Wayfinder：自动化操作系统专门化

Alexander Jung 兰卡斯特大学 英国 Unikraft GmbH 德国

Cezar Crăciunoiu 布加勒斯特理工大学 罗马尼亚

Nikolaos Karaolidis 曼彻斯特大学 英国

Hugo Lefeuvre 不列颠哥伦比亚大学 加拿大

Daniel Oñoro Rubio NEC欧洲实验室 德国

Felipe Huici Unikraft GmbH 德国

Charalampos Rotsos 兰卡斯特大学 英国

Pierre Olivier 曼彻斯特大学 英国

arXiv:2603.23425v1 [cs.OS] 24 Mar 2026

摘要

操作系统特化是一种广为人知的方法，可用于优化特定应用程序的性能、内存使用量、安全性或其他重要指标。针对某个应用程序对操作系统进行特化通常是一个手动过程，需要丰富的专业知识。通过配置进行专门化很适合自动化；然而，由于现代操作系统的配置空间规模庞大、难以量化、评估配置耗时较长，且存在大量无效配置，因此颇具挑战性。因此，现有的自动操作系统特化尝试仅限于启用或禁用功能特性，以尽量减少内存消耗或攻击面，无法针对性能等指标进行优化。

我们提出了 Wayfinder，这是一种能够完全自动化地专门配置操作系统、无需专家知识的框架。它可以针对在给定硬件配置上处理给定工作负载的应用程序，围绕任何可量化的性能、资源消耗或安全性指标，对操作系统配置的所有方面（编译时、启动时和运行时）进行专门化。Wayfinder 由一个自动化操作系统基准测试平台和一个驱动专门化过程的基于神经网络的搜索算法组成。其实现方式是在运行过程中学习哪些配置参数及其取值对性能影响最大，以及哪些配置会导致运行时故障。此外，还可以重复利用针对某个应用程序预训练的模型，以加速相关应用程序的专门化。我们在两个操作系统上评估了 Wayfinder，四个应用程序和两个目标指标：与默认配置相比，Wayfinder 完全自动地识别出专用配置，使应用程序性能最高提升 24%，内存使用量减少 8.5%。我们重点展示了神经网络的优势：它比竞争方法（随机方法和贝叶斯方法）更快地找到良好解决方案，并成功地在相关应用程序之间迁移知识。

CCS 概念： • **软件及其工程 → 操作系统**； **软件配置管理和版本控制系统**。

关键词： 操作系统、专门化

ACM 引用格式：

Alexander Jung、Cezar Crăciunoiu、Nikolaos Karaolidis、Hugo Lefeu- vre、Daniel Oñoro Rubio、Felipe Huici、Charalampos Rotsos 和 Pierre Olivier。2026年。Wayfinder：自动化操作系统专门化。载于第*21*届欧洲计算机系统会议（*EUROSYS '26*），*2026*年*4*月*27*日至*30*日，苏格兰爱丁堡。ACM，美国纽约州纽约市，18 页。https://doi。org/10。1145/3767295。3803589

1 引言

主流操作系统的通用性是以牺牲性能、资源消耗或安全性为代价的 [13, 15, 25, 46, 48, 70]。因此，权衡通用性所带来的成本的一种常见技术是操作系统专门化 [8, 25, 33, 46], ，即针对特定用例（例如应用程序、工作负载或硬件平台）定制操作系统的过程。操作系统专门化是系统研究中的热门课题，相关研究针对输入/输出 [15, 42, 57,65, 70, 73],图形处理器或硬件加速器 [81–83], 资源使用 [2, 42, 44, 54, 55], 安全性 [45, 49, 80, 91], 兼容性 [9, 66, 67, 69, 90], 或可扩展性 [14]展开。

大多数操作系统专门化工作都依赖专家开发者手动调整或重新实现特定的操作系统子系统。不幸的是，这类方法依赖专家的领域

本作品采用知识共享署名 4.0 国际许可协议授权。*EUROSYS '26*，苏格兰爱丁堡© 2026年版权归所有者/作者所有。ACM ISBN 979-8-4007-2212-7/2026/04
https://doi.org/10.1145/3767295.3803589

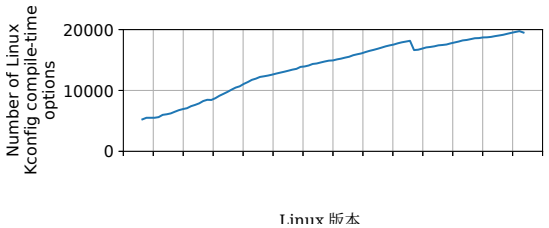


图 1. Linux 随时间变化的编译时配置空间。

知识，使绝大多数用户无法从这些技术中受益。为了解决这个问题，我们需要在操作系统专门化方面实现更多自动化。

我们专注于通过配置实现操作系统专门化：操作系统允许用户通过编译时、启动时或运行时配置参数，自定义其行为并针对特定用例调整功能。尽管与重新设计操作系统的部分组件相比，配置特化带来的收益可能相对有限，但研究表明，它在许多场景中都能带来显著效果 [39, 44, 45, 77, 88]。此外，这种方法非常适合自动化，无需专家知识即可实现。尽管先前的研究已探索通过自动启用和禁用功能来降低资源消耗 [1, 40, 77] 或缩小攻击面 [45, 88, 89], ，但在针对性能等其他指标时，这些研究都无法应对现代操作系统所提供的庞大而复杂的配置空间所带来的挑战。

我们提出了 Wayfinder，这是一个用于为应用程序全自动地专门化操作系统（如 Linux）配置的框架，且无需专家知识。Wayfinder 可自动针对目标应用程序在给定硬件配置上处理给定工作负载时的任何可量化指标（包括性能、资源消耗、安全性或兼容性），对操作系统配置的任意部分进行专门化，例如编译时、启动时和运行时参数。该框架包含一个自动化的操作系统配置、构建和基准测试平台，旨在支持可复现测试。Wayfinder 的核心基于 DeepTune，这是一种新颖的基于神经网络的优化算法，用于驱动专门化过程。DeepTune 的核心思想是通过预测以下内容来引导搜索，并据此选出有价值的候选配置：1）它们在目标指标上的表现（例如吞吐量、内存使用量）；2）它们有效的可能性（例如能够成功编译、运行时不会崩溃）。

Wayfinder 应对了若干挑战。首先，现代操作系统的配置空间极其庞大。如图1所示，Linux 6.0 有约 20000 个编译时选项，此外还有启动时和运行时参数。一些参数的取值可以是任意数字，这使问题更加棘手：总之，不可能探索每一种配置。其次，如果没有专家知识，就很难定义搜索空间。许多内核参数几乎没有文档说明：参数类型和有效取值范围都不得而知。

第三，评估一种配置需要时间，因为这可能需要构建内核映像、启动系统并运行测试，以评估目标指标。评估配置所需的时间可能差异很大，而自动化操作系统调优系统应优化发现专用配置所需的总时间。最后，搜索空间中包含许多纸面上有效的配置（例如，在 Linux 中，这些配置满足 KConfig 检查的约束），但它们会在编译时、启动时或运行时导致失败。我们的评估表明，使用朴素的随机搜索方法优化 Linux 操作系统配置时，约三分之一的尝试会失败，造成大量搜索迭代白白浪费。再加上评估速度缓慢，这会严重限制在给定时间预算内可探索的配置数量。

Wayfinder 的设计通过以下方式应对这些挑战。对于搜索空间庞大且难以界定的操作系统（例如 Linux），我们使用一种启发式算法离线确定搜索空间，该算法可推断每个操作系统参数的类型和取值范围。随后，我们的在线探索依赖 DeepTune——一种神经网络优化算法，它会逐步学习对目标指标性能影响最大的操作系统配置参数及其取值。为加快搜索，DeepTune 会学习避开可能触发故障的参数取值；随机搜索或贝叶斯优化等竞争方法则不具备这一特性。默认情况下，每针对一个应用程序／指标进行专门优化，学习过程都会从头开始。也可以先训练模型以优化特定应用程序，再应用迁移学习，即复用该模型，加快对其他具有相似特征的应用程序进行探索。

我们将 Wayfinder 应用于为热门云应用程序专门优化两种操作系统 Linux 和 Unikraft [42], 的配置。我们针对两个目标指标进行优化：应用程序性能和内存使用量。与因果推理 [38] 或贝叶斯优化 [39],等竞争方法不同，Wayfinder 能够扩展到现代操作系统（如 Linux）的庞大设计空间，并且相比随机搜索基线，能快数倍地找到专门优化的配置。Wayfinder 能高效预测故障，将崩溃率从 30% 降至 10%–25%。最后，我们证明迁移学习能够有效缩短搜索时间，并将崩溃率降至 10% 以下。例如，使用在 Redis 上训练的模型并将其应用于 Nginx，搜索速度可提升 24%，崩溃率低于 5%。

总的来说，本文做出了两项核心贡献：

- Wayfinder，一个无需专家知识即可自动配置、构建、运行和基准测试操作系统的评估平台。
- DeepTune 是一种新颖的神经网络优化算法，可驱动 Wayfinder 的专门化流程。

表1. Linux 6.0 的配置空间，包括启动时选项（内核命令行参数）、运行时选项（/proc/sys 和 /sys 中可写的文件）以及编译时选项（通过解析 Kconfig 层级结构获得）。

编译时选项					启动时选项	运行时选项
bool	tristate	string	hex	int		
7585	10034	154	94	3405	231	13328

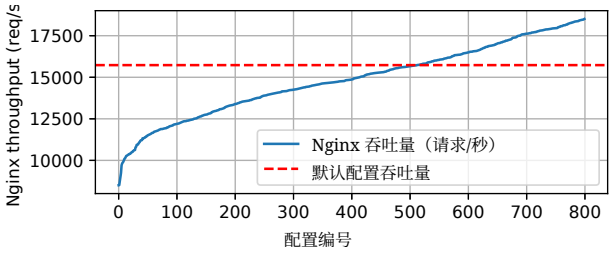


图2. Linux 内核的 800 种随机配置下的 Nginx 吞吐量。

2 背景、挑战与动机 2.1 通过配置实现专门化

操作系统专门化针对特定用例（例如应用程序、工作负载、硬件平台）和特定指标（例如性能、资源使用）对操作系统进行优化。操作系统专门化有许多方法 [14, 42, 45, 49, 54, 69, 70, 81]。我们专注于通过配置进行专门化 [1, 40, 44, 45,55, 66, 73, 80, 89],，即通过微调操作系统的编译时、启动时和运行时参数来实现专门化。

先前的研究探索了通过配置实现操作系统专门化，旨在减少资源使用（例如内存占用） [1, 40, 44, 55, 66, 73, 77] 和攻击面 [45,80, 88, 89]。其中一些方法需要手动操作 [42, 44,66, 73, 80], ，因此，鉴于 Linux 等真实世界操作系统的配置空间规模庞大，这些方法无法扩展到少数几个应用之外 [44, 73, 80]。为了解决这一问题，其他研究探索了用于专门化操作系统配置的自动化方法 [1, 40, 45, 77, 88, 89]。这些工作主要关注安全性（攻击面）或资源优化（内存占用）。因此，它们的目标是针对给定工作负载，确定哪些编译时配置参数（例如内核特性或模块）可以禁用，哪些是该工作负载必不可少的。

这些方法并不通用，也不适用于针对性能进行性能特化。除了仅启用或禁用编译时特性之外，我们还需要考虑完整的内核选项集合，其中许多选项可以取任意值，而其有效范围并不明确或未记录。为了扩展到众多应用程序，我们不能假定具备专家知识（即不能筛选相关选项）。简而言之，探索空间的规模几乎是无限的。以 Linux 内核配置空间的规模为例，如表1所示。Linux 的编译时配置包含 3000 多个选项（总计 20000 个选项中），这些选项可取任意整数。此外，由于 Linux 的通用特性，它必须适应范围广泛的工作负载，因此许多对性能至关重要的配置选项只能以运行时

参数。要正确设置这些参数以最大限度地提升性能，即使是手动设置并借助专家知识，也十分复杂，网上大量的性能调优指南便证明了这一点 [10, 16, 21, 22, 28, 58, 63, 76, 87]。对于

基于这些原因，许多以往的研究都没有探索通过配置来实现具有性能目标的自动化操作系统专门化。

2.2 优化操作系统配置以提升性能

为了进一步说明通过配置进行操作系统性能专门化的价值，我们随机生成了 800 个 Linux v4.19 配置，并评估其性能。对于每个随机配置，我们在 KVM 虚拟机中配置并运行相应的内核，在其中使用 wrk 执行基准测试，以评估 Nginx Web 服务器。我们在一台 Intel Xeon E5-2697 v2 机器上运行这些实验（2×24 个处理器核心@2.70 GHz，128 GB 内存）。我们希望获得 800 个有效配置，因此当某个配置失败时（约三分之一的随机生成配置会在运行时崩溃），我们会重新生成随机配置，直到获得有效配置。

图2 展示了将每个随机配置的性能与默认配置进行比较的结果。配置按性能升序排列。性能差异最高可达 80%，从低于 10K req/s 到 18K req/s 不等。一个关键观察结果是，最快配置的吞吐量比默认配置获得的吞吐量高 12%，尽管我们仅探索了设计空间中极小的一部分。

这说明，针对性能等指标优化操作系统配置是有意义的。不过，此处采用的随机搜索方法并非最优：在这个小型空间中，64% 的配置表现不如默认配置。此外，三分之一的配置会导致内核无法构建或启动，或在运行时崩溃/挂起（本文后文将这些情况统称为崩溃 ），从而浪费资源。考虑到探索空间规模庞大，且未对有效/无效配置加以区分，通过随机搜索寻找专门化配置会慢得难以接受。因此，我们提出 Wayfinder，利用高效的优化算法驱动配置空间探索。

2.3 基于配置的操作系统专门化优化算法

我们将搜索专用操作系统配置视为一个优化问题。以往的工作采用了各种方法，

例如贝叶斯优化 [5, 39], 因果推理 [38],和随机搜索 [39]。然而，若干问题导致这些方法不适用或效率低下：

可扩展性。 贝叶斯优化依赖高斯过程，其计算复杂性通常为 $O(n^3)$，内存消耗为 $O(n^2)$。因果分析算法的复杂性范围从 $O(n^4)$ 到 $O(n^3)$ [98], ，这使得它们只能处理相对较小的问题。**缺乏增量训练**。 在贝叶斯优化中，添加新的数据点并更新模型通常需要重新训练高斯过程。进行因果推断也需要重新计算因果图。再加上可扩展性问题，这会导致每次迭代的优化成本呈指数级增长。

难以针对高维数据拟合类别型和数值型参数。 贝叶斯优化另一个众所周知的局限是，在具有类别型输入特征 [31]（即取值范围固定的功能特性）的问题以及输入维度较高的问题上性能不佳。

总的来说，我们要处理的配置空间极其庞大，因此因果推断和贝叶斯优化都不适用。随机搜索在这样的大型空间中表现良好，但在这个问题上崩溃率很高（随机抽取的配置中有三分之一会失败）。我们将在评估中进一步展示这些局限（第 4 节）。

3 Wayfinder：设计与实现

探索空间规模庞大，而且依赖专家知识并不切实际，因此需要一个自动化探索平台。该平台可以编程实现自动配置、构建和基准测试一系列操作系统映像，从而针对特定应用程序和工作负载对其进行专门优化。

概述。Wayfinder 由一个基准测试流水线（§3.1）和 DeepTune（§3.2）组成。前者可自动构建、配置并对操作系统和应用程序映像进行基准测试；后者是一种优化算法，用于驱动探索过程，为给定应用程序找到专门优化的操作系统配置。

3.1 自动化基准测试流水线

Wayfinder 的输入包括用于表示目标操作系统配置空间的 YAML 文件（作业文件，见 §3.4），以及描述如何构建和对该操作系统映像进行基准测试的脚本，其中包括待测应用程序。

使用 Wayfinder 为给定应用程序专门定制操作系统时，探索过程由反复执行以下核心循环组成：1）根据给定配置在虚拟机中构建并启动操作系统映像；2）对运行在该操作系统映像上的目标应用程序进行基准测试；3）确定接下来要考虑的配置。用户为该平台提供时间预算或要运行的迭代次数，完成后系统会返回找到的最佳配置。Wayfinder 提供模块化 API，便于集成可插拔的搜索算法，这些算法负责完成步骤 3。它们通过采用各种方法来决定接下来要探索哪种配置，从而推动针对应用程序以及工作负载/度量的操作系统专门定制过程。目前，我们支持以下算法：

- 随机搜索：后续每个待探索配置都以随机方式生成，不考虑探索历史。
- 网格搜索：系统地探索所有可能的配置，每次尝试一个参数值。
- 贝叶斯优化：一种通过基于探索历史构建的概率模型，在探索（尝试新配置）与利用（进一步优化已知表现良好的配置）之间取得平衡的方法。
- *DeepTune*：一种基于机器学习、在探索与利用之间取得平衡的方法，详见 §3.2。

这些算法通过 API 与 Wayfinder 交互；该 API 提供各类信息，例如已探索配置的历史记录、相应的性能[1] 结果、导致构建失败或运行时崩溃的配置等。选定配置进行评估后，平台会创建两个相应的内部任务：一个构建任务，用于创建操作系统映像；另一个测试任务，用于测量其性能。这里的一项优化是，如果当前待探索配置与上一配置之间的差异仅涉及运行时参数，而不涉及启动时或编译时参数，则可以跳过构建任务。

Wayfinder 由 Go 编写，共约 1.5 万行代码，并以微服务集合的形式构建。它使用现成组件实现持久化、监控和日志记录。该平台运行在 Linux 主机上，接受基准测试的操作系统映像则运行于 QEMU/KVM 之上。

3.2 DeepTune

DeepTune 是驱动 Wayfinder 自动优化的人工智能算法。鉴于我们假设用户不具备性能调优专业知识，且面对前文所述的庞大配置空间，DeepTune 设计背后的核心思路是结合探索（即尝试新参数，找出哪些参数会产生影响）和利用（即优化已确定为重要的参数）。这种方法会自动筛选配置空间，将重点放在影响最大的参数上（相比之下，启发式方法可能需要不同程度的专业知识），同时确保不会陷入局部最优。DeepTune 通过以下方式解决我们在现有自动优化算法（如贝叶斯优化和因果推断 [38] （见 §2.3））中发现的局限性，从而兼顾高准确性和可扩展性：(1) 采用全新的神经网络（NN）模型设计来预测随机生成的排列行为，以及

[1]下文中，我们使用“性能”一词来指代一个或多个用户提供的指标的输出，这些指标可以是任何可量化的度量，例如吞吐量、延迟、内存使用量、映像大小等。

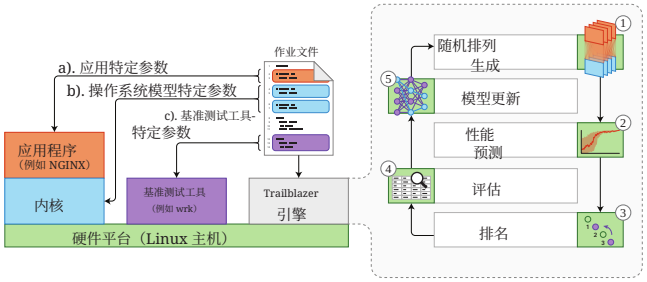


图 3。 DeepTune 的总体流程。

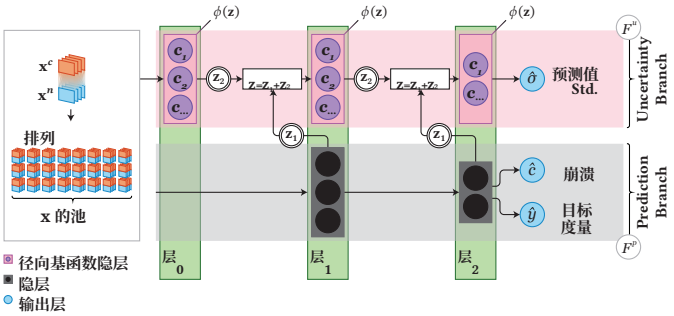


图 4。 DeepTune 模型概览。

(2) 提出一种新的评分函数，根据预测结果对配置进行排序。前者 (1) 由 DeepTune 模型（DTM）实现；DTM 是一种多任务神经网络，可预测运行时性能以及配置在运行时发生崩溃的可能性。DTM 还通过一种基于径向基函数（RBF）层 [47] 和 Chamfer 距离损失 [26]的新机制，提供不确定性度量。评分函数 (2) 综合模型预测、预测的不确定性以及与已知配置之间的不相似度，对候选配置进行排序。

图 3 展示了 DeepTune 的关键组成部分。DeepTune 首先从 Wayfinder 随机生成一组多样化的排列候选项 **1**。随后，DTM 会估算这些候选项的性能 **2**，评分函数 **3** 则对它们进行排序。Wayfinder 会评估排名第一的排列 **4**，然后更新 DTM **5**。算法会迭代预定的轮数，或直到达到性能目标为止。

DeepTune 模型（DTM）。本节应结合图 4 阅读。DTM（图 4）是一个多任务神经网络（NN），用于预测某个配置是否会崩溃、其预期性能，以及这些预测的性能不确定性。它由两个分支组成：预测分支是一个常规神经网络，用于预测排列崩溃概率和性能；不确定性分支则是一个径向基函数神经网络，用于估算预测性能的不确定性。

下文中，我们用粗体表示向量（$\mathbf{x}$），用大写字母表示矩阵（X），用小写字母表示标量（α）。我们可以将 DTM 定义为函数 $F(\mathbf{x}) \rightarrow (k, \hat{y}, \hat{\sigma})$，它将配置排列 $\mathbf{x}$ 映射为崩溃概率 k、预期性能 $\hat{y}$ 和预测的不确定性 $\hat{\sigma}$。每个配置可拆分为 $\mathbf{x} = (\mathbf{x}^k, \mathbf{x}^n)$，其中 $\mathbf{x}^k \in \mathbb{N}^k$ 是离散/类别参数的子集（例如，Nginx 的 DEFAULT_QDISC 参数可以是字符串 "pfifo"、"bfifo" 等），而 $\mathbf{x}^n \in \mathbb{R}^n$ 表示连续/整数参数。$F(\mathbf{x})$ 可拆分为不确定性分支 $\hat{F}^u(\mathbf{x}) \rightarrow (k, \hat{y})$ 和预测分支 $F^p(\mathbf{x}) \rightarrow (\hat{\sigma})$。

F^p 是一个常规的前馈深度神经网络 [41, 56, 86]。它由一系列全连接层组成，这些层使用修正线性单元（*ReLU*）激活函数和丢弃法 [86] 层。其最后一层输出性能和崩溃预测。

F^u（图 4 中的粉色区域）专门用于估算不确定性。它由一叠高斯径向基函数（RBF）层 [47],组成，每层包含神经元ϕ，其中包含质心 $\mathbf{c} = (c_1, c_2, c_{...})$。这些质心在训练过程中学习得到，可以理解为从数据集中学习到的功能特性（原型）。每个 RBF 层都与预测分支中的一个层并行。它以 $\mathbf{z}$ 作为输入，即前一层输出的功能特性/潜在表示的拼接。激活值 ϕ 的计算方式如下：

$$\phi(\mathbf{z}) = \exp\left(-\frac{\|\mathbf{z} - \mathbf{c}\|^2}{2\gamma^2}\right) \tag{1}$$

γ 是一个平滑参数，用于控制每个神经元的激活曲线有多平坦。该参数应通过经验拟合：我们发现，如果输入特征经过 z-score 标准化，a γ 取值为 0.1 较为合适。这种设计旨在增强对离群值或全新样本的鲁棒性：每个神经元的响应取决于样本与学习到的质心或数据原型之间的距离（由范数$\|\ldots\|$表示），因此当出现离群值时，输出会较低。

训练 DTM。我们通过最小化损失函数 $\mathrm{L} = L_{\mathrm{CCE}} + L_{\mathrm{Reg}} + L_{\mathrm{Cham}}$，以端到端方式训练 DTM。我们从先前的研究中引入以下三个组件：

- 分类交叉熵损失 LCCE [56] 使 DTM 能够通过学习历史数据来识别哪些排列可能导致崩溃。
- 不确定性回归损失 LReg [41] 解决了性能预测的挑战，同时量化了不确定性。该方法最初用于计算机视觉任务 [41],，我们可以将其用于使 DTM 同时估计给定排列的预期性能概况，并提供对预测预期误差的估计。这有助于在下一步排列采样策略中做出更好的选择。
- 径向基函数层的正则化损失 LCham [26] 使 DTM 能够学习质心$\mathbf{c}$，使其贴合训练

数据。直观来说， L_{Cham} 就是*Chamfer* [26] 距离，最初由计算机视觉领域提出，用于计算两个点云之间的距离 [12]。在这里，它计算质心与输入数据之间的距离；最小化该距离可以有效地分布质心，使其贴合训练数据分布。

尽管 L_{CCE}、 L_{Reg}和 L_{Cham} 已在其他领域的既有研究中出现，[12, 26, 41, 56],我们仍提出了一种在 L 中使用它们来解决该问题的新方法。

评分函数。在选择下一个要尝试的配置时，我们必须在探索与利用之间取得平衡。因此，我们提出评分函数 sf(**x** X)，它综合考虑样本**x** 与已知样本点 X之间的不相似度，以及它们的估计误差或不确定性 $\hat{}\sigma$，以优先探索欠探索区域，同时利用预期改进更大的区域。

具体而言，评分函数首先计算搜索空间中的候选点 **x** 与已知样本点 X之间的不相似度 ds(**x** X)。该差异性度量反映了样本的多样性，并指出尚未得到充分探索的区域：

$$\text{ds}(\mathbf{x}, X) = 1 - \frac{1}{(1 + ||\mathbf{x} - X||_2^2)} \tag{2}$$

我们使用预先设定的权重 α，将样本不相似度与预测样本不确定性相结合来计算最终得分。预测样本不确定性表示为 $F^u(\mathbf{x})$（将**x** 映射到 $\hat{}\sigma$，见图 4）：

$$\text{sf}(\mathbf{x}, X) = \alpha \text{ds}(\mathbf{x}, X) + (1 - \alpha) F^u(\mathbf{x}). \tag{3}$$

我们的实验表明，将 α 设为 0.5，能有效平衡利用与探索。

我们选择损失函数和评分函数的理由如下：损失函数训练系统同时识别无效（会导致崩溃）的配置并预测性能，而评分函数则利用这些预测，有策略地选择新的配置，以平衡探索与利用。RBF 分支增加了一种机制，使 DeepTune 能够估算其置信度/不确定性。

我们的 DeepTune 实现一次针对单一指标进行优化。不过，只需为 F^p 和F^u添加额外的输出层，即可扩展其功能以处理多个指标。这项修改使 DTM 能够同时预测多个目标。在评分阶段，我们对每个目标指标应用公式 3，以获得各自的分数。然后，我们通过计算加权平均值，或使用

这些单项得分的另一种聚合方法。

3.3 迁移学习

默认情况下，针对某个应用程序和度量进行优化的每个过程都从一个空白模型开始；影响性能或导致崩溃的参数必须从头重新学习。



图 5. 交叉相似度矩阵：列与行交点处的值越接近 1，表示应用程序的性能受到相似参数的影响。

迁移学习 [36]是指在某个应用程序上预训练一个模型，然后将其用于另一个相关应用程序，以加快优化过程。迁移学习背后的直觉源于不同任务之间功能特性的相似性：当应用程序具有相似的特征，且要优化的指标也相近时，在一个应用程序上预训练的模型很可能对另一个应用程序也有用。换句话说，两个应用程序的性能可能会受到相同配置参数变化的影响。例如，Redis 和 Nginx 都是网络密集型应用：在其中一个应用上训练 DeepTune 实例后，另一个应用所关注的参数子集及其取值（如网络栈参数）就已经得到部分识别，从而加快搜索。反过来，这个特定实例对 NPB 就不起作用，因为 NPB 属于 CPU 和内存密集型应用。

为验证这一假设，我们构建了交叉相似度矩阵（图 5），以评估本次评估所考察的四种应用程序中，对性能影响最大的内核配置选项之间的相似之处和差异（详情见 §4）。为创建该矩阵，我们首先针对每种应用程序收集 2,000 个随机 Linux 配置。随后，我们使用特征重要性算法 [17]来确定各配置选项在预测性能时的重要性。最后，我们将重要性得分视为向量，并计算它们之间的欧几里得范数距离。对 Nginx、Redis 和 SQLite 而言，性能影响最大的参数相似；这三者都属于系统密集型应用。Redis 与 SQLite 的相似度高于它与 Nginx 的相似度，这并不令人意外，因为 Redis 和 SQLite 都是数据库。另一方面，NPB 由于其计算密集型和内存密集型特性，对不同参数表现出敏感性。这表明，Wayfinder 的迁移学习对某些类别的应用程序可能有效。

3.4 定义探索空间

Wayfinder 要启动探索过程，需要对配置空间进行描述（即作业文件，如前文所述

见 §3.1）。该描述包含配置参数列表、参数类型以及它们可能取值的范围。Linux 等现代系统的配置十分复杂，因此要完整描述配置空间颇具挑战。虽然可以静态获取编译时和运行时参数的部分信息（例如，通过分析 Kconfig 文件和内核命令行参数说明 [53]），但运行时参数则不然：这些参数可能缺乏文档说明，甚至完全没有文档说明；而且许多参数的有效值范围无法通过静态方式确定，因为这取决于运行时因素（例如，可用内存的大小）。

我们开发了以下启发式方法来确定 Linux 配置空间。Linux 通过虚拟文件系统提供多种运行时配置选项，例如 /proc/sys、/sys。我们首先启动一个运行待研究 Linux 版本的虚拟机，并列出这些路径下可写的文件，以确定所有配置选项。对于每个可写文件，我们读取其内容，并假定返回的值对应于相关配置选项的默认值。接下来，我们通过检查默认值的类型来确定选项的类型。如果它是数字且等于 0 或 1，我们就假定该选项为布尔型。如果它既不等于 0，也不等于 1，则将其视为任意整数。

最后，我们通过将选项的默认值按较大的因子（10）多次放大和缩小，估算其可能取值范围，并尝试将选项设为这些新值，方法是向对应的伪文件写入数值。如果写入操作成功且虚拟机没有崩溃，我们就认为该新值处于该选项的有效范围内。我们有意对配置值的探索保持粗略，因为寻找性能改进将是 Wayfinder 的任务。请注意，这种技术无法处理非数值型的配置参数（例如字符串），因为很难自动确定这类参数的有效值。非数值型运行时参数非常少（见表1），因此必要时我们会转而进行人工探索。Wayfinder 将逐一探索分类参数，不对其可能取值之间的关系（例如线性关系）作任何假设。对于字符串参数，探索范围不会超出能够自动提取的值。除了 Linux 之外，这种方法也应能轻松推广到其他操作系统。我们认为，未来应为内核参数制定更正式的规范，并借鉴模糊测试社区贡献的系统调用接口定义 [32]。

3.5 在部署工作流中的集成

实用性：将 ***Wayfinder*** 集成到常见部署工作流中。我们设想在应用程序开发的测试/评估阶段使用 Wayfinder。系统以工作负载和能够代表部署条件的机器为依据，为应用程序运行时优化内核配置，并在这些条件下运行。如果在此阶段，平台确定的某项配置将部署到生产环境，工程师应审查该部署，以确保其满足生产要求。Wayfinder 可以在基准测试步骤中检查系统是否正常运行，但与任何 AI 工具一样，它无法免除站点可靠性工程师（通常还包括持续集成流水线）检查其推送到生产环境的配置的责任。如果某项配置不满足生产要求，用户可以进一步限制探索范围，排除这部分空间：

- 限制搜索涵盖的参数。 用户可以选择指定一组应保持固定值、不参与 Wayfinder 搜索过程的参数。例如，这有助于确保 Wayfinder 不会禁用相关安全选项（地址空间布局随机化等）。用户还可以指示 Wayfinder 优先调整某些类型的参数（编译时/启动时/运行时），例如，当待优化的内核无法重启时，这一功能就很有用。
- 更全面的基准测试。 用户还可以扩展 Wayfinder 使用的基准测试工具（图 4 中的紫色部分），以检查部署的其他功能（例如，运行测试套件）。如果检查未通过，Wayfinder 将学习导致异常行为的内核配置。

总的来说，我们的设计基于一个简单的假设：虽然用户不需要具备操作系统性能优化方面的专业知识，但我们假设他们能够检查部署是否满足生产要求。

安全性考量。 在对安全性敏感的场景中，若在缺乏专家知识的情况下依赖全自动化框架，确实会引发合理的安全顾虑。此处的威胁模型如下：若未意识到与安全性相关的选项，Wayfinder 可能会忽略、禁用这些选项，或在优化过程中将其移除，从而生成可能在实际部署后遭到利用的内核配置。

如前所述，我们可以将安全关键参数固定为安全值，从而启用安全感知搜索模式。在我们设想用户不具备特定专业知识的情境下，这意味着我们假定能够识别此类重要的安全相关选项。我们认为这一假设是合理的：Wayfinder 的开发者可以识别常用的安全参数，无需用户付出任何精力或具备相关专业知识。我们也可以安全地假定，用户知道任何需要设为特定值的非标准安全参数；否则，他们一开始就无法安全地运行其应用程序。最后，如上所述，与任何其他自动化工具一样，Wayfinder 的输出在投入生产前都应由可靠性工程师检查。

对工作负载和硬件的敏感性。 Wayfinder 为在特定平台上运行并处理特定

工作负载的特定应用程序专门定制内核配置。与大多数性能评估研究类似，工作负载或硬件发生变化时，需要重新运行评估，才能获得针对所考虑的软件和硬件的准确结果。Wayfinder 可以扩展为预测不同于已评估硬件或工作负载的性能，方法是采用跨工作负载 [96]和跨平台 [92] 性能估算方法，例如异构计算领域中的相关方法 [61, 68]。我们将这一目标留作未来工作。

4 评估

本次评估回答以下问题：

- 鉴于操作系统提供的配置空间极为庞大，Wayfinder 能多快为给定应用程序找到专用配置？在给定迭代次数/时间预算下，Wayfinder 能找到的最佳配置是什么？这些结果与基线和竞争方案相比如何？（§4.1）
- 当 Wayfinder 在某个应用程序上预训练模型时，应用迁移学习并使用该模型加快为其他应用程序搜索专用配置，有哪些好处？（§4.2）
- Wayfinder 预测崩溃的效果如何？（§4.3）
- 除了 Linux 和性能之外，Wayfinder 能否应用于不同的操作系统和其他度量？（§4.4）

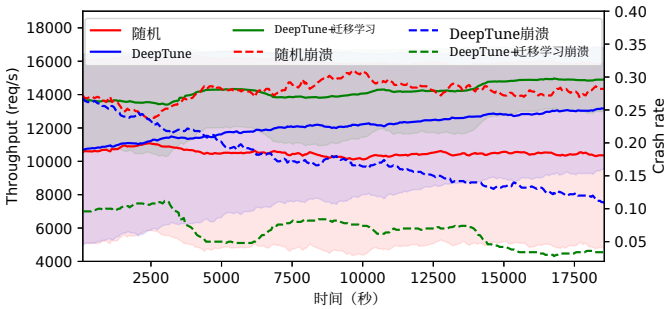
我们选择了一组热门应用程序：Nginx [64]Web 服务器和 Redis [74] 键值存储（两者都是网络密集型），SQLite [85] 数据库管理系统（存储密集型），以及 NAS 并行基准测试 [11]（NPB，CPU 和内存密集型）。Nginx 使用 wrk [99], 进行基准测试，Redis 使用 redis-benchmark [75], 进行基准测试；两种情况下，我们的优化目标都是最大化吞吐量。SQLite 使用 LevelDB 的 SQLite3 基准测试 [50],进行基准测试，该测试会执行大量 SQL插入操作；我们的目标是最小化每项操作的平均执行时间。我们使用 NPB 的 OpenMP 版本，并选取一组 CPU 和内存密集型程序：FT、MG、CG、IS（运行整个套件耗时过长），规模类别为 S、W、A 和 B。每次运行中，我们针对每个规模类别执行所有程序，并汇总各项基准测试报告的每秒操作数，这就是我们要最大化的度量。所有实验均在一台双路服务器上运行，该服务器配备 2 颗 I ntel Xeon E5-2697 v2（2×24 个处理器核心，主频 2.70 GHz，内存 128 GB），运行 Debian 10，并配置为实现高且稳定的性能：在核心 0–1 上启用 isolcpus，禁用超线程和地址空间布局随机化，并使用性能 CPU 调节器。Redis 和 SQLite 因其单线程特性而在 1 个处理器核心上运行，而 Nginx 和 NPB 则在 16 个处理器核心上运行。该服务器具有 2 个非统一内存访问节点，但我们将实验限制在单个节点上，以避免非统一内存访问影响性能测量。为避免实验共置造成干扰，所有测试配置均依次进行基准测试，不会同时共置运行实验。

除非另有说明（§4.2），Wayfinder 默认禁用迁移学习，且每个搜索过程都从头开始，使用空白模型。我们选择随机搜索 [100],作为基线；众所周知，这种方法在我们所针对的大型探索空间中表现良好。该方法通过持续生成参数值随机且彼此独特的配置来探索设计空间。我们不与网格搜索进行比较，因为众所周知，在我们所针对的大型配置空间中，网格搜索不如随机搜索。我们还展示了 Wayfinder 的表现优于贝叶斯优化 [79], , 这是一种常用于优化长时间运行的黑盒函数的技术，它利用概率模型，在尝试此前未探索过的参数与专注于已知会影响性能的参数之间进行权衡。我们还展示了 Wayfinder 优于 Unicorn [38], ；这项密切相关的工作侧重于使用因果推断优化操作系统和应用程序配置。正如我们所展示的，贝叶斯优化和 Unicorn 无法扩展到 Wayfinder 所针对的大型配置空间（例如 Linux 配置），因此我们在较小的探索空间中将其作为比较对象。我们还介绍了 Wayfinder 如何与基于编译时动态分析的优化器协同发挥作用，例如 Cozart [43], 。这项相关工作利用动态分析，大幅减少 Linux 内核配置选项的数量，从而显著缩小占用空间和探索空间。我们表明，Cozart 生成的配置可作为有效基线，供 Wayfinder 通过运行时选项进一步优化。最后，在所有实验中，用于启动每个搜索过程的初始操作系统配置均为随机设定。

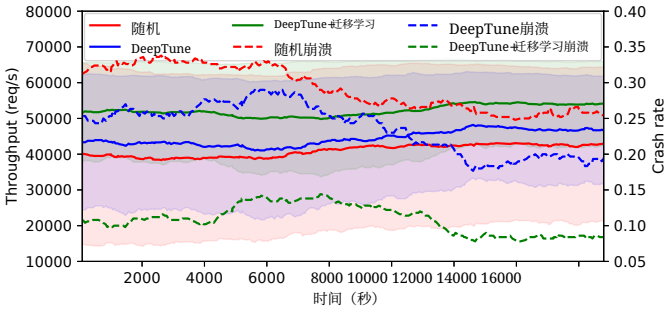
4.1 配置空间搜索的性能 arch

操作系统配置特化。 我们运行 Wayfinder，针对每个目标应用程序对 Linux 内核进行性能特化。我们使用 Debian 10 的内核 v4.19（长期支持版本）。在这些基于性能的实验中，我们将 Wayfinder 配置为优先探索运行时参数。我们分别运行 Wayfinder 和随机搜索 250 次迭代（探索 250 个参数；耗时 3.5 至 5.2 小时，具体取决于应用程序），并比较找到的配置的性能。

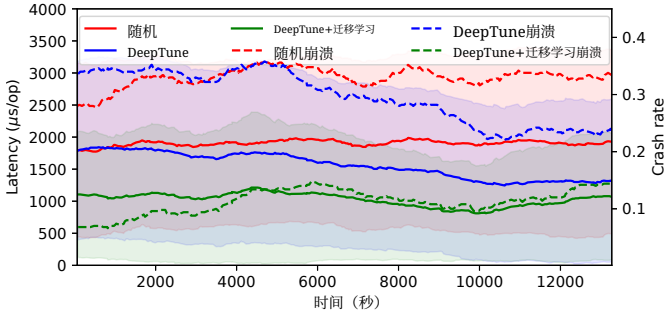
表 2 展示了 Wayfinder 找到的最佳配置的性能，以及找到这些配置所需的平均时间和默认 Lupine Linux [44] 配置的性能。Lupine 是一个针对通用性能进行专门优化的 Linux 内核，也就是说，它并非针对任何特定应用程序。为 Nginx 找到的配置速度提升了 24%，这表明 Wayfinder 能够成功找到性能优于默认配置的配置。对于 NPB，性能提升仅为 2%。由于该基准测试主要消耗 CPU 和内存，且不需要任何系统功能，因此操作系统配置对其性能几乎没有影响。



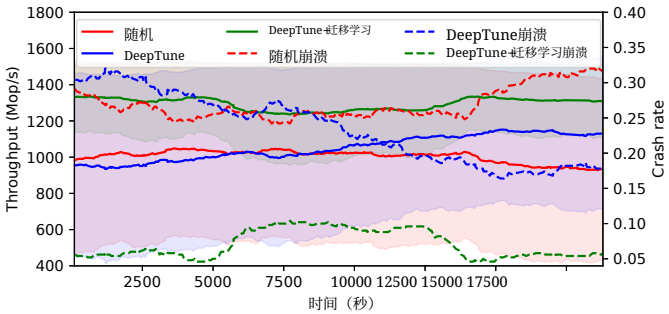
(a) Nginx（性能：吞吐量，**越高越好**）。



(b) Redis（性能：吞吐量，**越高越好**）。



(c) SQLite（性能：操作延迟，**越低越好**）。



(d)NPB（性能：Mop/s，**越高越好**）。

图6. Wayfinder 在 Nginx、Redis、SQLite 和 NPB 上进行 250 次迭代搜索会话后找到的配置的性能演变（实线；为便于阅读，对 5 次运行的结果进行了平滑处理，每条曲线后方的区域表示结果的分布范围）；分别展示了未使用和使用迁移学习（TL，使用 Redis 训练）的情况，以及随机搜索。虚线表示崩溃率。

为 SQLite 找到的最佳配置也未能提升性能，这似乎表明，对于这一场景，默认配置已经非常高效。

图6展示了各方法找到的配置的性能。虚线表示崩溃率，其中 1 表示每次迭代都会发生一次崩溃。搜索过程开始时，Wayfinder 找到的配置性能与随机搜索相近。经过一定次数的迭代后，DeepTune 的神经网络学会了重要参数及其高效用法，使配置性能超越随机搜索。例如，对于 Nginx，经过 250 次迭代后，平滑后的吞吐量比随机搜索高出 20% 以上。与随机搜索相对稳定的崩溃情况不同，Trailblazer 的崩溃率会随着时间推移而下降，因为它逐渐学会避开可能导致崩溃的配置：例如，对于 Nginx，经过 250 次迭代后，崩溃率从 0.3 降至 0.1。

扩展到大型配置空间和长时间搜索过程。在此，我们将 Wayfinder 的可扩展性与竞争对手 Unicorn [38] 进行比较。由于 Unicorn 无法扩展到 Linux 配置的规模，我们创建了一个具有已知局部和全局极大值的合成数据集，以提高收敛的可能性。该数据集的参数总数与 Unicorn 原论文中使用的参数数量相匹配 [38]。我们测量执行时间的变化

表 2. Wayfinder 在经过 250 次迭代后应用于 Linux v4.19 时找到的性能最佳配置。

App.	羽扇豆 Linux	Way-寻路器	性能 unit	相对性能	平均查找时间	
					无迁移学习	TL
Nginx	15731	19593	请求/秒	1.24 倍	415s	92s
Redis	58000	66118	请求/秒	1.14 倍	312s	69s
SQLite	284	284	μ秒/次	1x	248s	76s
NPB	1497	1522	Mop/s	1.02倍	243s	76s

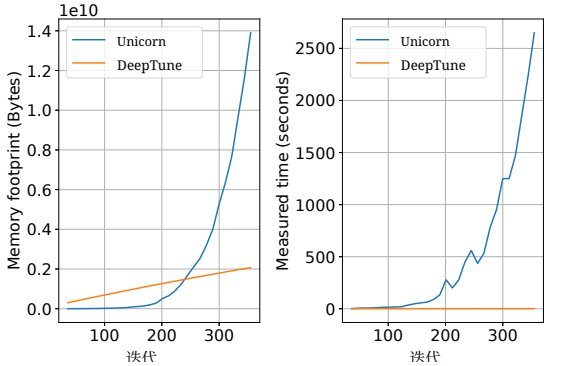


图7. DeepTune 与 Unicorn 在一次搜索过程中的内存消耗（左）和算法执行时间（右）的变化。

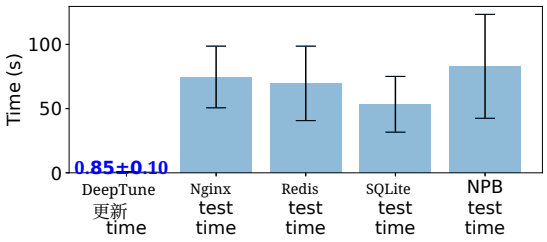


图8. DeepTune 的平均更新时间与不同应用程序的平均测试时间。

以及两种算法每次迭代的峰值内存消耗（使用 Python 的 tracemal- loc [72]）。结果如图7所示。可以看到，随着算法迭代，Unicorn 的执行时间和内存消耗都呈指数级增长。这些限制源于 Unicorn 在设计上选择了因果分析算法，其复杂度通常为 $O(n^4)$，因此不适用于现代操作系统配置的大型搜索空间。相比之下，DeepTune 的时间和内存消耗复杂度均呈线性增长，为 $O(n)$。

由于这些可扩展性问题，我们无法在本次评估介绍的其他实验中与 Unicorn 进行比较。实际上，这些目标设计空间比 Unicorn 能够处理的设计空间大几个数量级。

搜索循环执行时间。 我们测量评估单个配置的平均执行时间，以了解 Wayfinder 的性能。图8展示了执行时间，并将其细分为运行 DeepTune 以决定评估哪种配置所花费的时间，以及评估该配置所花费的时间，即启动内核、启动目标应用程序、运行实验等。评估配置主导着搜索过程：根据应用程序的不同，平均需要 60–80 秒；由于所评估配置带来的性能影响，耗时也会有很大差异。相比之下，DeepTune 一次迭代的执行时间不到一秒，这表明瓶颈在于配置评估，而非搜索算法。

高影响配置参数。我们查询了 DeepTune 学到的模型，以评估 Wayfinder 识别对性能有重大影响的参数的能力，其中包括一些此前已由专家确定的参数。出于篇幅考虑，也因为 Nginx 是许多性能调优指南的目标，我们在此重点讨论 Nginx。

关于对性能有积极影响的主要参数，Trailblazer 不仅能识别调优指南中有详细说明的参数，还能发现指南未提及的其他重要配置选项。例如，它能识别 TCP/IP 协议栈中每个套接字可排队的最大连接数（net.core.somaxconn）、套接字接收缓冲区的默认大小（net.core.rmem_default）以及 TCP 保活时间（net.ipv4.tcp_keepalive_time）。这些参数已在 Nginx/网络性能调优指南中被列为影响显著的参数 [37, 63, 78, 102]。不过，Wayfinder 还能识别那些以不太直观的方式影响性能的参数，例如计算内存统计信息（如 vmstat 报告的数据）的频率（vm.stat_interval）。正如调优指南 [23],所述，降低该频率有助于应对延迟敏感场景。因此，Wayfinder 能自动找出并针对专家认定会影响性能的非显而易见参数进行优化。关于对性能有负面影响的主要参数，Wayfinder 识别出若干会导致性能显著下降的选项，例如将内核日志详细级别（printk）设得过高、延迟输出内核日志（printk_delay），以及启用块 I/O 调试（vm.block_dump）。日志记录和调试对 Nginx 性能的影响广为人知 [39], , 这再次展现了 Wayfinder 识别已有记录的瓶颈的能力。

4.2 迁移学习效率

为了评估迁移学习的效率，我们使用 DeepTune 在 Redis 上训练了一个模型，进行了 250 次迭代（耗时 4.6 小时），并评估该模型在本次评估涉及的其他 3 个应用程序中寻找专用配置的效果。

结果如图6所示，图例中标注为“迁移学习”的曲线展示了找到的配置的性能，以及迁移后模型的崩溃率。与从未经训练的模型开始探索过程相比，使用迁移学习时，配置的性能始终更高：例如，在 Nginx 的探索过程开始时找到的第一个配置，使用迁移学习时的性能比不使用迁移学习或采用随机搜索时高出 1.33× 。使用迁移学习时，崩溃率通常也低得多，在大多数情况下低于 10%。表 2 从另一个角度展示了迁移学习的效率，其倒数两列列出了使用和不使用迁移学习时找到专用配置所需的时间。这项技术带来的时间节省十分显著：搜索速度提升了 4.5× 倍（Nginx）至 3.2× 倍（NPB）。请注意，图6中的曲线是多次运行的平均值，因此，DeepTune+ 迁移学习的曲线并非恰好从图6b中 DeepTune 不使用迁移学习时曲线的终点开始，这并不奇怪。

4.3 崩溃预测

采用随机探索时，大量配置（约三分之一）会在运行时失败，造成了显著的时间浪费。Wayfinder 也会遇到失败，但它能够学习哪些配置参数可能触发崩溃，因此可以避免许多崩溃。

表 3. DeepTune 的基准预测准确率（1 表示 100% 的准确率）。故障准确率和 运行准确率分别表示对故障事件和应用程序运行（非故障）事件的预测准确率。平均绝对误差（MAE）是归一化平均绝对误差。

应用程序	故障准确率	Run 准确率	性能预测 归一化平均绝对误差
Nginx	0.796	0.397	0.273
Redis	0.789	0.310	0.361
SQLite	0.742	0.456	0.112
NPB	0.755	0.455	0.359

表 3 展示了 DeepTune 在预测给定配置是否会失败（故障预测准确率）以及预测给定配置是否能够成功执行（运行预测准确率）时的准确率。表中还展示了预测配置性能时的归一化平均绝对误差。可以看到，故障预测准确率较高，介于 75% 到 80% 之间，因此 Wayfinder 能够避免大量崩溃。运行预测准确率较低，介于 0% 到 36% 之间，因此我们依靠故障预测准确率来估计配置发生故障的概率，并判断是否值得评估该配置。

4.4 不同操作系统与目标指标

将应用程序移植到其他操作系统：***Unikraft***。为了展示 Wayfinder 如何应用于 Linux 以外的操作系统，我们现在将 Wayfinder 应用于 Unikraft [42] 库操作系统。我们编译了一个 Unikraft Nginx 镜像，并使用 Wayfinder 优化其配置，以最大限度地提高请求吞吐量。Unikraft 提供的配置空间远小于 Linux 的配置空间，因此可以与贝叶斯优化进行比较：我们探索了 33 个配置参数（10 个 Nginx 应用级参数和 23 个 Unikraft 操作系统参数），形成了由 3.7×10^{13} 个排列组成的搜索空间。

我们以 3 小时的时间预算启动探索过程。图 9 展示了找到的配置的性能。可以看到，Wayfinder 很快便收敛到一种专用配置，并在 100 分钟后找到该配置。贝叶斯优化则需要 160 多分钟才能找到性能相近的配置。在这一时间预算内，我们还观察到，随机搜索无法找到高性能配置。图 9 清楚地展示了 DeepTune 行为的三个阶段。DeepTune 首先逐步发现性能更好的配置。大约 25 分钟后，它开始采用一组影响显著的参数，并进入利用阶段，专注于这些参数以快速提升性能。

此阶段在约 100 分钟后结束，随后 DeepTune 返回探索阶段。

请注意，最佳配置带来的性能提升显著高于 Linux 中观察到的提升（见图6）。这可以用以下事实解释：Unikraft 是一种单内核 [54]，可提供低延迟的用户态/内核态

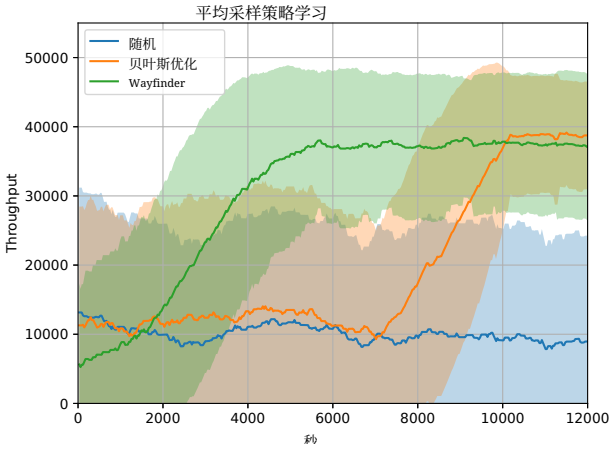


图 9. Wayfinder、随机搜索和贝叶斯优化所找到的配置的性能变化，这些配置用于运行在 Unikraft 单内核上的 Nginx。结果来自 5 次运行，并经过平滑处理以便于阅读。

切换，在配置得当时，能够显著加速对延迟敏感且系统资源密集型的工作负载 [42]。

应用于其他度量：内存占用。 为了评估 Wayfinder 是否适用于优化性能以外的目标，我们测量了它寻找内存占用较小的 Linux 内核配置的效率。过去的多项研究都将这一度量作为目标，因为它对轻量级虚拟化 [44, 73, 80] 或嵌入式系统 [1, 40],等特定领域至关重要，也会影响安全性 [45, 89]等方面。为优化内存消耗，我们将 Wayfinder 配置为构建 RISC-V Linux 的映像。RISC-V 是一种在嵌入式系统中广受欢迎的指令集架构。我们通过在 QEMU 仿真环境中启动映像来测量内存消耗（尽管仿真会影响性能，但不会影响内存消耗）。与该主题以往的研究 [1, 40, 44, 45, 89], 一致，在本实验中，我们将 Wayfinder 配置为优先调整编译时选项（而非 §4.1 中采用的运行时选项）。我们启动 Wayfinder 和随机搜索的探索过程，时间预算为 3 小时。

图 10 展示了找到的配置的内存占用变化情况。默认配置的内存占用为 210 MB。3 小时后，Wayfinder 找到了一种配置，其内存占用为 192 MB（平滑后），相较默认配置减少了 8.5%。探索过程结束时，随机搜索找到的配置的内存占用为 203 MB（平滑后），即减少了 5.5%。此外，可以看出，Wayfinder 的失败配置数量远少于随机搜索。这得益于系统预测崩溃的能力，而这一能力在该实验中尤为有效：最后 100 分钟内仅发生了四次崩溃。

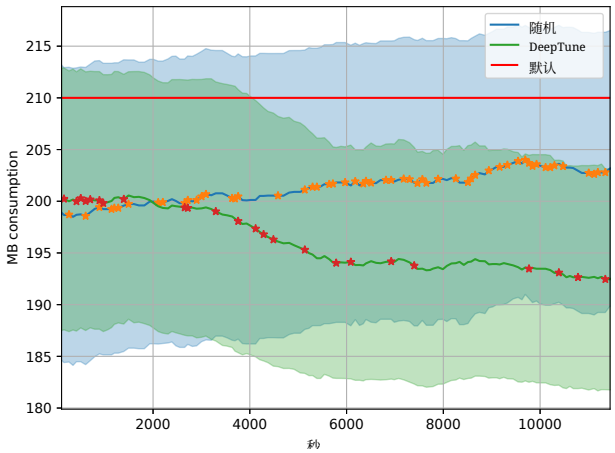


图 10. Wayfinder 和随机搜索在 3 小时搜索会话中找到的配置所构建的 Linux映像的内存占用。展示 5 次运行的结果，并经过平滑处理以便于阅读。崩溃以星号标示。

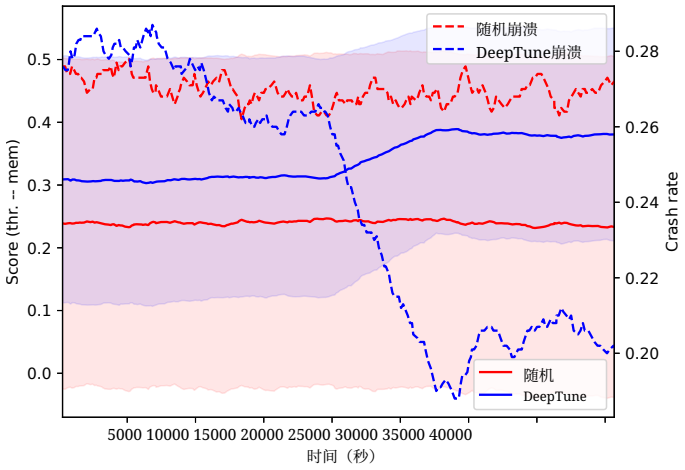


图 11. 基于 Cozart 对吞吐量和内存消耗进行联合优化的学习策略的演变。数值越高越好。测试应用程序为 Nginx。图表经过平滑处理以便于阅读。崩溃率在单独的坐标轴上标示。

与编译时优化器协同工作。 使用前文讨论的方法，Wayfinder 也可以检查编译时参数。然而，由于 Linux 内核管理构建依赖关系的方式以及 Wayfinder 探索配置的方式，这一过程可能效率低下。起初，Wayfinder 生成的配置近乎随机，因此往往需要完全重新构建 Linux 内核。即使只有一个参数设置不正确，也可能导致构建失败。为解决这一问题，我们使用 Cozart [43], 添加了一个初始优化步骤。Cozart 利用动态分析，大幅减少 Linux 内核中的未使用组件。这一减少使配置规模大幅缩小

表 4. 在 Cozart 的基础上，吞吐量-内存实验（图 11）中找到的前 5 项最佳结果。请注意，这些数字是在与 Cozart 论文 [43], 相同的配置下获得的，因此无法与表 2 进行比较（内核版本不同，使用四个 CPU 核心而非一个）。

Rank	得分	内存（mB）	吞吐量（请求/秒）
1	0.84	327.72	47002
2	0.82	328.57	47215
3	0.81	329.67	49375
4	0.79	329.75	48606
5	0.78	330.46	49375
Cozart	–	331.77	46855

空间，使 Wayfinder 能够专注于 Linux 中尚未探索的运行时参数，同时提升性能。我们观察到，与基线相比，吞吐量提高了 31%，内存使用量略有下降，这与 Cozart 评估中报告的情况相似。

接下来，我们在 Cozart 基线上应用 Wayfinder 优化。我们将优化得分定义如下： s

$$s = \text{mXNorm}(t) - \text{mXNorm}(m), \tag{4}$$

其中， t 表示吞吐量， m 表示内存消耗，而 mXNorm(x) 是最小-最大归一化函数，可将吞吐量和内存数值转换到统一的尺度上。因此，得分越高，表示吞吐量越高、内存使用量越少。

图 11 显示，Wayfinder 能够有效学习并创建一种策略，在 Cozart 基线上运行时，其表现优于随机搜索。与图 9 类似，该图展示了 DeepTune 的探索与利用策略。大约 300 分钟后，DeepTune 找到了一组能够提高吞吐量-内存得分的参数。随后，它利用这组参数约 100 分钟。由于 DeepTune 专注于搜索空间中较为稳定的部分，因此崩溃率显著降低。此阶段结束后，DeepTune 切换回探索。这导致崩溃率上升，因为它会探索更多未知配置。

表 4 展示了对吞吐量和内存使用量进行联合探索时获得的前五名得分。尽管第一和第二种排列的吞吐量明显较低，但它们占用的内存更少，因此总得分最高。相比之下，第三种排列的吞吐量大幅提高，而内存使用量仅略有增加。与 Cozart 基线相比，这些排列始终能在保持较低内存消耗的同时，实现更高的吞吐量。

总之，我们展示了，将 Cozart 的编译优化与 Wayfinder 的运行时优化相结合，可以调整操作系统，以提升其在各种指标上的性能。

5 相关工作

基于配置的操作系统专门化。一些研究通过配置 Linux 来实现专门化，以降低资源使用量 [1, 40, 44, 73, 77, 80] 或缩小攻击面 [45, 89]。有些方法需要手动操作 [44, 73, 80]，无法扩展到大量应用程序；另一些则实现了自动化。Undertaker [45, 89]使用动态分析追踪工作负载执行的内核代码，并将这些代码与编译时选项关联起来，从而尽可能多地编译掉不需要的代码。内核定制 [40] 是在 Undertaker 的基础上构建的。研究发现，该工具探索的配置中有很大一部分会在运行时失败，因此提出了一种搜索方法，尝试从已知能够成功执行的配置中推导出有效配置。这些方法旨在缩小攻击面/内存占用，因此目标是确定对于给定工作负载，哪些编译时配置参数可以关闭，哪些必须开启。由于我们关注的是性能（以及资源使用等其他指标），这些方法无法直接用于解决我们的问题。Wayfinder 不仅仅是开启或关闭编译时特性，还会考虑内核完整的编译时、启动时和运行时选项，其中许多选项可以取任意值，且其有效范围没有文档说明。Cozart [43] 同样是在 Undertaker 的基础上构建的。Cozart采用类似的方法，通过动态分析确定未使用的配置选项，从而缩小 Linux 内核的占用空间，同时还能降低内存使用量并提升性能。不过，性能提升并非 Cozart 的主要关注点，但它确实为 Wayfinder 提供了更好的起点。正如前文所述，Wayfinder 与 Cozart 并没有太多共同之处，但在通过编译时选项和运行时选项优化 Linux 内核方面，两者的协同效果很好。AutOS [20]针对人工智能物联网应用程序性能对操作系统内核（包括 Linux）进行专门化：它将配置表示为树，并将其传递给大语言模型，由模型借助描述环境并不断细化搜索的提示词来驱动探索。这项工作的局限在于仅支持编译时选项，而且编写提示词需要专业知识。

基于配置的应用程序优化。Uni-corn [38] 使用因果推断，自动为嵌入式软件定制配置。KML [3, 4]使用机器学习技术来优化存储子系统的配置。一些早期研究通过静态和/或动态分析，探究了软件配置参数对性能的影响 [51, 94, 95]。另一些研究 [18]则专注于为特定工作负载识别对性能最相关的配置参数，本例中的工作负载位于存储子系统中。所有这些系统都需要专家知识来预先识别可能重要的参数集合，而且其目标配置空间远小于 Wayfinder 的配置空间。具体而言，我们已表明（图 7），Uni-corn 无法扩展到如此大的配置空间。

在无服务器计算领域，AWS Lambda Power Tuning [7,19]是一个自动化平台，用于优化分配给函数的内存。Sizeless [24] 提出了一种回归模型，用于估算函数性能。COSE [5] 采用贝叶斯优化，为函数寻找专用配置。在数据库领域，Otter-tune [93] 等多项工作也会自动优化性能。这类特定于应用程序的方法所面对的配置空间，比现代操作系统的配置空间小几个数量级。因此，这些方法依赖贝叶斯优化等技术，而这些技术并不适用于我们的问题。

基于配置的优化也应用于其他领域，包括硬件设计（加速器调优 [62, 101]、FPGA 布局规划 [59]）、神经架构搜索 [97],以及分布式计算放置 [60]。这些方法与其面向的应用领域紧密相关（例如，它们针对搜索空间的特定表示进行操作 [97]），因此无法直接用于操作系统专门化。

机器学习文献中的其他优化工作。近年来，参数搜索/优化受到了广泛关注 [6, 27, 29, 34, 35, 38, 52, 71, 84]。一些方法依赖贝叶斯优化。例如，SMAC [52],使贝叶斯优化能够轻松应用于通用优化问题。SMAC 还支持随机森林，与高斯过程相比，它提升了可扩展性和性能预测能力，但对预测值的后验不确定性估计较差（而这类估计是进行探索与利用决策所必需的）。Snoek 等人 [84] 提出用贝叶斯神经网络（BNN）取代高斯过程。贝叶斯神经网络是通用近似器，能够预测后验不确定性，但其拟合和泛化能力通常低于传统神经网络。贝叶斯神经网络的评估成本也很高，因为它们需要采用蒙特卡洛采样策略，反复运行多次。Hase 等人 [35] 成功将这一方法应用于化学领域，并将其扩展到多目标优化 [34]。Gal 等人 [30] 通过从丢弃法中采样来近似贝叶斯神经网络。最后，Kendall 等人 [41] 研究了机器学习问题所需的几类不确定性，并提出了一种用于回归问题的不确定性估计机制。由于传统激活函数的特性，它们通常无法为离群样本提供良好的不确定性估计。DeepTune 提出了一种新的神经网络设计，能够在单次运行中高精度地预测系统性能、故障和准确率。我们还提出了一种新的评分函数，用以构建均衡的排列策略来进行选择。

6 结论

我们提出了 Wayfinder，这是一种完全自动化的通过配置进行操作系统专门化的方法。Wayfinder 配备了

一个基准测试平台，以及 DeepTune——一种基于神经网络的优化算法，可预测配置的有效性和性能。Wayfinder 无需专家知识，并可扩展到现代操作系统（如 Linux ）的庞大配置空间。针对某个应用程序运行 Wayfinder 并完成优化后，训练好的模型可以重复使用，从而加快对其他类似应用程序进行专门化的速度。Wayfinder 能够针对各种指标找出专用配置；与默认配置相比，应用程序性能最高可提升 24%，内存使用量可降低 8.5%。与采用随机搜索和贝叶斯优化的竞争方法相比，Wayfinder 能更快地收敛到优质解决方案。

致谢

我们感谢匿名评审者提出的意见和见解。特别感谢 Margo Seltzer 提出宝贵意见，显著提升了论文的质量。本研究部分经费来自 Innovate UK 资助项目 10164504（MicrOS），以及英国工程与自然科学研究委员会的资助项目 EP/V012134/1（UniFaaS）、EP/V000225/1（SCorCH）和 EP/X015610/1（FlexCap）。我们感谢加拿大自然科学与工程研究委员会（NSERC）的支持。我们感谢加拿大自然科学与工程研究委员会（CRSNG）提供的支持。

A 制品附录

A.1 摘要

该制品包含 *Wayfinder*（操作系统专用化框架）的源代码，还包含用于重现论文结果的脚本，和/或用于生成论文中图表的数据集。其目的是让读者能够重现实验，并深入了解 *Wayfinder* 的功能和用法。

A.2 描述与要求

A.2.1 如何获取。最新版本的制品可在 GitHub[2] 上找到。或者，也可以从我们的 Zenodo 存档[3]下载各个版本。请注意，制品评估 (AE) GitHub 仓库仅包含制品的一部分，即用于重现本文实验的脚本。Wayfinder 框架代码、工具和其他示例可在 Wayfinder[4] 仓库中找到。

为了精确复现本文的测量结果，我们向 EuroSys'26 AEC 评审提供了服务器的访问权限。该服务器配备 Intel(R) Xeon(R) CPU E5-2690 v3 @ 2.60GHz、315 GB 内存，运行 Ubuntu 24.04 和 Linux 6.8.0-53-generic 版本。运行制品并不需要访问这一特定配置。硬件和软件依赖项详见下文。

A.2.2 硬件依赖。一颗 Intel(R) Xeon(R) CPU E5-2690 v3 @ 2.60 GHz 处理器，配备至少 64 GB 内存；通常，任何具有双路插槽的处理器都符合要求。处理器必须至少有 8 个处理器核心。运行对应图7的实验需要 64 GB 内存，因为该实验会测试内存用量。请注意，重现本文结果需要配备这么多处理器核心和内存，而运行 Wayfinder 则不需要。至少需要 100 GB 磁盘空间。

A.2.3 软件依赖。 该制品已在 Ubuntu 24.04（Noble Numbat）上通过测试，所用 Linux 内核版本为 6.8.0-53-generic（已启用 KVM）、Docker 版本为 28.1.1（或任何较新的版本）、QEMU 5.2 和 Libvirt 7.4。其他依赖项使用容器，这些容器已由 Wayfinder 和制品脚本进行版本管理。

A.2.4 基准测试。 所有数据集均包含在该制品中，或由脚本自动生成，或预先生成。

A.3 设置

运行任何实验之前，请按照上文 A.2.3 中的详细建议准备好主机。系统设置完成后，克隆 AE 仓库：

```
$ git clone \
https://github.com/unikraft/wayfinder-eurosys26-ae
```

然后按照*README.md* 中的逐步说明，从 Wayfinder 仓库下载、编译并配置 Wayfinder：

```
$ git clone https://github.com/unikraft/wayfinder
```

最后，完成所有配置后，启动 Wayfinder 及其辅助容器：

```
$ docker-compose up -d registry influxdb
postgres redis $ sudo ./脚本/wayfinder.sh
```

A.4 评估工作流程[5]

实验应按顺序运行。每个实验都会使用不同的构建器镜像来生成待测试的内核。这些镜像需要在运行实验之前构建，例如：

```
$ docker build -t localhost:5000/linux-nginx:latest -
f Dockerfile.jessie . $ docker push
localhost:5000/linux-nginx:latest
```

[2]https://github.com/unikraft/wayfinder-eurosys26-ae [3] https://doi.org/10 18598079 45281./zenodo. https://github.com/unikraft/wayfinder 和 https://doi.org/10.5281/zenodo.18592520

[5]评估所遵循的提交、评审和徽章授予方法详见 https://sysartifacts.github.io/eurosys2026/。

每项实验的准备工作都是必需步骤，包括构建平台将要测试的内核。每次内核构建操作只需几分钟，并且应在系统上没有其他实验运行时进行。准备好某项实验后，即可使用类似以下的语法运行：

```
$ ~/wayfinder/dist/wfctl create ./作业.yaml $ ~/wayfinder/dist/wfctl start --isol-level full --isol-split both -l 1 -s 随机 "$ID"
```

在我们的配置下，运行所有实验平均需要数天。这是因为我们收集了数千种排列来展示演变过程。对于与 Wayfinder 交互的实验，可以考虑减少排列和重复次数。对于其余实验，我们提供了预生成数据集，可用于快速生成图表。

A.4.1 主要主张。

- *(C1)*：*Wayfinder* 可全自动识别专用配置，与默认配置相比，应用程序性能最高可提升 *24%*，内存使用量可减少 *8.5%*。第 *4.1* 节和第 *4.4* 节所述实验及其结果证明了这一点，结果见图*6*和图 *10*。

- *(C2)*：我们强调了神经网络的优势：它比竞争方法（随机和贝叶斯）更快地找到优质解决方案，并能成功在相关应用程序之间迁移知识。第 *4.4* 节所述实验及其结果证明了这一点，结果见图 *9* 和图 *11*。

A.4.2 实验。第 A.3 节提到的制品仓库中展示和组织的每项实验都有一个 *README.md* 文件，并以 *figure-** 和 *table-**格式的目录进行结构化组织。这些文件简要介绍了实验以及相关的准备和执行步骤：强烈建议至少通读一遍。没有注明时间估算的实验需要几分钟。运行仓库中的实验总共大约需要 10 到 12 小时；如果进行重复实验，则最多可能需要 32 小时。在这两种情况下，其中只有 2 到 3 小时需要手动操作。Wayfinder 框架会负责在后台运行作业。

在同一台机器上重现实验，应能得到与论文中相近的结果。在其他机器上，我们预计绝对数值会有所不同，但趋势和排序应相似。由于探索过程具有非确定性，重复测量会存在一定差异，但多次运行的总体趋势和平均值应与论文中呈现的结果一致。图1、图2和表1可通过手动方式重现。由于生成这些图表所需的数据要经过非常漫长的探索时间（数天），图5、图6、图7、图8、图9、图10、图11可利用现有数据集重现。最后，数据中的测量使用以下指标：*requests/s*、*ms/op*、*Mop/s*、秒、*MB*。

A.5 可复用性说明

审阅者可以使用 Wayfinder代码仓库中的示例，基于自定义应用程序及其配套的基准测试工具创建自己的作业。构建 Wayfinder开发者Docker镜像、运行通用作业和构建自定义镜像的说明，可在我们 AE 仓库根目录下的 *README.md* 文件中找到[6]。

A.6 一般说明

所有实验都附有*README.md*文件，其中详细说明了获取相应图表所需的手动步骤。图1、表1和图2对应的文件夹中包含从头开始生成它们的逐步指南。其他实验文件夹则包含已获取的数据集，以及用于生成相关图表/表格的脚本。大多数实验每项至少需要运行一天，因此建议检查并调整所提供的脚本，以减少迭代次数。我们强烈建议在开始复现实验之前，仔细阅读每项实验中的说明。

参考资料

[1] Mathieu Acher、Hugo Martin、Juliana Alves Pereira、Arnaud Blouin、Jean-Marc Jézéquel、Djamel Eddine Khelladi、Luc Lesoil 和 Olivier Barais。2019年。学习超大配置空间：*Linux* 内核大小的关键影响因素。博士学位论文。Inria Rennes-Bretagne Atlantique。[2] Alexandru Agache、Marc Brooker、Alexandra Iordache、Anthony Liguori、Rolf Neugebauer、Phil Piwonka 和 Diana-Maria Popa。2020年。Firecracker：面向无服务器应用程序的轻量级虚拟化。载于*NSDI*，第 20 卷。419–434。[3] Ibrahim Umit Akgun、Ali Selman Aydin、Andrew Burford、Michael McNeill、Michael Arkhangelskiy 和 Erez Zadok。2023年。利用机器学习改进存储系统。*ACM*存储汇刊 19，1（2023年），1–30。[4] Ibrahim Umit Akgun、Ali Selman Aydin、Aadil Shaikh、Lukas Velikov 和 Erez Zadok。2021年。用于提升存储系统性能的机器学习框架。载于第 *13* 届 *ACM*存储与文件系统热点问题研讨会论文集。94–102。[5] Nabeel Akhtar、Ali Raza、Vatche Ishakian 和 Ibrahim Matta。2020年。Cose：利用统计学习配置无服务器函数。载于*IEEE INFOCOM 2020-IEEE*计算机通信会议。IEEE，129–138。[6] Takuya Akiba、Shotaro Sano、Toshihiko Yanase、Takeru Ohta 和 Masanori Koyama。2019年。Optuna：新一代超参数优化框架。载于第 *25* 届 *ACM SIGKDD*知识发现与数据挖掘国际会议论文集。2623–2631。[7] 亚马逊云科技。2026年。使用 AWS Lambda Power Tuning 对函数进行性能分析。https://docs.aws.amazon.com/lambda/latest/operatorguide/profile-functions.html [访问日期：2 月 11, 2026]。

[6]https://github.com/unikraft/wayfinder-eurosys26-ae/blob/main/README.md

[8] Thomas E Anderson。1993年。应用专用操作系统的必要性。加州大学伯克利分校，计算机科学系。[9] Sergei Arnautov、Bohdan Trach、Franz Gregor、Thomas Knauth、Andre Martin、Christian Priebe、Joshua Lind、Divya Muthukumaran、Dan O' keeffe、Mark Stillwell 等。2016年。Scone：基于 Intel SGX 的安全 Linux 容器。载于 OSDI，第 16 卷。689–703。[10] Risan Bagja Pradana。2017年。优化 Nginx 配置。https: // github.com/risan/nginx-config [访问日期：2 月 11, 2026]。[11] David Bailey、Tim Harris、William Saphir、Rob Van Der Wijngaart、Alex Woo 和 Maurice Yarrow。1995年。NAS 并行基准测试 2.0。技术报告。技术报告 NAS-95-020，NASA 艾姆斯研究中心。[12] Harry G. Barrow、Jay M. Tenenbaum、Robert C. Bolles 和 Helen C. Wolf。1977年。参数对应与倒角匹配：两种新的图像匹配技术。载于第 5 届国际人工智能联合会议论文集（IJCAI'77，第 2 卷）。[13] Andrew Baumann、Paul Barham、Pierre-Evariste Dagand、Tim Harris、Rebecca Isaacs、Simon Peter、Timothy Roscoe、Adrian Schüpbach 和 Akhilesh Singhania。2009年。多内核：面向可扩展多核系统的新型操作系统架构。载于第 22 届 ACM SIGOPS 操作系统原理研讨会论文集（SOSP'09）。ACM，29–44。doi:10.1145/1629575.1629579[14] Andrew Baumann、Dongyoon Lee、Pedro Fonseca、Lisa Glendenning、Jacob R Lorch、Barry Bond、Reuben Olinsky 和 Galen C Hunt。2013年。利用 Bascule 安全、高效地组合操作系统扩展。载于第 8 届 ACM 欧洲计算机系统会议论文集。239–252。[15] Adam Belay、George Prekas、Ana Klimovic、Samuel Grossman、Christos Kozyrakis 和 Edouard Bugnion。2014年。IX：面向高吞吐量和低延迟的受保护数据平面操作系统。载于第 11 届 USENIX 操作系统设计与实现研讨会论文集（OSDI'14）。USENIX，49–65。[16] Vincent Bernat。2011年。调优 Linux IPv4 路由缓存。https:// vincent.bernat.ch/en/blog/2011-ipv4-route-cache-linux [访问日期：2 月 11, 2026]。[17] Leo Breiman。2001年。随机森林。机器学习（2001年）。[18] Zhen Cao、Geoff Kuenning 和 Erez Zadok。2020年。Carver：为存储系统调优找出重要参数。载于第 18 届 USENIX 文件与存储技术会议（FAST 20）。43–57。[19] Alex Casalboni。2023年。AWS Lambda Power Tuning GitHub 代码库。https://github.com/alexcasalboni/aws-lambda-power-tuning [访问日期：2 月 11, 2026]。[20] Huilai Chen、Yuanbo Wen、Limin Cheng、Shouxu Kuang、Yumeng Liu、Weijia Li、Ling Li、Rui Zhang、Xinkai Song、Wei Li 等。2024年。（海报）AutoOS：利用大语言模型让操作系统更强大。载于第四十一届国际机器学习会议。[21] Bob Cromwell。2023年。Linux 性能调优——TCP。https: // cromwell-intl.com/open-source/performance-tuning/tcp.html [访问日期：2 月 11, 2026]。[22] Abhishek Dubey。2019年。Redis 最佳实践与性能调优。https://iamabhishek-dubey.medium.com/redis-best-practices-and-performance-tuning-c48611388bbc [访问日期：2 月 11, 2026]。[23] Jeremy Eder。2015年。Red Hat Enterprise Linux 7 低延迟性能调优。https://access.redhat.com/sites/default/files/ attach ments/201501-perf-brief-low-latency-tuning-rhel7-v1.1.pdf [访问日期：2 月 11, 2026]。[24] Simon Eismann、Long Bui、Johannes Grohmann、Cristina Abad、Nikolas Herbst 和 Samuel Kounev。2021年。Sizeless：预测无服务器函数的最佳大小。载于第 22 届 Middleware 会议论文集。248–259。[25] Dawson R. Engler、M. Frans Kaashoek 和 James O'Toole Jr.。1995年。外核：一种面向应用级的操作系统架构资源管理。载于《第15届ACM操作系统原理研讨会》（SOSP'95）论文集。ACM，251–266。doi:10.1145/ 224056.224076[26] Haoqiang Fan、Hao Su 和 Leonidas J. Guibas。2017年。用于从单幅图像进行三维物体重建的点集生成网络。载于 IEEE 计算机视觉与模式识别会议（CVPR）论文集。[27] Matthias Feurer、Aaron Klein、Katharina Eggensperger、Jost Sprin- genberg、Manuel Blum 和 Frank Hutter。2015年。高效且稳健的自动化机器学习。《神经信息处理系统进展》28（2015年）。[28] Mike Freemon。2022年。在保持低延迟的同时优化 TCP，以实现高广域网吞吐量。https://blog.cloudflare.com/ optimizing-tcp-for-high-throughput-and-low-latency/ [访问日期：2月。 11, 2026]。[29] Pascal Friederich、Florian Häse、Jonny Proppe 和 Alán Aspuru- Guzik。2021年。用于新一代物质模拟的机器学习势函数。《自然·材料》（2021年）。[30] Yarin Gal 和 Zoubin Ghahramani。2016年。将丢弃法视为贝叶斯近似：表示深度学习中的模型不确定性。载于国际机器学习会议。[31] Eduardo C. Garrido-Merchán 和 Daniel Hernández-Lobato。2020年。利用高斯过程在贝叶斯优化中处理类别变量和整数值变量。《神经计算》380（2020年），20 35 10.1016 . .2019.11.004 – 。doi: /j [32] 2026 neucom 谷歌。 Google Syzkaller：系统调用描述语言。https://github.com/google/syzkaller/blob/master/docs/ syscall_ descriptions_syntax.md [访问日期：2月。 11, 2026]。[33] Per Brinch Hansen。1970年。多道程序系统内核。《ACM通讯》13，4（1970年），238–241。[34] Florian Häse、Loïc M Roch 和 Alán Aspuru-Guzik。2018年。Chimera：为自驱动实验室实现基于层次结构的多目标优化。《化学科学》（2018年）。[35] Florian Häse、Loïc M. Roch、Christoph Kreisbeck 和 Alán Aspuru- Guzik。2018年。Phoenics：用于化学的贝叶斯优化器。《美国化学会中心科学》（2018年）。[36] Asmaul Hosna、Ethel Merry、Jigmey Gyalmo、Zulfikar Alom、Ze- yar Aung 和 Mohammad Abdul Azim。2022年。迁移学习：通俗易懂的介绍。《大数据期刊》9，1（2022年），102。[37] IBM。2021年。调优 Linux 系统，以提升并行作业性能。https://www.ibm.com/docs/en/smpi/10.2?topic=mpi-tuning-your-linux-system [访问日期：2月。 11, 2026]。[38] Md Shahriar Iqbal、Rahul Krishna、Mohammad Ali Javidian、Baishakhi Ray 和 Pooyan Jamshidi。2022年。Unicorn：从因果关系的视角推理可配置系统的性能。载于《第十七届欧洲计算机系统会议》（法国雷恩）（EuroSys '22）论文集。计算机协会，美国纽约州纽约市，199–217。doi:10.1145/3492321.3519575[39] Alexander Jung、Hugo Lefeuvre、Charalampos Rotsos、Pierre Olivier、Daniel Oñoro-Rubio、Felipe Huici 和 Mathias Niepert。2021年。Wayfinder：朝着自动推导最优操作系统配置迈进。载于《第12届 ACM SIGOPS亚太系统研讨会》论文集。115–122。[40] Junghwan Kang。2017年。Linux 内核定制的实用方法。Linux基金会北美开源峰会，加利福尼亚州洛杉矶（2017年）。[41] Alex Kendall 和 Yarin Gal。2017年。贝叶斯深度学习需要哪些不确定性来支持计算机视觉? 载于《神经信息处理系统进展》，I. Guyon、U. Von Luxburg、S. Bengio、H. Wallach、R. Fergus、S. Vishwanathan 和 R. Garnett（编），第30卷。Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/ paper/2017/file/2650d6089a6d640c5e85b2b88265dc2b-Paper.pdf [42] Simon Kuenzer、Vlad-Andrei Bădoiu、Hugo Lefeuvre、Sharan San- thanam、Alexander Jung、Gaulthier Gain、Cyril Soldani、Costin Lupu,

Ştefan Teodorescu、Costi Răducanu、Cristian Banu、Laurent Mathy、Răzvan Deaconescu、Costin Raiciu 和 Felipe Huici。2021年。Unikraft：轻松构建快速、专用的单内核。第十六届欧洲计算机系统会议。doi: 10.1145 3447786.3456248 / Hsuan-Chi [43] Kuo、Jianyan Chen、Sibin Mohan 和 Tianyin Xu。2020年。为操作系统核心设定配置：操作系统内核精简的实用性。ACM测量与分析计算系统汇刊。 4, 1, 03 2020 27 3379469 文章（5月），页 [44] 数。doi:10.1145/ Hsuan-Chi Kuo、Dan Williams、Ricardo Koller 和 Sibin Mohan。2020年。披着单内核外衣的 Linux。载于《第十五届欧洲计算机系统会议》（希腊伊拉克利翁）（EuroSys '20）论文集。计算机协会，美国纽约州纽约市，15页。doi:10.1145/3342195.3387526[45] Anil Kurmus、Reinhard Tartler、Daniela Dorneanu、Bernhard Heinloth、Valentin Rothberg、Andreas Ruprecht、Wolfgang Schröder-Preikschat、Daniel Lohmann 和 Rüdiger Kapitza。2013年。攻击面指标与编译时操作系统内核定制。载于NDSS。[46] Butler W Lampson 和 Robert F Sproull。1979年。面向单用户机器的开放式操作系统。载于第七届 ACM 操作系统原理研讨会论文集。98–105。[47] Chien-Cheng Lee、Pau-Choo Chung、Jea-Rong Tsai 和 Chein-I Chang。1999年。稳健的径向基函数神经网络。《IEEE系统、人类与控制论汇刊B辑》（控制论） 29, 6 1999 674 685 10.1109 3477.809023 （），–。doi: / [48] Hugo Lefeuvre、Vlad-Andrei Bădoiu、Ştefan Teodorescu、Pierre Olivier、Tiberiu Mosnoi、Răzvan Deaconescu、Felipe Huici 和 Costin Raiciu。2021年。Flexos：让操作系统隔离更加灵活。载于《操作系统热点问题研讨会》论文集。79–87。[49] Hugo Lefeuvre、Vlad-Andrei Bădoiu、Alexander Jung、Stefan Lucian Teodorescu、Sebastian Rauch、Felipe Huici、Costin Raiciu 和 Pierre Olivier。2022年。FlexOS：迈向灵活的操作系统隔离。载于第27届 ACM 编程语言与操作系统体系结构支持国际会议（瑞士洛桑）（ASPLOS '22）论文集。计算机协会，美国纽约州纽约市，467–482。doi:10.1145/3503222.3507759[50] LevelDB贡献者。2020年。LevelDB 的 SQLite3基准测试。 https://github.com/google/leveldb/blob/main/benchmarks/ db_bench_ sqlite3.cc [访问日期：2月。 11, 2026]。[51] Chi Li、Shu Wang、Henry Hoffmann 和 Shan Lu。2020年。静态推断软件配置的性能属性。载于《第十五届欧洲计算机系统会议》论文集。1–16。[52] Marius Lindauer、Katharina Eggensperger、Matthias Feurer、André Biedenkapp、Difan Deng、Carolin Benjamins、Tim Ruhkopf、René Sass 和 Frank Hutter。2022年。SMAC3：用于超参数优化的多功能贝叶斯优化软件包。《机器学习研究期刊》（2022年）。[53] Linux贡献者。2023年。内核的命令行参数。 h t t p s:// www.kernel.org/doc/html/latest/admin-guide/kernel-parameters.html。[54] Anil Madhavapeddy、Richard Mortier、Charalampos Rotsos、David Scott、Balraj Singh、Thomas Gazagnaire、Steven Smith、Steven Hand 和 Jon Crowcroft。2013年。单内核：面向云的库操作系统。ACM SIGARCH计算机体系结构新闻41，1（2013年），461–472。[55] Filipe Manco、Costin Lupu、Florian Schmidt、Jose Mendes、Simon Kuenzer、Sumit Sati、Kenichi Yasukata、Costin Raiciu 和 Felipe Huici。2017年。我的虚拟机比你的容器更轻量（也更安全）。载于《第26届操作系统原理研讨会》论文集。218–233。[56] Anqi Mao、Mehryar Mohri 和 Yutao Zhong。2023年。交叉熵损失函数：理论分析与应用。载于《第40届国际机器学习会议》（美国夏威夷州檀香山）（ICML'23）论文集。JMLR.org，文章编号992，共26页。[57] Ilias Marinos、Robert N.M. Watson 和 Mark Handley。2014年。面向性能的网络栈专用化。《SIGCOMM 计算机通信评论》 44, 4 2014 175 186 10.1145 2740070.2626311 （8月），–。[58] 2018 2 doi：/ Patrick Meenan。优化采用 BBR 和 tcp_notsent_lowat 的 HTTP/ 优先级处理。 https://blog.cloudflare.com/http-2-prioritization-with-nginx/ [访问日期：2月 11, 2026]。[59] Azalia Mirhoseini、Anna Goldie、Mustafa Yazgan、Joe Wenjie Jiang、Ebrahim Songhori、Shen Wang、Young-Joon Lee、Eric Johnson、Omkar Pathak、Azade Nazi 等。2021年。一种用于快速芯片设计的图布局方法。《自然》594，7862（2021年），207–212。[60] Azalia Mirhoseini、Hieu Pham、Quoc V Le、Benoit Steiner、Rasmus Larsen、Yuefeng Zhou、Naveen Kumar、Mohammad Norouzi、Samy Bengio 和 Jeff Dean。2017年。通过强化学习优化设备布局。载于《国际机器学习会议》。PMLR，2430–2439。[61] Sparsh Mittal。2016年。非对称多核处理器架构设计与管理技术综述。《ACM计算调查》（CSUR）48，3（2016年），1–38。[62] Luigi Nardi、David Koeplinger 和 Kunle Olukotun。2019年。实用的设计空间探索。载于《2019年 IEEE 第27届计算机与电信系统建模、分析与仿真国际研讨会》（MASCOTS）。IEEE，347–358。[63] Rick Nelson。2014年。调优 NGINX 以提升性能。 https://www.f5.com/company/blog/nginx/tuning-nginx [11, 访问日期：2026] 2月 。[64] Nginx 贡献者。2023年。Nginx 网站。https://nginx.org/ [访问日期：2月 11, 2026]。[65] Ruslan Nikolaev、Mincheol Sung 和 Binoy Ravindran。2020年。LibrettOS：一种可动态适配的多服务器库操作系统。载于《第16届 ACM SIGPLAN/SIGOPS 虚拟执行环境国际会议论文集》。114–128。[66] Pierre Olivier、Daniel Chiba、Stefan Lankes、Changwoo Min 和 Binoy Ravindran。2019年。一种二进制兼容的单内核。载于《第15届 ACM SIGPLAN/SIGOPS虚拟执行环境国际会议论文集》（VEE）（VEE'19）。ACM，59–73。[67] Pierre Olivier、Hugo Lefeuvre、Daniel Chiba、Stefan Lankes、Changwoo Min 和 Binoy Ravindran。2021年。一种系统调用级别的二进制兼容单内核。《IEEE 计算机汇刊》71，9（2021年），2116–2127。[68] Pierre Olivier、A. K. M. Fazla Mehrab、Stefan Lankes、Mohamed Lamine Karaoui、Robert Lyerly 和 Binoy Ravindran。2019年。HEXO：在低成本、低功耗嵌入式系统上卸载高性能计算密集型工作负载。载于《第28届高性能并行与分布式计算国际研讨会论文集》（美国亚利桑那州凤凰城）（HPDC '19）。计算机协会，美国纽约州纽约市， 85 96 10.1145 3307681.3325408 –。[69] doi：/ Meni Orenbach、Pavel Lifshits、Marina Minkin 和 Mark Silberstein。2017年。Eleos：面向 SGX 飞地的无退出操作系统服务。载于《第十二届欧洲计算机系统会议论文集》。238–253。[70] Simon Peter、Jialin Li、Irene Zhang、Dan R. K. Ports、Doug Woos、Arvind Krishnamurthy、Thomas Anderson 和 Timothy Roscoe。2015年。Arrakis：操作系统即控制平面。《ACM 计算机系统汇刊》33，4（2015年11月），1–30。doi:10.1145/2812806[71] Robert Pollice、Gabriel dos Passos Gomes、Matteo Aldeghi、Riley J Hickman、Mario Krenn、Cyrille Lavigne、Michael Lindner-D'Addario、AkshatKumar Nigam、Cher Tian Ser、Zhenpeng Yao 等。2021年。加速材料设计的数据驱动策略。《Accounts of Chemical Research》（2021年）。[72] Python 贡献者。2023年。tracemalloc - 跟踪内存分配。 https://docs.python.org/3/library/tracemalloc.html [访问日期：2月 11, 2026]。[73] Ali Raza、Thomas Unger、Matthew Boyd、Eric B Munson、Parul Sohal、Ulrich Drepper、Richard Jones、Daniel Bristot De Oliveira、Larry Woodman、Renato Mancuso 等。2023年。单内核 Linux（UKL）。

载于《第十八届欧洲计算机系统会议论文集》。590–605。[74] Redis 贡献者。2023年。Redis 网站。https://redis.io/ [访问日期：2月 11, 2026]。[75] Redis 贡献者。2025年。Redis 基准测试工具。https://github.com/redis/redis/blob/unstable/src/redis-benchmark.c [访问日期：2月 11, 2026]。[76] Mark Richards。2021年。极限 HTTP 性能调优：在 4 个虚拟 CPU 的 EC2 实例上实现 120 万 API 请求数/秒。https://talawah.io/blog/extreme-http-performance-tuning-one-point-two-million/ [访问日期：2月 11, 2026]。[77] Andreas Ruprecht、Bernhard Heinloth 和 Daniel Lohmann。2014年。大规模系统软件产品线中的自动特征选择。《ACM SIGPLAN 通知》50，3（2014年），39–48。[78] Emily Serverwise。2022年。最大化 NGINX 性能：调优积压队列和 net.core.somaxconn 参数的综合指南。https://www.getpagespeed.com/server-setup/nginx/maximizing-performance-a-comprehensive-guide-to-tuning-the-backlog-and-net-core-somaxconn-parameters [访问日期：2月 11, 2026]。[79] Bobak Shahriari、Kevin Swersky、Ziyu Wang、Ryan P Adams 和 Nando De Freitas。2015年。让人类退出循环：贝叶斯优化综述。《IEEE 会议录》104，1（2015年），148–175。[80] Zhiming Shen、Zhen Sun、Gur-Eyal Sela、Eugene Bagdasaryan、Christina Delimitrou、Robbert Van Renesse 和 Hakim Weatherspoon。2019年。X-containers：打破壁垒，提升云原生容器的性能与隔离性。载于《第二十四届编程语言与操作系统架构支持国际会议论文集》。121–135。[81] Mark Silberstein。2017年。OmniX：面向全可编程系统的以加速器为中心的操作系统。载于《第十六届操作系统热点研讨会论文集》。69–75。[82] Mark Silberstein、Bryan Ford、Idit Keidar 和 Emmett Witchel。2013年。GPUfs：将文件系统与 GPU 集成。载于《第十八届编程语言与操作系统架构支持国际会议论文集》。485–498。[83] Mark Silberstein、Sangman Kim、Seonggu Huh、Xinya Zhang、Yige Hu、Amir Wated 和 Emmett Witchel。2016年。GPUnet：面向 GPU 程序的网络抽象。《ACM 计算机系统汇刊》（TOCS）34，3（2016年），1–31。[84] Jasper Snoek、Oren Rippel、Kevin Swersky、Ryan Kiros、Nadathur Satish、Narayanan Sundaram、Md. Mostofa Ali Patwary、Prabhat 和 Ryan P. Adams。2015年。使用深度神经网络实现可扩展的贝叶斯优化。载于 ICML。[85] SQLite 贡献者。2023年。SQLite 网站。https://www.sqlite.org/index.html [访问日期：2月 11, 2026]。[86] Nitish Srivastava、Geoffrey Hinton、Alex Krizhevsky、Ilya Sutskever 和 Ruslan Salakhutdinov。2014年。丢弃法：一种防止神经网络过拟合的简单方法。《机器学习研究期刊》（2014年）。[87] Timo Stark。2022年。避免 NGINX 配置中的十大错误。https://www.f5.com/company/blog/nginx/avoiding-top-10-nginx-configuration-mistakes [访问日期：2月 11, 2026]。[88] Jacob Swanson、Myra B. Cohen、Matthew B. Dwyer、Brady J. Garvin 和 Justin Firestone。2014年。超越彩虹：可配置系统中的自适应故障规避。载于《第22届 ACM SIGSOFT 软件工程基础国际研讨会论文集》（中国香港）（FSE 2014）。计算机协会，美国纽约州纽约市，377–388。doi:10.1145/2635868.2635915[89] Reinhard Tartler、Anil Kurmus、Bernhard Heinloth、Valentin Rothberg、Andreas Ruprecht、Daniela Dorneanu、Rüdiger Kapitza、Wolfgang Schröder-Preikschat 和 Daniel Lohmann。2012年。利用编译时可配置性自动缩减操作系统内核的 TCB。载于系统可靠性热点研讨会第八届会议（*HotDep 12*）。[90] Chia-Che Tsai、Kumar Saurabh Arora、Nehal Bandi、Bhushan Jain、William Jannen、Jitin John、Harry A Kalodner、Vrushali Kulkarni、Daniela Oliveira 和 Donald E Porter。2014年。面向多进程应用的库操作系统协作与安全隔离。载于 第九届欧洲计算机系统会议论文集。1–14。[91] Chia-Che Tsai、Donald E Porter 和 Mona Vij。2017年。Graphene-SGX：面向 SGX 上未经修改应用程序的实用库操作系统。载于 *USENIX* 年度技术会议。645–658。[92] Pavel Valov、Jean-Christophe Petkovich、Jianmei Guo、Sebastian Fis-chmeister 和 Krzysztof Czarnecki。2017年。在不同硬件平台之间迁移性能预测模型。载于 第八届 *ACM/SPEC* 国际性能工程会议论文集。39–50。[93] Dana Van Aken、Andrew Pavlo、Geoffrey J Gordon 和 Bohan Zhang。2017年。通过大规模机器学习实现数据库管理系统自动调优。载于 *2017* 年 *ACM* 国际数据管理会议论文集。1009–1024。[94] Miguel Velez、Pooyan Jamshidi、Florian Sattler、Norbert Siegmund、Sven Apel 和 Christian Kästner。2020年。Configcrusher：迈向可配置系统的白盒性能分析。自动化软件工程 27（2020），265–300。[95] Miguel Velez、Pooyan Jamshidi、Norbert Siegmund、Sven Apel 和 Christian Kästner。2021年。机器学习上的白盒分析：对可配置系统性能进行建模。载于*2021 IEEE/ACM* 第 *43* 届软件工程国际会议（*ICSE*）。IEEE，1072–1084。[96] Yu Wang、Victor Lee、Gu-Yeon Wei 和 David Brooks。2019年。通过分析公开数据集预测新工作负载或 CPU 性能。*ACM* 架构、代码与优化汇刊 15，4，文章 53（2019 年 1 月），21 页。doi:10.1145/3284127[97] Colin White、Mahmoud Safari、Rhea Sukthanker、Binxin Ru、Thomas Elsken、Arber Zela、Debadeepta Dey 和 Frank Hutter。2023年。神经架构搜索：来自 1000 篇论文的洞见。*arXiv* 预印本 *arXiv:2301.08727*（2023）。[98] Marcel Wienöbst、Max Bannach 和 Maciej Liśkiewicz。2021年。因果图模型的可扩展性：算法与计算复杂性。载于 第三十七届人工智能不确定性会议论文集。PMLR。[99] wrk 贡献者。2021年。wrk：现代 HTTP 基准测试工具。https://github.com/wg/wrk [访问日期：2 月 11, 2026]。[100] Zelda B Zabinsky。2009年。随机搜索算法。美国华盛顿大学工业与系统工程系 （2009）。[101] Dan Zhang、Safeen Huda、Ebrahim Songhori、Kartik Prabhu、Quoc Le、Anna Goldie 和 Azalia Mirhoseini。2022年。面向领域优化深度学习加速器的全栈搜索技术。载于 第 *27* 届 *ACM* 编程语言与操作系统架构支持国际会议论文集。27–42。[102] Mike Zupan。2014年。繁忙服务器推荐的 TCP 保活设置。https://serverfault.com/questions/641606/recommended-tcp-keepalive-settings-for-a-busy-server [访问日期：2 月 11, 2026]。