x86_64、aarch64、sw_64 和 loon-

garch64 等指令集架构的处理器平台，可运行在 KylinOS、UOS、openEuler 和 Ubuntu 等基于 Linux 的操作系统，兼容 GCC[3]、Clang[4]、AOCC[5]、ICC[6] 和 BiSheng 等编译环境。本次基于 UnixBench 的性能调优研究选择主流的国产 ARM 架构平台进行测试验证，参考机器具体配置信息如表 3 所示。

表 3 测试平台环境配置信息

平台信息	ARM 架构平台 1	ARM 架构平台 2
CPU 信息	某国产 8 核处理器 1	某国产 8 核处理器 2
内存信息	DDR5 8 GB×2	DDR4 8 GB×2
硬盘信息	M.2 512 GB SSD	M.2 512 GB SSD
操作系统	麒麟 V10（2403）	麒麟 V10（2403）
编译器	GCC 9.3.0	GCC 9.3.0

2.2 测试优化与分析

2.2.1 不同编译优化等级测试结果分析

GCC 提供了多种优化等级，通过参数设置来控制代码的优化程度，影响程序的性能和执行效率 [7-8]。在 UnixBench 的测试中，优化选项的选择可能会影响到整数运算、浮点运算、系统调用等子测试的结果。因此，深入理解和精确配置编译器参数是实现性能优化的重要环节。国产 ARM 架构各平台使用不同优化等级的总分性能数据如表 4 所示，各测试子项性能表现如图 2~5 所示。

表 4 国产 ARM 架构不同平台使用不同编译优化等级的性能数据表

平台	ARM 架构平台 1			ARM 架构平台 2		
编译参数	-O2	-O3	-Ofast	-O2	-O3	-Ofast
单核总分	2 323.2	2 338.2	2 343.9	1 907.2	1 928.1	1 923.9
多核总分	7 220.1	7 291.5	7 233.0	7 412.0	7 489.3	7 507.7

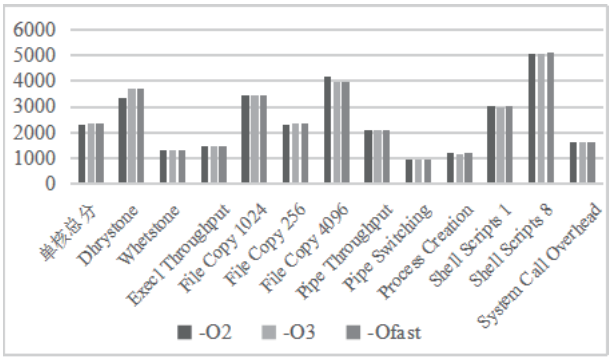

图 2 国产 ARM 架构平台 1 不同编译优化等级的单核数据

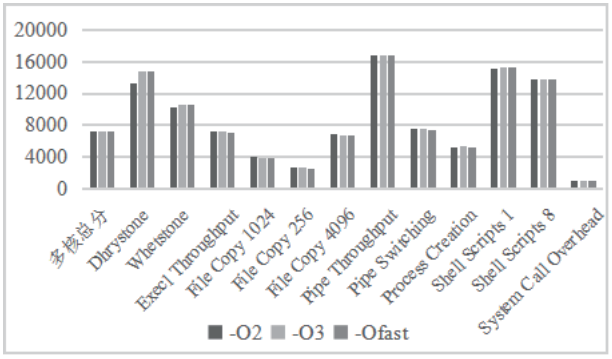

图 3 国产 ARM 架构 1 平台不同编译优化等级的多核数据

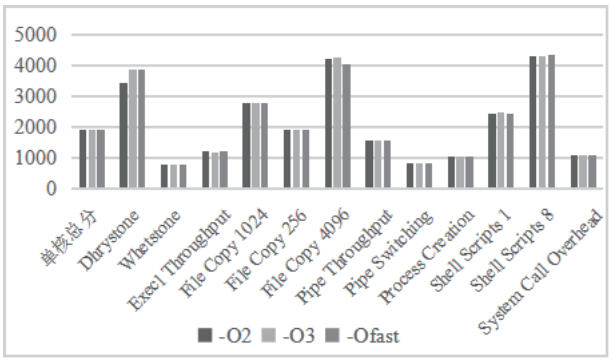

图 4 国产 ARM 架构平台 2 不同编译优化等级的单核数据

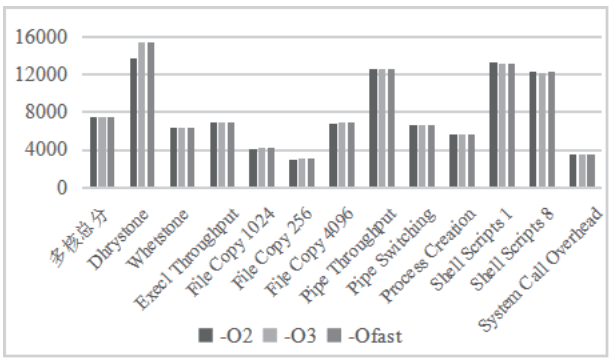

图 5 国产 ARM 架构平台 2 不同编译优化等级的多核数据

如表 4 及图 2~3 所示，国产 ARM 架构平台 1 在整型计算测试项中启用 -O3 优化等级较 -O2 等级单核性能提升 12%，多核性能提升 11%。在浮点计算测试项中启用 -O3 优化等级较 -O2 等级单核性能提升 3%，多核性能提升 2%。启用 -Ofast 优化等级较 -O3 基本无异。如表 4 及图 4~5 所示，国产 ARM 架构平台 2 在整型计算测试项中启用 -O3 优化等级较 -O2 等级单核性能提升 12%，多核性能提升 12%。在浮点计算测试项中启用 -O3 优化等级较 -O2 等级单核性能提升 2%，多核性能提升 2%。启用 -Ofast 优化等级较 -O3 整型计算性能稍有下降。

综合来看，编译优化等级对 UnixBench 测试工具的 Dhrystone（整型计算）、Whetstone（浮点计算）两个测试子项有优化效果，但并非优化等级越高越好，测试时启用 -O3 编译优化等级最优。

2.2.2 静态编译测试结果分析

编译器使用 -static 标志进行编译是指链接静态库而不是动态库。当使用静态链接时，所有的依赖库都被合并到最终的可执行文件中[9]。其优势包括减少加载时间，避免依赖冲突，增加优化机会等，可以提升 UnixBench 相关测试子项性能。添加静态编译参数 -static 后各平台性能数据如表 5 所示，各测试子项的性能表现如图 6~9 所示。

表 5 国产 ARM 架构不同平台启用静态编译的性能数据表

平台	ARM 架构平台 1		ARM 架构平台 2	
编译参数	默认	-static	默认	-static
单核总分	2 323.2	2 532.6	1 907.2	2 098.6
多核总分	7 220.1	7 769.3	7 412.0	8 041.3

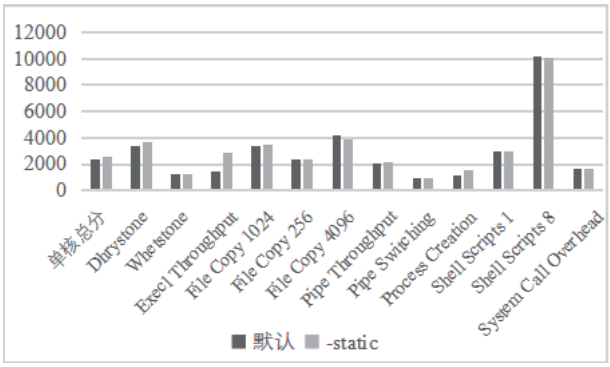

图 6 国产 ARM 架构平台 1 静态编译的单核测试数据

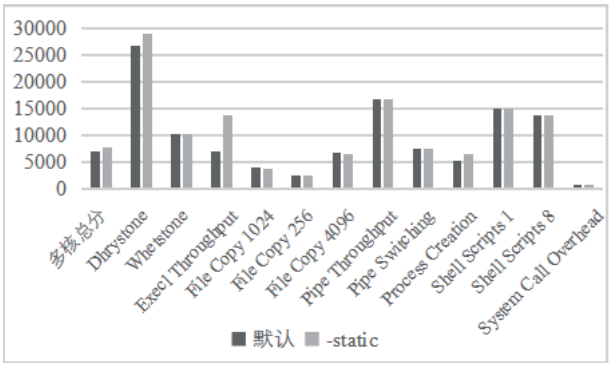

图 7 国产 ARM 架构平台 1 静态编译的多核测试数据

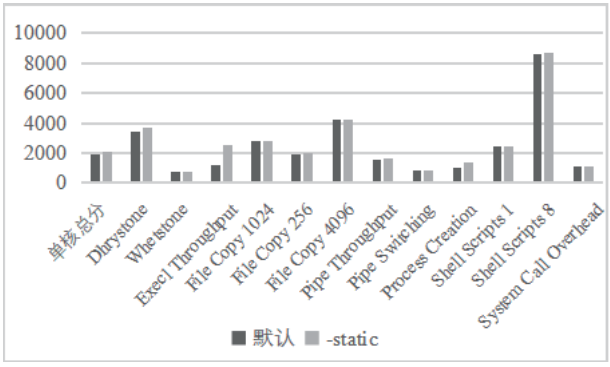

图 8 国产 ARM 架构平台 2 静态编译的单核测试数据

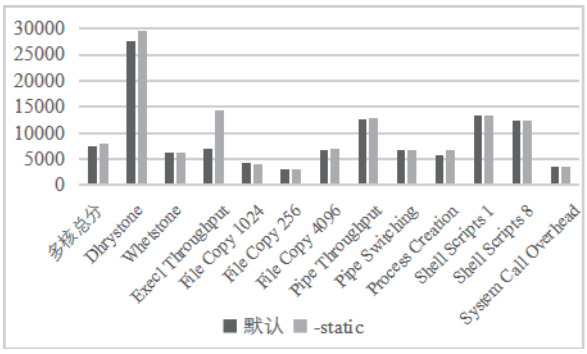

图 9 国产 ARM 架构平台 2 静态编译的多核测试数据

如表 5 及图 5~9 所示，启用 -static 静态编译标志后，Execl 吞吐量测试项方面，国产 ARM 架构平台 1 单核性能提升 96%，多核性能提升 95%；国产 ARM 架构平台 2 单核性能提升 106%，多核性能提升 107%；进程创建测试项方面，国产 ARM 架构平台 1 单核性能提升 34%，多核性能提升 27%；国产 ARM 架构平台 2 单核性能提升 32%，多核性能提升 19%。

综上，在 ARM 架构平台上启用 -static 静态编译标志后，Execl 吞吐量测试项与进程创建测试项性能均有较大提升，但各平台提升比例不同，国产 ARM 架构平台 2 性能提升较高。因此，测试时可启用 -static 静态编译提升平台性能。

2.2.3 链接时优化编译测试结果分析

LTO 是一种在链接阶段对程序进行优化的技术[10]。在编译过程中，多个中间文件通过链接器合并为一个程序，LTO 能够对中间文件进行整体分析和跨模块的优化。这种优化方式能够缩减代码体积，提高程序的执行效率以及提升编译器优化效果，提升整型、浮点计算及 Execl 吞吐量测试子项性能。添加编译器链接时优化参数 -flto 后各平台性能数据如表 6 所示，各测试子项性能表现如图 10~13。

表 6 国产 ARM 架构不同平台启用链接时优化性能数据表

平台	ARM 架构平台 1		ARM 架构平台 2	
编译参数	默认	-flto	默认	-flto
单核总分	2 323.2	2 461.8	1 907.2	2 069.6
多核总分	7 220.1	7 658.9	7 412.0	7 954.3

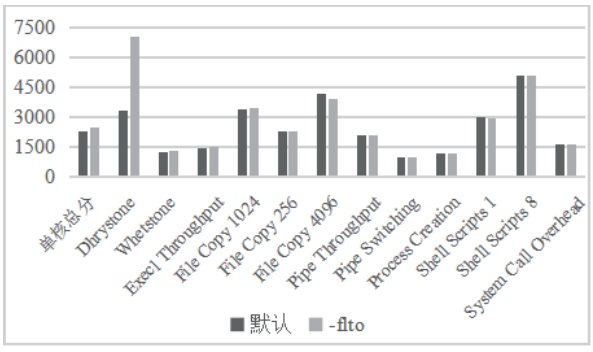

图 10 国产 ARM 架构平台 1 链接时优化编译的单核数据

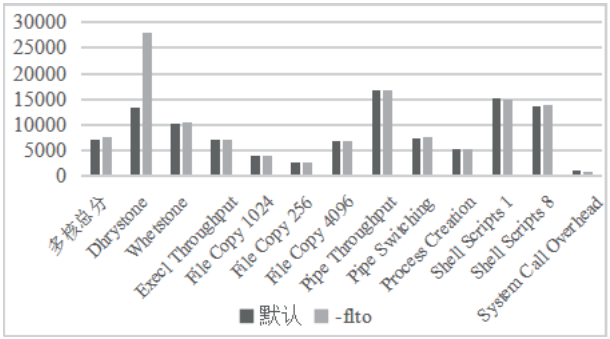

图 11 国产 ARM 架构平台 1 链接时优化编译的多核数据

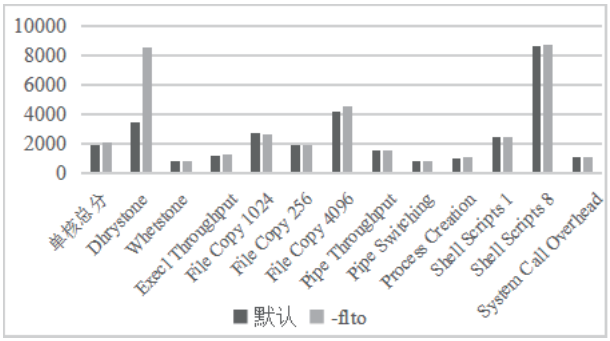

图 12 国产 ARM 架构平台 2 链接时优化编译的单核数据

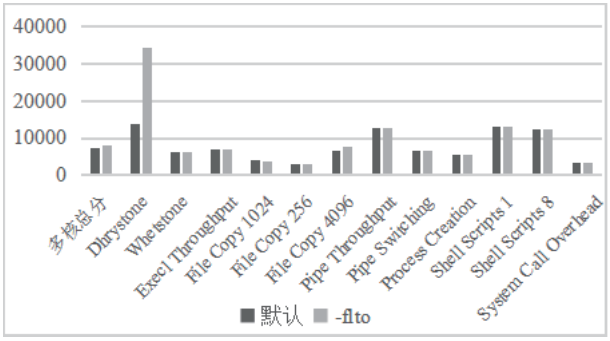

图 13 国产 ARM 架构平台 2 链接时优化编译的多核数据

如图 10~13 所示，启用 -flto 链接时优化标志后，整型计算性能提升最高，国产 ARM 架构平台 1 性能提升 110%，国产 ARM 架构平台 2 性能提升 149%。在两个国产 ARM 架构平台上性能提升表现一致，各平台提升比例不同，国产 ARM 架构平台 2 性能提升较高，与处理器微内核架构正相关。因此，测试时可启用 -flto 链接时优化提升平台性能。

2.2.4 开启编译参数优化测试结果分析

基于上述各平台在独立启用相关编译优化选项时的性能表现，本节研究在国产 ARM 架构平台上应用上述所有优化参数，旨在验证组合编译参数对各架构性能提升的具体效果。详细的性能数据如表 7 所示。增加上述所有优化参数后，各架构平台的单核及多核性能均有明显提升。开启编译器优化参数后，UnixBench 各测试子项性能提升明显，进而带动总分提升，其中国产 ARM 架构平台 1 单核性能提升 16%，多核性能提升 15%；国产 ARM 架构平台 2 单核性能提升 19%，多核性能提升 17%。

表 7 不同平台组合编译参数优化性能数据表

平台	ARM 架构平台 1		ARM 架构平台 2	
参数	默认	编译参数优化	默认	编译参数优化
单核总分	2 323.2	2 695.9	1 907.2	2 264.7
多核总分	7 220.1	8 267.1	7 412.0	8 638.7

3 结语

本研究全面探讨了 UnixBench 测试工具的各个方面，包括其测试负载设计、测试运行流程及计分算法。通过对 UnixBench 各个组成部分的深入分析，明晰了其在评估系统性能过程中的核心影响因素。基于此，从编译器层面进行了系统化的优化与验证工作。研究发现，通过选择合适的编译器优化选项可显著提升 UnixBench 测试性能。此外，还提出了一系列具有针对性的优化策略， 旨在帮助开发者和系统管理员更好地理解和利用 UnixBench 来评估和优化系统的性能表现。

参考文献：

[1] BAID D, MANDAL T, JOGLEKAR N, et al. Benchmarking techniques for performance analysis of operating systems and programs[J].International research journal of engineering and technology, 2023, 10(7): 1205-1210.

[2] KELLY L. UnixBench[EB/OL]. [2025-05-12]. https://github.com/kdlucas/byte-unixbench.

[3] GCC, the GNU compiler collection[EB/OL]. (2025-04-25) [2025-05-12]. http://gcc.gnu.org/.

[4] 龚丹，苏小红，王甜甜 .Clang 编译平台优势分析 [J]. 智能计算机与应用，2017,7(3): 188-190.

[5] AMD Optimizing C/C++ Compiler[EB/OL]. (2024-10-10) [2025-05-12]. https://developer.amd.com/amd-aocc/.

[6] INTEL C++ Compiler[EB/OL]. [2025-05-12]. https://software.intel.com/ru-ru/c-compilers.

[7] 严博 . Odin Ⅱ系统编译及综合模块的设计与实现 [D]. 南京：东南大学，2017.

[8] 赖庆宽 . 基于峰值架构的跨平台编译器分析优化技术研究 [D]. 南充：西华师范大学，2020.

[9] MELNIK D, KURMANGALEEV S, AVETISYAN A, et al.Optimizing programs for given hardware architectures with static compilation: methods and tools[J]. Proceedings of the institute for system programming of the RAS, 2014, 26(1): 343-356.

基于改进 SVM 的电子通信信道恶意干扰信号辨识

龚 岩 [1]
GONG Yan

摘 要 在复杂的通信环境中，信号会经过多条不同路径传播后叠加，导致信号的幅度、相位和延迟发生复杂变化，这些变化使提取信号特征与实际情况偏差较大，进而影响后续对恶意干扰信号的辨识。为此，提出基于改进 SVM 的电子通信信道恶意干扰信号辨识方法。通过设定非线性元件输出表征和信道变量的正反向变换公式，结合信道数量构建电子通信信道模型。基于该模型采集信号后，利用主成分分析提取关键特征，采用小波域滤波技术根据分解层级和模极大值序列判定有效信息信号，有针对性地去除信号中的噪声。运用改进的 SVM 理论在特定空间定义超平面作为决策边界，实现信号分类。结合自适应滤波理论、小波变换和变步长 LMS 算法，快速适应恶意干扰信号的动态变化，进行辨识处理。实验结果表明，该方法在信号去噪和恶意干扰信号辨识上均表现优异，识别精度高。

关键词 改进 SVM；电子通信信道；恶意干扰信号；信号噪声；信号特征

doi：10.3969/j.issn.1672-9528.2025.05.027

0 引言

在复杂的通信环境中，恶意干扰信号的存在严重威胁着通信系统的正常运行，导致信息传输错误甚至通信中断。针对干扰信号的辨识问题，已有多种方法被提出并应用于实际场景中。张盛楠等人 [1] 结合傅里叶基函数分析、断续偏离量化处理及变换域与变步长 LMS 算法，实现激光通信干扰信号的自适应辨识与消噪。该方法由于未准确考虑非线性元件和复杂信道变量，自适应调整时不能准确跟踪干扰信号变化，辨识效果不佳。刘佳楠等人 [2] 提出基于短时傅里叶变换和幻影卷积网络的复合干扰识别算法，有效识别无线通信中的复合干扰信号种类和干噪比，但在非线性信道中，短时傅里叶变换难以精确解析复杂的相位和幅度变化模式，信号特征提取不完整，影响对复合干扰信号的准确识别。李刚等人 [3] 运用 EMD 分解稳态信号，小波变换去噪，结合多模特征融合模型与卷积、全连接层及分类器，实现干扰信号的有效识别与概率分类。但实际信道的非线性失真使信号频率成分和幅值变化复杂，EMD 分解可能无法准确得到反映信号内在特征的固有模态函数，影响后续干扰信号识别。为此，本文提出基于改进 SVM 的电子通信信道恶意干扰信号辨识方法，旨在克服现有方法不足，提高识别精度和适应性。

1 基于改进 SVM 的电子通信信道恶意干扰信号辨识

1.1 构建电子通信信道模型

在电子通信信道恶意干扰信号辨识过程中，电子通信环境复杂多变，存在多种影响因素。为了将这些复杂的因素进行整合和量化表示，本研究构建了电子通信信道模型。

1. 江西制造职业技术学院信息工程学院 江西南昌 330095

[10] GLEK T, HUBICKA J. Optimizing real world applications with GCC link time optimization[EB/OL]. (2010-11-03) [2024-06-25].https://doi.org/10.48550/arXiv.1010.2196.

【作者简介】

吴登勇（1983—），男，山东济南人，本科，高级工程师，研究方向：计算机体系结构。

于英杰（1992—），男，山东烟台人，本科，中级工程师，研究方向：计算机体系结构。

许瑞（1983—），女，河北衡水人，硕士，高级工程师，研究方向：计算机体系结构。

金庆哲（1994—），男，山东德州人，本科，助理工程师，研究方向：计算机体系结构。

张帅（1996—），男，山东聊城人，硕士，研究方向：计算机体系结构。

付文文（1998—），女，山东聊城人，硕士，研究方向：计算机体系结构。

（收稿日期：2025-01-02 修回日期：2025-05-12）