基于 Linux 系统参数在 X86 与 ARM 架构下 Nginx 性能调优的比较研究

陈文雄[1,2]，李乐乐[1]，喻之斌[1,2]

[1]（中国科学院深圳先进技术研究院 深圳 518055）

[2]（中国科学院大学 北京 100049）

摘要：在当今的数字时代，Nginx 已成为 Linux 系统上最为普及的 Web 应用服务器，占据了市场份额第一名。鉴于其在确保用户服务质量方面的关键作用，对 Nginx 性能的优化显得尤为重要。尽管 Nginx 服务器广泛部署于 X86 和 ARM 这两种主要的硬件架构之上，迄今为止，针对这两种架构下 Nginx 性能调优的对比分析尚处于空白。本研究旨在填补这一缺口，通过对比这两种架构的系统参数自动调优效果，揭示了显著的差异性结果：在处理动态请求的场景下，X86 架构的性能明显胜过 ARM 架构，其 P99 延迟比 ARM 低达 515 毫秒，性能提升高达 287%。反之，在处理静态请求时，ARM 架构则展现出更加卓越的表现，其 P99 延迟比 X86 低 220 毫秒，性能提升达到了 60%。这一发现突出了 X86 和 ARM 架构在不同类型的负载处理上的特定优势，并明确指出了不同硬件架构对于 Nginx 性能优化策略的显著影响。因此，系统管理员在针对不同硬件架构进行 Nginx 优化时，必须考量架构特有的静态与动态请求之间的性能差异和迭代效率，以确保最佳性能表现。

关键词：Nginx；X86；ARM；Linux 参数；性能调优

中图分类号：TP39 文献标志码 A doi: 10.12146/j.issn.2095-3135.20240307002

Comparative Analysis of Nginx Performance Tuning Based on Linux System Parameters on X86 versus ARM Architectures

Wenxiong Chen[1,2],Lele Li[1], Zhibin Yu[1,2]

[1] (Shenzhen Institute of Advanced Technology, Chinese Academy of Sciences, Shenzhen 518055, China)

[2](University of Chinese Academy of Sciences, Beijing 100049, China)

Corresponding Author：Zhibin Yu E-mail：zb.yu@siat.ac.cn

Abstract: In today's digital age, Nginx has emerged as the most prevalent web application server on Linux systems, securing the top position in market share. Given its critical role in ensuring the quality of service for users, optimizing the performance of Nginx servers is important. Despite the widespread deployment of Nginx servers across the two main hardware architectures, X86 and ARM, a comparative analysis of performance tuning on these architectures remains unexplored. This study aims to bridge this gap by employing automatic system parameter tuning on Nginx across these architectures, revealing the significant difference. When handling dynamic requests, the optimized performance of Nginx on X86 architecture significantly outperforms that of the ARM architecture. As a result, the optimized performance of Nginx on X86 architecture achieves a P99 latency of 515 milliseconds, which is performance improvement of 287% than that of the ARM architecture. Conversely, when processing static requests, the ARM architecture demonstrates superior performance, with a P99 latency of 220 milliseconds, resulting in a performance increase of 60% than that of

来稿日期：2024-03-07 修回日期：2024-04-11

基金项目：深圳市科技计划项目（JCYJ20220818101607015）

作者简介：陈文雄，硕士研究生，研究方向为 Nginx Web 服务器的性能优化等；李乐乐，科研助理，研究方向为云计算资源优化、大数据性能加速等。喻之斌，博士，研究员博士研究生导师，研究方向为多核体系结构，通用 GPU 体系结构，大数据体系结构，云计算等；

E-mail：zb.yu@siat.ac.cn。

X86 architecture. These findings highlight the distinct advantages of X86 and ARM architectures in handling different types of loads. It shows the significant impact of hardware architecture on optimizing Nginx's performance. Therefore, to optimize the performance of Nginx web server, system administrators must consider the performance differences between static and dynamic requests of Nginx and the unique iterative efficiency over different hardware architectures.
Key words: Nginx; X86; ARM; Linux Parameters; Performance Tuning
Funding:: This project is supported by the Shenzhen Science and Technology Program (JCYJ20220818101607015)

1 引 言

随着互联网技术的不断进步和应用的日益复杂化，对 Web 服务器的性能要求也相应提升。Nginx[1]，作为一款在全球范围内被广泛采用的高性能 Web 服务器，因其优异的并发处理能力和资源利用效率而受到业界的广泛认可，根据 W3Tech 排名[2]，该网站在世界前 100,000 个网站中占 34.1%，稳居第一。这种服务器软件能够有效地支持高流量的 Web 服务，确保在用户访问高峰期间网站的快速响应和稳定运行[3]。Nginx 的性能不仅取决于其自身的优化和配置，还受到运行它的硬件平台特性的影响[4][5]。当前，X86 和 ARM 架构是市场上最主流的两种服务器硬件架构，它们在设计理念、能效比以及适用场景上存在显著差异。X86 架构以其强大的计算能力和广泛的应用支持在服务器市场上占据了主导地位[6]，而 ARM 架构则以其高效的能源利用率和较低的成本在某些领域逐渐获得优势[7]。随着 ARM 架构服务器性能的不断提升，它们开始被越来越多地用于运行网络服务和应用，与传统的 X86 服务器形成了竞争。

尽管 Nginx 已在 X86 和 ARM 两种硬件架构上广泛部署，但对其在这些架构上的性能表现进行系统性比较的研究还相对缺乏。特别是，随着 ARM 架构在高性能服务器市场上的崛起，这一研究空白显得更为重要。近年来，ARM 架构已开始应用于 Nginx 服务器中，并展现出了不错的效果。特别是在亚马逊基于 ARM 架构的 Graviton2 处理器支持的 EC2 AMI 上运行 Nginx，比 X86 实例具有 40%更高性价比和 20%较低成本[8]。这一发现不仅证实了 ARM 架构服务器在运行 Nginx 方面的潜力，也突显了深入了解 Nginx 在 X86 与 ARM 架构上性能差异的重要性，这对于系统管理员选择最合适的硬件平台和优化服务器配置具有重要的实际意义。

此外，Linux 系统参数的调整是提升服务器性能的一个关键方面。这些参数影响着操作系统级别的资源分配和任务调度，从而间接影响着运行在其上的应用程序，包括 Nginx 的性能。然而，关于这些参数调整在不同硬件架构下对 Nginx 性能的具体影响，现有研究成果相对有限，尤其缺乏系统化的分析和实验验证。

因此，本研究致力于通过系统的实验评估和分析，填补这一研究空白。本研究深入探讨 Linux 系统参数调优对 X86 与 ARM 架构服务器上 Nginx 处理静态与动态请求性能的影响，旨在全面评估和比较这两种架构下 Nginx 的性能表现，并基于实验结果识别最有效的系统参数调整策略。通过这项研究，本研究期望为系统管理员提供科学的配置建议和实践指导，帮助他们在多样化的硬件环境中优化 Nginx 的性能，从而更好地满足用户需求，推动 Web 服务技术的发展。

现有文献回顾表明，尽管 Web 服务器性能优化的研究逐渐增多，但大部分研究侧重于软件层面的优化，如负载均衡算法改进[9]、缓存机制改进[10]等，对于硬件架构差异及操作系统参数调优的探讨相对较少。特别是 ARM 架构作为高性能计算的新选择，其在性能优化方面的研究不仅具有重要的现实意义，也有着广泛的应用价值。本研究通过专注于 Linux 系统参数对 Nginx 性能的影响进行深入分析，旨在为该领域提供全新的见解和理论支持。

本研究的创新之处在于，本研究不仅系统比较了 X86 与 ARM 架构服务器上 Nginx 的性能差异，而且通过一系列细致的实验设计和方法论，揭示了不同 Linux 系统参数调整对这两种架构性能影响的具体机制。这为系统管理员在面对多变的硬件选项和配置挑战时，提供了基于实验数据的直接指导。此外，本研究的发现还将为未来在不同硬件平台上部署和优化 Web 服务器的研究提供宝贵的参考和启示。

2 研究现状

2.1 X86 架构与 ARM 架构的性能对比

在计算技术领域的现代发展轨迹中，X86 与 ARM 架构的性能比较已成为研究的焦点，这两种架构代表了计算硬件的不同设计哲学和应用领域。X86 架构，作为高性能计算的传统代表，广泛应用于服务器和桌面计算环境中，强调复杂指令集计算机（complex instruction set computer，CISC）架构以提高任务处理的灵活性和复杂度[6]。相对地，ARM 架构采用精简指令集计算机（reduced instruction set computer，RISC）架构原理，优势为在移动设备和边缘计算领域的功耗较低[7]。随着技术的演进和市场需求的变化，ARM 架构已逐步扩展到服务器市场，引发对其性能、能效及应用适宜性的广泛讨论。

尽管对这两种架构的比较已经引起了广泛的关注，但目前在性能优化方面，特别是在高性能 Web 服务器应用程序，如 Nginx 的运行环境下，关于它们性能差异的研究仍相对有限。近期的研究成果揭示了在特定场景下这两种架构的性能和能耗差异。例如，da Silva 等人[11]对比了 ARM Cortex AS7 与英特尔 X86 平台（代表为 Nvidia Tegra X1 和 Pentium Braswell N3700）在运行 Apache 和 Nginx 服务器时的性能，发现 ARM 处理器在某些条件下性能略优，但英特尔 CPU 在能耗方面表现更佳。此外，Kalyanasundaram 等人[12]的研究进一步证明了在大数据处理场景下，ARM 架构服务器能够在保持与 X86 架构服务器相近性能的同时显著降低运营成本。

此外，Chen Xinghan 等人[13]的研究通过在 AWS Lambda 上使用 ARM64 处理器托管无服务器功能的实验，发现尽管只有部分函数在 ARM64 上运行得更快，但绝大多数函数的托管成本更低。Jiang 等人[14]的实证研究评估了 Amazon Graviton ARM 架构处理器在处理大规模计算密集型工作负载的能力，并将其与 X86 架构处理器进行了比较。尽管存在一些限制，例如 L3 缓存和内存访问速度，但 ARM 架构处理器在处理包括 Web 服务、视频转码和 TB 级排序在内的各种任务时，能够实现与 X86 架构处理器相当甚至更高的性价比。

这些研究为理解 X86 与 ARM 架构的性能差异提供了宝贵的见解，但它们通常集中在特定的测试条件或应用场景上，缺乏在 X86 和 ARM 架构下对 Web 服务器性能优化的全面比较。此外，这些研究往往未能深入探讨不同架构的技术细节，如缓存架构、内存管理策略等因素，这些因素如何影响 Web 服务的性能和响应能力，缺乏不同架构下 Nginx 参数调优的研究。

2.2 Nginx 性能调优的研究现状

对于 Nginx 性能调优的研究，已经表明系统资源如 CPU、内存和网络带宽在处理高并发请求时对 Nginx 服务器性能的显著影响[15][16]。有效地管理和优化这些资源是关键，以提高 Nginx 服务的响应能力和处理效率。

Wang J.等人[15]通过对 Nginx 架构和工作流程的深度分析，识别了 Nginx 在初次安装时的性能短板，并通过手动调整 Linux 内核参数成功提升了 Web 服务器的性能。然而，这种方法存在的挑战在于有限的参数调整范围和手动配置过程的复杂性。Wang Runzhe 等人

[17]采用增强的自动调优工具在商业和工业服务应用中实施，实现了 Nginx 性能的 2%至14%提升。尽管这种方法有效，但其参数选择依赖于领域知识和经验，缺乏一种系统化的框架来确定对 Nginx 性能有显著影响的参数组合。Chen Wenxiong 等人[18]提出的 INTOP 自动调优策略代表了该领域的一个重要进步，通过系统化识别和调整影响 Nginx 性能的关键参数组合，提供了一种高效且系统化的性能优化策略。这种方法减少了对专业知识的依赖，并通过自动化过程大幅提高了优化的效率和效果。

综上所述，本研究旨在通过深入分析和系统化实验评估，填补上述研究空白。本研究专注于评估 Linux 系统参数调整在 X86 与 ARM 架构服务器上运行 Nginx 时的性能影响，旨在全面比较这两种架构在性能优化方面的差异。本研究希望通过这项研究为系统管理员提供明确的指导，帮助他们在多样化的硬件环境中优化 Nginx 的性能。通过这种方式，本研究期望能够更好地满足用户需求，并为计算行业的可持续发展提供支持。

3 方法

为了构建一个详尽且逻辑清晰的实验方法部分，本研究将深入探讨实验环境设置、测试方法和自动调优方法，确保研究的科学性和实用性得到充分体现。

3.1 实验环境设置

在本研究中，搭建了两套实验集群，每套由 5 台服务器组成。X86 实验集群配备了 Intel(R) Xeon(R) Silver 4214 CPU，每台服务器具有 48 个逻辑核和 125GB DDR4 内存，总计 120 个 CPU 内核和 625GB 内存。而 ARM 实验集群在 Nginx 负载均衡服务器上采用 HiSilicon Kunpeng 920-4826 CPU，拥有 48 个物理核和 124GB DDR4 内存，其余服务器配置与 X86 实验集群相同。所有服务器运行在 Ubuntu 18.04.6 LTS 操作系统上，Nginx 版本统一为 1.20.1。通过这一详细的实验环境设置，本研究计划系统评估和分析在调整 Linux 系统参数后，Nginx 在处理请求时在这两种硬件平台上的性能差异。

3.2 测试方法

在本研究中，本研究使用 Lua 脚本语言模拟 Nginx 服务器上的静态和动态请求负载[19]，分别称为静态负载测试(static load test，SLT)和动态负载测试(dynamic load test，DLT)。（1）SLT 脚本从预定义的静态文件集中任意选择路径，然后发起 HTTP GET 请求到这些路径。这种模拟方法有效地模仿了用户在请求网页上的静态资源时的行为，包括 HTML、CSS、JavaScript 文件、图片和其他资源。（2）DLT 脚本为每次迭代生成唯一的随机数作为请求参数，这形成了引发 HTTP GET 请求的动态 URL 请求。这种技术准确地模拟了用户从网页获取动态内容，包括但不限于通过应用程序编程接口检索的数据。SLT 和 DLT 过程都通过 wrk2[20]来实施，这是一个开源的 HTTP 压力测试工具。在实验中，配置 wrk2 以每秒 10,000 个请求的速率在 Nginx 负载均衡器上执行负载测试。这些测试的结果，特别是 99%的延迟(99th Percentile，P99)，用作评估 Nginx 在 SLT 和 DLT 下性能的实用指标，能直观地反映用户的实际体验，尽管 CPU 和内存使用率也是重要的参考指标。这种测试方法不仅确保了数据的准确性，还能有效模拟真实世界中的 Web 服务请求场景。

3.3 自动调优方法

在本研究中，本研究采用了 INTOP 方法[18]，通过系统性地调整和评估 Linux 系统参数，目的在于优化 Nginx 在 X86 和 ARM 架构服务器上的性能。本研究的方法涵盖了从参数筛选到全局搜索的五个细致步骤，旨在全面提升 Nginx 服务器的性能表现。

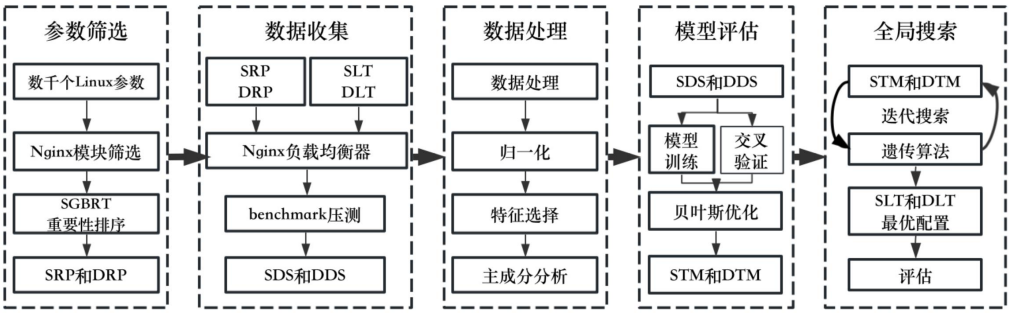

图 1 自动调优方法框图

Fig. 1 Block diagram of the auto-tuning method

3.3.1 参数筛选

在本研究的首个阶段，本研究细致地研究了 Nginx 处理请求的内部机制，并将这些信息与系统参数进行了关联。通过深入分析 Nginx 的核心模块——事件模块、HTTP 模块、邮件模块及其关键的 HTTP 子模块（例如 Gzip 和 Cache 子模块）[1]，本研究初步识别出 100 个关键系统参数。特别地，本研究关注了这些模块如何与系统资源（如内存、网络配置）交互[16][21]，并基于其对 Nginx 处理静态和动态请求的性能影响进行筛选。为了突出那些对 Nginx 性能有显著影响的参数，本研究采用了随机梯度提升回归树（stochastic gradient boosting regression trees，SGBRT）[22]进行特征重要性排序。SGBRT 因其在处理复杂数据集时展现出的强大特征选择和评估能力、高效性和鲁棒性，被广泛用于这一目的。最终，本研究筛选并确定了最具影响力的静态请求参数（static request parameters，SRP）和动态请求参数（dynamic request parameters，DRP）。

3.3.2 数据收集

数据收集阶段旨在系统地记录 Nginx 在不同 SRP 和 DRP 参数配置下的性能指标。通过使用 wrk2 工具执行 SLT 和 DLT 负载测试，本研究获得了大量性能数据。这些数据包括传统的性能指标 P99 延迟值，为后续的建模阶段提供了丰富的数据集，分别命名为静态数据集（static data set，SDS）和动态数据集（dynamic data set，DDS）。

3.3.3 数据处理

在数据处理阶段，本研究采用了多种数据预处理技术，包括归一化处理、特征选择、以及主成分分析（principal component analysis，PCA）[23]。这些步骤旨在优化性能预测模型的准确性和效率。归一化处理保证了不同性能指标在模型中的均衡贡献，特征选择帮助去除了无关或冗余的特征，降低模型复杂度。PCA 通过降维处理，从海量性能指标中提炼出最关键的因素，为构建高效的预测模型打下了基础。

3.3.4 模型评估

利用收集和处理过的数据，本研究应用了深度神经网络（deep neural network，DNN）[24]、支持向量回归（support vector regression，SVR）[25]、随机森林回归（random forest regressor，RFR）[26]、类别提升（category boosting，CatBoost）[27]、轻量级梯度提升机（light gradient boosting machine，LightGBM）[28]、以及极端梯度提升（extreme gradient boosting，XGBoost）[29]等多种机器学习算法构建性能预测模型。这些模型旨在建立系统参数配置与 Nginx 性能（延迟）之间的精确关系。模型的评估通过交叉验证[30]进行，以确保模型的泛化能力。同时，采用贝叶斯优化技术[31]微调模型的超参数，确保了高准确性的性能预测，形成了 SDS 数据集训练好的模型（static trained model，STM）和 DDS 数据集训练好的模型（dynamic trained model，DTM）。

3.3.5 全局搜索

在最终步骤，本研究利用遗传算法（genetic algorithm，GA）[32]结合先前训练好的

STM 和 DTM 模型，执行全局参数搜索。此过程的核心目标是识别出能够显著提升 Nginx 在 SLT 和 DLT 负载测试性能的最佳系统参数配置。选择遗传算法作为本研究参数搜索的工具，是因为在面对复杂的优化问题时，GA 展现出了其卓越的全局搜索能力和避免陷入局部最优解的能力。此外，现存众多算法，如模拟退火算法[33]、粒子群优化算法[34]和差分进化算法[35]虽也用于探索复杂配置空间，但它们或显示出缓慢的收敛速度，或存在陷入局部最优解的风险。相比之下，GA 通过模拟自然界的进化和遗传机制，在解决包含多个局部最优解的复杂问题方面表现更为出色。尤其是，GA 在多个局部最优之间的搜索鲁棒性使其能够在充满挑战的、庞大的配置参数空间中精确定位全局最优解。因此，针对本课题旨在一个充满局部最优的庞大配置参数空间中识别出能够最大化 Nginx 性能的配置这一充满挑战的探索领域，本研究选择了遗传算法。通过这种方法，本研究期望能够准确定位那些能够优化 Nginx 性能的关键系统参数配置。

本研究方法的设计和执行全面覆盖了 X86 和 ARM 硬件平台，每个实验步骤都独立地在两种平台上进行，以确保结果数据的全面性和精确性。这种系统化和全面的实验设计不仅深入探讨了 Nginx 性能优化的有效方法，而且成功揭示了 X86 和 ARM 架构在性能优化过程中的不同表现及其原因。通过精确的参数筛选、全面的数据收集与处理、先进的模型评估方法，以及有效的全局优化搜索策略，本研究不仅提高了参数调优的可行性和可靠性，而且为在不同硬件平台上部署和优化 Web 服务器提供了深刻的见解和有效的策略。

4 结果及分析

本研究主要关注于模型精度、遗传算法的迭代收敛和不同硬件架构的 Nginx 性能提升对比。

4.1 模型精确度

本研究对六种不同的回归模型进行了全面评估，以确定它们在预测 Nginx 服务器性能方面的效果。

具体的模型误差如图 2 所示，通过深入的对比分析，本研究观察到 XGBoost、LightGBM、CatBoost 和 RFR 这四种集成学习模型在 X86 架构服务器上的平均预测误差分别为 16%、15%、17%和 19%，而在 ARM 架构服务器上，相应的平均预测误差则为 15%、15%、12%和 16%。这一结果表明，尽管这四个模型的预测误差都保持在 20%以下的阈值，但在 X86 和 ARM 两种硬件平台上，模型的误差差别并不显著，显示了这些集成学习模型在不同硬件平台上的稳定性和准确性。

相对而言，DNN 和 SVR 模型在 X86 服务器上的平均误差分别达到了 25%和 26%，而在 ARM 服务器上，这些误差稍有下降，分别为 23%和 22%。这进一步强调了 XGBoost、LightGBM、CatBoost 和 RFR 在预测精度上相对于 DNN 和 SVR 模型的优势，以及它们在跨平台应用时保持误差低和稳定性高的能力。

基于上述分析，本研究选择误差较低的 XGBoost、LightGBM、CatBoost 和 RFR 模型作为 GA 算法调用的模型（即 STM 和 DTM 模型）。这一选择不仅是基于这些模型在 X86 和 ARM 平台上所展示的准确性，也是考虑到它们在处理不同架构服务器数据时的稳定性和鲁棒性[36]。这证明了集成学习模型在预测 Nginx 服务器性能方面的高效性，同时强调了在进行硬件平台选择时，模型的准确性和稳定性是两个重要的考量因素。

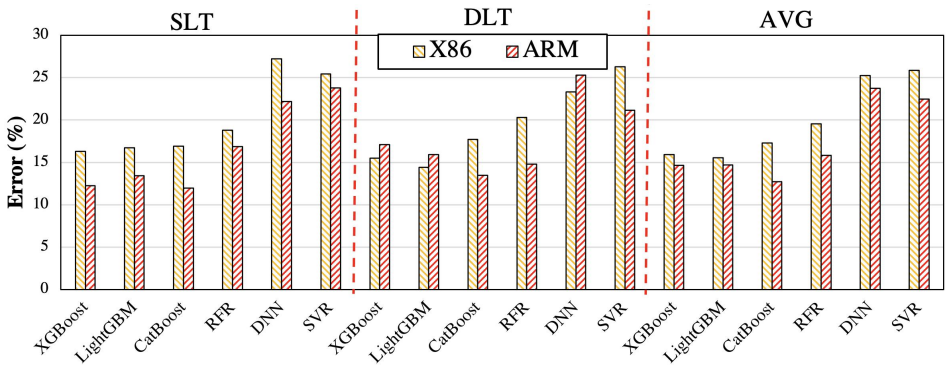

图 2 模型误差
Fig. 2 Model Error

4.2 遗传算法的迭代收敛

本研究采用 GA 算法深入探索操作系统参数配置，以确定在高并发负载测试下 Nginx 的最优配置。实验旨在探讨如何有效减少寻找最优配置所需的迭代次数及时间。

4.2.1 遗传算法的迭代收敛次数

GA 算法迭代收敛次数如图 3 所示，在 SLT 和 DLT 负载测试中，相较于 X86 架构，ARM 架构所需的迭代次数显著减少，分别从 34 至 77 次降至 22 至 54 次。ARM 架构相比 X86 架构所需迭代次数更少的原因，可以归结于其架构设计原则和能效优化。ARM 的 RISC 架构设计原则[7]，使其在执行 GA 搜索时，由于更简洁的指令执行流程而更高效，特别是在处理遗传算法这类需要大量简单迭代操作的场景中。此外，ARM 架构的能效优化特性可能在 GA 迭代过程中的资源管理方面产生积极影响[7]，使 ARM 服务器在追求最优配置过程中能维持更高的运算效率。这两点因素共同导致 ARM 架构在寻找最优参数配置时需要更少的迭代次数。

此外，实验还发现，负载测试的类型对迭代效率具有重要影响。例如，在图 3 (a) 和图 3 (b) 的 X86 架构中，采用 XGBoost 模型进行 SLT 训练所需的迭代次数为 35 次，而进行 DLT 训练则需 60 次；相对应地，在图 3 (c) 和图 3 (d) 的 ARM 架构中，SLT 和 DLT 训练的迭代次数分别为 23 次和 44 次。这些发现揭示了优化过程中架构和请求类型的显著影响。进一步分析，SLT 和 DLT 训练所需迭代次数的差异反映了不同请求类型的影响。由于 SLT 通常涉及对固定资源的请求，其数据复杂性相对较低，这使得集成学习模型如 XGBoost、LightGBM、CatBoost 和 RFR 能够更快地识别出影响性能的关键特征并快速调整参数，从而减少所需的迭代次数。相比之下，DLT 的动态内容增加了数据的复杂性，导致模型在适应和学习这种复杂性时需要更多的迭代。

4.2.2 遗传算法的迭代收敛时间

GA 算法迭代时间如图 4 所示，尽管 ARM 架构在迭代次数方面相对 X86 架构有优势，但这并不直接导致其迭代时间总体较短。具体地，在 SLT 负载测试中，ARM 架构结合 LightGBM 模型寻找最优配置的时间仅为 2 分钟，而 X86 架构需要 4 分钟。相反，在 DLT 负载测试中，X86 架构使用 LightGBM 模型找到最优配置的时间减少至 4 分钟，而 ARM 架构所需时间延长至 12 分钟。这些结果揭示了两种架构在处理 Nginx 的不同负载测试方面的明显差异：ARM 架构在处理静态负载测试方面展现出显著的效率优势，而 X86 架构在动态负载测试中表现更为高效。

这一差异的原因可以从几个方面进行分析：（1）架构设计特点：ARM 架构的 RISC 架构优化了简单重复任务的处理效率[7]，这对于 SLT 负载测试中相对简单、重复的请求处理具有优势。相比之下，X86 架构的 CISC 架构更适合处理复杂的计算任务[6]，如 DLT 负

载测试中的动态内容生成和处理。（2）系统资源管理：ARM 架构在能效管理方面的优化可能使其在执行简单、计算不密集的任务时更加高效[7]，从而在静态负载测试中快速收敛。而 X86 架构由于其强大的计算资源和优化的资源调度机制[6]，在处理资源密集型的动态请求时能够更快地找到最优配置。（3）负载测试特性：SLT 负载测试通常涉及对固定资源的请求[19]，这种测试的性能优化较为直接和简单。而 DLT 负载测试涉及更多的逻辑处理和资源动态分配[19]，其优化过程更为复杂和时间消耗。

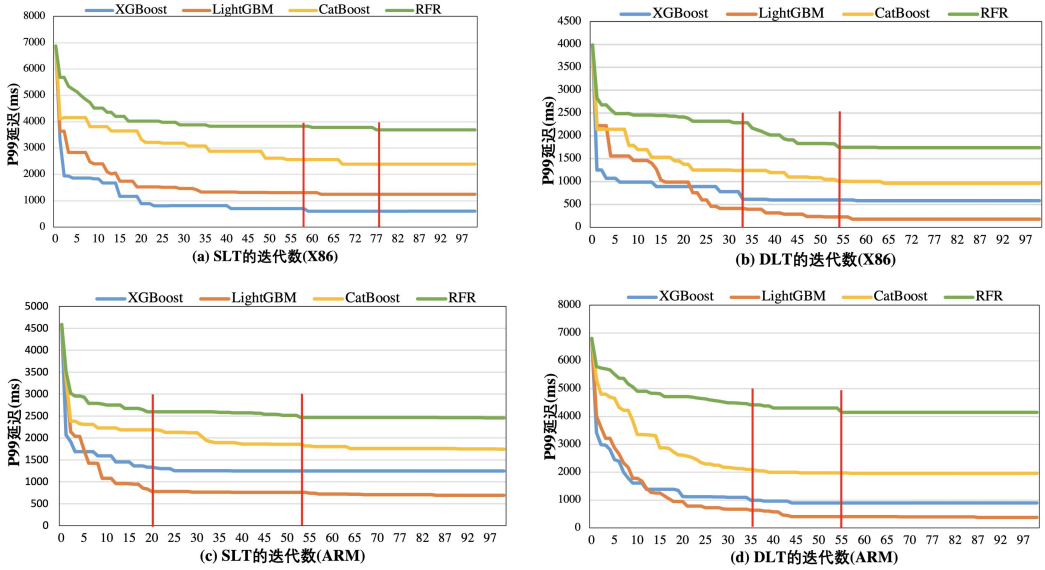

图 3 X86 和 ARM 架构在 SLT 和 DLT 负载测试中的迭代数

Fig. 3 The number of iterations in SLT and DLT load tests for X86 and ARM architectures

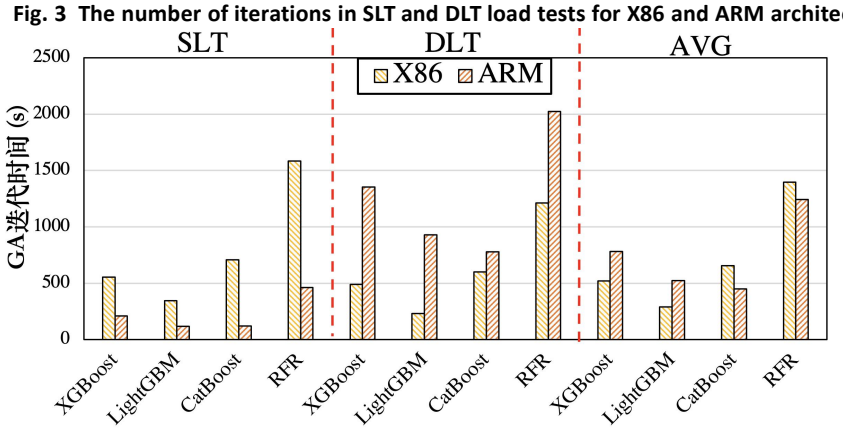

图 4 GA 算法的迭代时间

Fig. 4 Iteration time of the GA algorithm

4.3 性能提升对比

本研究专注于通过 GA 算法对操作系统参数进行自动调优，目标是在高并发负载测试条件下为 Nginx 实现最优配置。本研究的核心在于评估自动调优对于 X86 和 ARM 架构在 SLT 与 DLT 负载测试中 Nginx 性能的影响。

根据图 5 (a) 所示，自动调优后，X86 架构在性能上平均提升了 3.8 倍，达到最高 22 倍的提升，而根据图 5 (b) 所示 ARM 架构的性能平均提升了 3.3 倍，峰值提升达到 12 倍。这些显著的性能提升表明，默认配置在资源分配和利用上存在显著的局限性。

根据图 6 所示，深入的结果分析揭示了在 SLT 中，ARM 架构展现出相对于 X86 架构更低的第 99 百分位（P99）延迟，具体减少了 220 毫秒，意味着 Nginx 性能优化提升达到 60%。这一结果的背后，可能主要得益于 ARM 架构在处理静态内容方面的高能效比及资

源利用效率[7]。ARM 架构的设计原则优化了能效与处理速度，使其在静态负载测试下表现出较好的性能。

反观 DLT 中的情况，X86 架构的 P99 延迟显著低于 ARM 架构，具体减少了 515 毫秒，性能提升达 287%。这种显著的差异指向了 X86 架构在处理动态内容请求时更为强大的计算能力和更优的并发处理机制[6]。X86 架构的 CISC 架构设计使其在执行复杂的动态内容处理任务时，能够有效地管理和执行计算密集型操作，因而在 DLT 中展现出更高的性能。

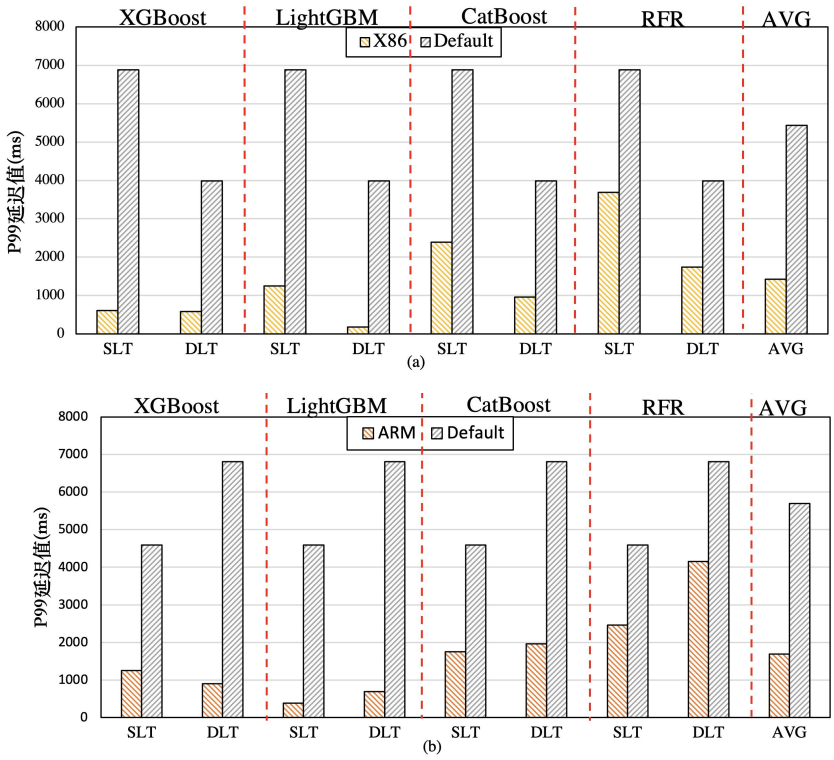

图 5 X86 与 ARM 架构在 SLT 与 DLT 负载测试中相对于默认配置的加速对比

Fig. 5 X86 vs. ARM Architecture Acceleration Relative to Default Configuration in SLT vs. DLT Load Tests

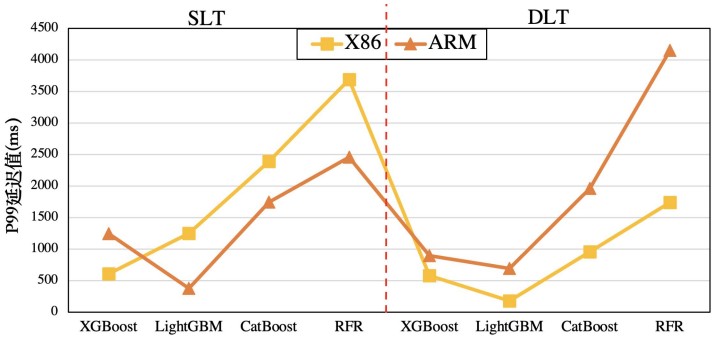

图 6 X86 与 ARM 架构在 SLT 与 DLT 负载测试中的加速对比

Fig. 6 X86 vs. ARM Architecture Acceleration in SLT vs. DLT Load Tests

综上所述，本研究不仅凸显了自动参数调优在提升 Nginx 服务器性能方面的重要性，也深入探讨了不同硬件架构上处理静态与动态负载测试时性能差异的深层次原因。具体来说，本研究的发现可以归纳为三个主要点：（1）集成学习模型在 X86 与 ARM 硬件架构下均表现出高准确性，突显了跨平台性能优化中模型选择的重要性；（2）ARM 架构在迭

代次数上较 X86 显示出优势，但迭代收敛时间的对比显示两种架构各有千秋，其中 ARM 在静态测试中更迅速，而 X86 在动态测试中更高效；（3）性能提升的对比分析揭示了 ARM 架构在处理静态内容时的优势，与 X86 架构在动态内容处理上的高效性。这些发现为未来在多样化硬件平台上部署和优化 Web 服务器提供了重要的理论和实践价值，强调了性能优化时考虑硬件架构特性的必要性，并为利用遗传算法进行操作系统参数优化的研究领域贡献了宝贵的见解。

5 进一步讨论

本研究深入探讨了在 X86 与 ARM 硬件架构下进行 Nginx 性能调优的复杂性，揭示了其中的关键局限性与误差来源，并提出了不同硬件架构的特性及其迭代效率的考量。

5.1 本研究的局限性及误差来源

本章节探究了 Nginx 性能调优在 X86 与 ARM 架构下的差异性及其背后的原因，揭示了优化过程中的关键局限性与潜在误差来源。

5.1.1 局限性

（1）硬件架构的本质差异导致的性能波动：X86 与 ARM 架构的设计理念和实现方式在根本上不同，直接影响了性能优化的适用性和效果。X86 架构的 CISC 架构设计优化了复杂操作的处理能力，适合执行多任务处理和高负载应用程序[6]。这一点在负载测试中得到了体现，其中 X86 架构显示出比 ARM 更快的数据处理能力，尤其是在处理动态内容和高并发请求时。相比之下，ARM 架构利用其 RISC 架构设计在能效和处理简单任务上具有优势，特别是在能源消耗敏感的应用场景中表现更佳[7]。这些差异导致针对一种架构优化的策略不一定适用于另一种，增加了跨架构性能优化的复杂性。

（2）实验环境的限制性对泛化性的影响：本研究在固定的实验环境中进行，特别是选用 Ubuntu 18.04.6 LTS 作为操作系统平台，可能限制了结果的泛化能力。操作系统版本之间的差异，如 Linux 内核的更新，可能带来网络栈、文件系统处理及安全性能的改进，从而影响 Nginx 的性能。这意味着，本研究的发现可能不完全适用于运行更新或不同版本 Linux 操作系统的环境，限制了结果在不同技术背景下的应用范围。

（3）Nginx 配置多样性的考量不足：Nginx 性能的优化不仅取决于硬件架构和系统级参数，还受到 Nginx 自身配置的影响[37]。实验中，特定的参数设置（如 worker_processes 和 worker_connections）反映了特定的应用场景，但实际应用场景的多样性意味着这些配置可能并不普遍适用。例如，静态内容分发与动态内容生成对 Nginx 配置的需求存在显著差异。本研究未能充分考虑这一点，可能限制了优化策略的适用性和有效性，在不同应用场景下的性能提升效果可能有所不同。

通过深入分析这些局限性，本研究为未来的性能优化研究提供了重要的考虑因素，强调了在制定优化策略时需要全面考虑硬件架构特性、操作系统版本以及 Nginx 配置多样性的重要性。

5.1.2 误差来源

（1）网络条件的实际复杂性：在实验设置中，假定的理想网络环境忽略了带宽限制和丢包等关键因素，这些因素在真实世界应用中对性能有决定性影响。例如，带宽限制和丢包现象可能会严重降低 Nginx 服务器处理高流量请求的能力[38]。因此，未能考虑这些网络变量可能导致性能评估结果与实际应用场景存在偏差。

（2）性能评估模型的准确性问题：自动调优过程对性能评估模型的依赖性高，但这些模型的预测准确性受限于训练数据集的代表性和质量[39]。例如，若训练数据未覆盖 Nginx 在极端负载下的行为，则模型可能无法准确预测这些情况下的性能表现。此外，模

型超参数（如 XGBoost 的学习率和树的深度）的选择对预测结果有显著影响。不恰当的超参数设置可能导致模型过拟合或欠拟合，从而降低预测的准确性和可靠性。

（3）全局搜索策略的局限性：尽管理论上遗传算法可以找到全局最优解，但在实际应用中，该算法的搜索效率和结果质量受到种群大小、交叉率和变异率等算法参数的影响[40]。特别是在参数空间复杂度高的情况下，遗传算法可能陷入局部最优解，而无法达到真正的全局最优。此外，算法的收敛速度可能受到参数维度的影响，导致在高维参数空间中寻找最优解需要长时间迭代。这些局限性表明，即使采用了先进的全局搜索策略，也无法保证总是能快速有效地找到最优的 Nginx 配置参数组合。

通过对这些误差来源的深入分析，本研究强调了在性能评估和优化过程中考虑实际网络条件、提高模型训练的质量和代表性，以及优化全局搜索策略的重要性，以提高 Nginx 性能调优研究的准确性和可靠性。

本研究的局限性和误差来源的深入剖析揭示了在 Nginx 性能优化研究中必须考虑的复杂性和挑战。本研究发现，硬件架构的本质差异、实验环境的限制性、以及 Nginx 配置的多样性均可能导致优化策略的适用性和有效性受限。此外，误差来源如网络条件的实际复杂性、性能评估模型的准确性问题，以及全局搜索策略的局限性，进一步增加了性能优化工作的难度。这些发现强调了未来研究中需要采取更全面的方法，包括考虑实际网络环境、使用更精准的性能评估模型，以及改进全局搜索策略，从而提高性能优化的效率和效果。

5.2 不同硬件架构的特性及其迭代效率的考量

在探索不同硬件架构对 Nginx 性能优化的影响时，理解和利用每种架构独特的特性、迭代效率的影响和软硬件协同优化的重要性至关重要。：

5.2.1 确定和利用架构特性

（1）性能与能效的平衡：X86 架构适合高并发的 Web 服务，这一点在大型电商平台的实际部署中得到验证，其中 X86 处理并发请求的能力显著优于 ARM。相反，ARM 架构在物联网应用中展现出其能效优势，降低能源消耗同时保持合理的响应速度，这对于能源敏感的部署场景至关重要。

（2）针对特定硬件的优化策略：ARM 架构在静态内容分发中因其高能效比而减少运营成本，实证表明能耗降低约 40%。而 X86 的多线程能力使其在动态内容生成中表现出色，例如增加 Nginx worker 进程以提升处理能力，这在处理社交媒体背景任务时特别有效。

（3）软件配置的硬件适配：根据 ARM 和 X86 的特点调整 Nginx 配置，如 ARM 上降低 worker_processes 以减少能耗，而在 X86 上提高该值以充分利用计算资源。通过这种方法，可在新闻网站等高流量场景中实现最佳性能。

5.2.2 提高迭代效率的策略

（1）基于模型的参数调优：使用 XGBoost 等机器学习模型预测参数调整对性能的影响，提高调优精度。这避免了低效的盲目尝试，使得性能测试更加目标导向，减少了不必要的迭代。

（2）并行测试与评估：利用 Docker 容器并行运行多个 Nginx 实例进行性能测试，大幅提高测试效率。这种方法使得同时评估多种配置成为可能，缩短了优化周期。

（3）动态调整搜索策略：根据实时反馈调整遗传算法中的参数，如提高变异率以探索更广的参数空间，针对性地优化对性能影响较大的参数，使搜索过程更加高效。

5.2.3 综合考虑软硬件因素

（1）软硬件协同优化：Linux 内核参数和 Nginx 配置的调整需考虑硬件特性，如在

ARM 上优化 vm.swappiness 以减少交换操作，同时调整 Nginx worker_processes 以匹配硬件能力，实现软硬件的最佳匹配。

（2）跨架构性能基准测试：定期执行基准测试，评估不同硬件架构的性能差异，如 ARM 处理静态请求的效率高于 X86，为系统管理员提供决策依据，同时监控由于技术进步导致的性能变化。

（3）灵活的优化策略：随技术进步和应用需求变化调整优化策略，如新兴 ARM 服务器的性能提升可能使其更适用于之前由 X86 主导的高性能计算任务。同时，优化策略应随云计算和大数据技术的演进而演变，确保策略的现代性和适用性。

通过这些策略，本研究不仅能够针对不同硬件架构的特性制定出更为精确和高效的性能优化措施，还能在快速变化的技术环境中保持优化策略的前瞻性和灵活性，为 Nginx 服务器的性能优化提供持续的支持。

综上所述，本研究深入探讨了 Nginx 性能调优过程中可能遇到的关键局限性和误差来源，并提出了针对不同硬件架构的优化策略及其迭代效率的考量。本研究强调了考虑硬件架构差异、实验设置限制，以及调优方法局限性的重要性，并探讨了提高迭代效率的策略。通过模型驱动的参数选择、并行测试评估，以及动态搜索策略调整，本研究旨在提高性能优化的准确性和效率。此外，综合软硬件因素的考量对于实现最佳性能调优也至关重要。本研究为系统管理员和开发者提供了在变化的技术环境中优化 Nginx 性能的实用指导，未来研究应继续改进策略，以满足日益增长的 Web 服务需求。

6 结 论

本研究通过对 X86 与 ARM 架构下 Nginx 服务器进行系统参数优化和性能测试，揭示了两种硬件架构在不同负载条件下的性能差异。结果表明，X86 架构在处理动态请求方面具有显著优势，而 ARM 架构则在静态请求处理上展现更高效能。这一发现强调了性能优化策略需针对硬件特性和负载类型进行定制。研究还分析了性能提升的原因，包括硬件设计、资源调度效率，以及遗传算法迭代特性等因素。因此，为了最大化 Nginx 服务器的性能，系统管理员需要深入理解不同硬件架构的特点，以及如何根据这些特点调整系统参数，从而针对特定的负载条件实现优化。

参考文献

[1] Nginx. nginx news [EB/OL]. (2024,03,01)[2024-04-17]. https://nginx.org/.

[2] W3Techs. Web Server Technology Usage Statistics [J/OL]. W3Techs, (2024,02,01)[2024-04-17]. Available at: https://w3techs.com/technologies/overview/web_server/.

[3] Nguyen VN. A comparative performance evaluation of web servers [J]. VNU Journal of Science: Comp. Science & Com. Eng., 2017, 31(3): 28–34.

[4] Chen S, GalOn S, Delimitrou C, et al. Workload characterization of interactive cloud services on big and small server platforms [C] // 2017 IEEE International Symposium on Workload Characterization, 2017: 125-134.

[5] Sankaralingam K, Menon J, Blem E. A detailed analysis of contemporary arm and X86 architectures [R]. TR1783, Madison: University of Wisconsin-Madison, Department of Computer Sciences, 2013.

[6] TechSci Research LLC. X86 Server Market Report [R]. 17414, New York: TechSci Research LLC, 2023.

[7] Rahman TN, Khan N, Zaman ZI. Redefining Computing: Rise of ARM from consumer to Cloud for energy efficiency [J]. World Journal of Advanced Research and Reviews, 2024,

21(1): 817-835.
[8] Nginx. NGINX Plus on Arm-Based AWS Graviton2 AMIs [EB/OL]. (2024,03,03)[2024-04-17]. https://www.nginx.com/resources/datasheets/nginx-plus-arm-based-aws-graviton2-amis/.
[9] Ma C, Chi YH. Evaluation test and improvement of load balancing algorithms of Nginx [J]. IEEE Access, 2022, 10: 14311-14324.
[10] Laghari AA, He H, Laghari RA, et al. Cache performance optimization of QoC framework [J]. EAI Endorsed Transactions on Scalable Information Systems, 2019, 19(20): e7.
[11] Silva M, Husemann R, Malheiros M. Comparação de desempenho das arquiteturas SOC de CPU ARM Cortex A57 e Intel X86 como servidores dos softwares Apache e Nginx [J]. Academic Highlights Journal, 2017, 9(4): 281-297.
[12] Kalyanasundaram J, Simmhan Y. ARM Wrestling with Big Data: A Study of Commodity ARM64 Server for Big Data Workloads [C] // 2017 IEEE 24th International Conference on High Performance Computing (HiPC), 2017: 203-212.
[13] Chen X, Hung LH, Cordingly R, et al. X86 vs. ARM64: An Investigation of Factors Influencing Serverless Performance [C] // Proceedings of the 9th International Workshop on Serverless Computing, 2023: 7-12.
[14] Jiang Q, Lee YC, Zomaya AY. The power of ARM64 in public clouds [C] // 2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID), 2020: 459-468.
[15] Wang J, Kai Z. Performance analysis and optimization of nginx-based web server [J]. Journal of Physics: Conference Series. 2021, 1955(1): 012033.
[16] Sharma R. Nginx high performance [M]. Birmingham: Packt Publishing Ltd, 2015: 86-90.
[17] Wang R, Wang Q, Hu Y, et al. Industry practice of configuration auto-tuning for cloud applications and services [C] // Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2022: 1555-1565.
[18] Chen WX, Li LL, Yu ZB. INTOP: Improving Nginx Performance by Tuning OS Parameters [C] // 2023 International Conference on High Performance Big Data and Intelligent Systems (HDIS), 2023: 16-23.
[19] DeJonghe D. Nginx CookBook [M]. Sebastopol: O'Reilly Media, 2020.
[20] Tene G, Barker M. Wrk2: A constant throughput, correct latency recording variant of wrk [EB/OL]. (2019-09-22)[2024-04-17]. https://github. com/giltene/wrk2.
[21] 陶辉. 深入理解 Nginx-模块开发与架构解析 [M]. 北京: 机械工业出版社, 2019.
[22] Friedman J H. Greedy function approximation: A gradient boosting machine [J]. The Annals of Statistics, 29(5): 1189-1232.
[23] Çetin V, YILDIZ O. A comprehensive review on data preprocessing techniques in data analysis [J]. Pamukkale Üniversitesi Mühendislik Bilimleri Dergisi, 2022, 28(2): 299-312.
[24] Wang X, Zhao Y, Pourpanah F. Recent advances in deep learning [J]. International Journal of Machine Learning and Cybernetics, 2020, 11: 747-750.
[25] Brereton RG, Lloyd GR. Support vector machines for classification and regression [J]. Analyst, 135(2): 230-267.
[26] Segal MR. Machine learning benchmarks and random forest regression [R]. CA 94143-0560, San Francisco: University of California, San Francisco, 2004.
[27] Prokhorenkova L, Gusev G, Vorobev A, et al. CatBoost: unbiased boosting with categorical

features [C] // Proceedings of the 32nd Conference on Neural Information Processing Systems (NeurIPS 2018), 2018: 31.

[28] Ke G, Meng Q, Finley T, et al. Lightgbm: A highly efficient gradient boosting decision tree [C] // Proceedings of the 31st Conference on Neural Information Processing Systems (NIPS 2017), 2017: 30.

[29] Chen T, Guestrin C. Xgboost: A scalable tree boosting system [C] // Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining, 2016: 785-794.

[30] Raschka S. Model evaluation, model selection, and algorithm selection in machine learning [Z/OL]. arXiv preprint arXiv:1811.12808, 2018.

[31] Wu J, et al.. Hyperparameter optimization for machine learning models based on Bayesian optimization [J]. Journal of Electronic Science and Technology, 2019, 17(1): 26-40.

[32] Kumar M, Husian M, Upreti N, Gupta D. Genetic algorithm: Review and Application [J]. International Journal of Information Technology and Knowledge Management, 2010, 2(2): 451-454.

[33] Guilmeau T, Chouzenoux E, Elvira V. Simulated annealing: A review and a new scheme [C] // 2021 IEEE Statistical Signal Processing Workshop (SSP), 2021: 101-105.

[34] Wang D, Tan D, Liu L. Particle swarm optimization algorithm: An overview [J]. Soft computing, 2018, 22: 387-408.

[35] Pant M, Zaheer H, Garcia-Hernandez L, et al. Differential Evolution: A review of more than two decades of research [J]. Engineering Applications of Artificial Intelligence, 2020, 90: 103479.

[36] Sagi O, Rokach L. Ensemble learning: A survey [J]. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, 2018, 8(4): e1249.

[37] Aivaliotis D. Mastering Nginx [M]. Birmingham: Packt Publishing Ltd, 2016.

[38] Nedelcu C. Nginx HTTP Server [M]. Birmingham: Packt Publishing Ltd, 2015.

[39] Cong G, Fan W, Geerts F, Jia X, Ma S, et al. Improving data quality: Consistency and accuracy [C] // Proceedings of the 33rd international conference on Very large data bases, 2007: 315-326.

[40] Sivanandam SN, Deepa SN, Sivanandam SN, et al. Applications of genetic algorithms [M]. Berlin: Springer Berlin Heidelberg, 2008.