arXiv:2611.31983v1 [cs.AI] 24 Nov 2026

探索带宽高效预取器以缩减缓存层次结构

Patrick Tser Jern Kon*
konp@google.com
Google
USA

Akanksha Jain*
avjain@google.com
Google
USA

Abraham Gonzalez
abegonzalez@google.com
Google
USA

Rui Meng
rmeng@google.com
Google
USA

Dilara Soylu
soylu@stanford.edu
斯坦福大学
USA

Shubham Agarwal
shubham3@berkeley.edu
加州大学伯克利分校
USA

Yawen Wang
yawenw@google.com
Google
USA

Iman Hosseini
imanhosseini@google.com
Google
USA

Hanna Alam
hannaalam@google.com
Google
USA

Jinsung Yoon
jinsungyoon@google.com
Google
USA

Tomas Pfister
tpfister@google.com
Google
USA

Nanyun Peng
violetpeng@google.com
Google
USA

Chun-Liang Li
chunliang@google.com
Google
USA

Parthasarathy Ranganathan
parthas@google.com
Google
USA

摘要

共享末级缓存（LLC）日益占据服务器处理器芯片面积的主要部分，迫使架构师在核心数量与缓存容量之间进行权衡。尽管智能数据预取为重新利用这部分硅面积提供了途径，但为多核系统设计协调的多级预取策略，却受到耗时一天的周期级模拟和复杂工作负载的严重制约。这一模拟瓶颈从根本上限制了新兴的基于大语言模型的微架构发现框架；例如，最先进的 ArchAgent 系统在单核基准测试上一次只演进一个缓存层级，直到最后才加入多核配置，并且在多核共享缓存争用场景下落后于人类冠军。

为克服这一瓶颈，我们推出了 DuoArchitect，这是一种自主智能体框架，旨在实现多保真度、多核心协同发现预取器。在 DuoArchitect 中，智能体共同演化评估代理和协调的预取策略，覆盖一级数据缓存、二级缓存和末级缓存层级，构建出比集群评估快 3–26× 倍的模拟代理，以及与硬件无关的预言器，以指导搜索。为了研究更智能的预取器能够替代多少缓存硅片面积，我们在缩减的缓存预算下评估并演化预取器，并发现，在最严格的缓存约束下（一级数据缓存和末级缓存容量均减半）进行演化，能够在我们测试的每种缓存容量下都产生总体最强的预取器：DuoArch-Nano。在第四届数据预取竞赛（DPC4）基准测试中，在全尺寸缓存下，DuoArch-Nano 实现了 9.2% 的几何平均 IPC 加速比（在四核心上为 +10.5% 混合工作负载），同时将四核DRAM流量降低了11.4%，收益超过人类冠军 BertiGO 和智能体 ArchAgent v2 的两倍。当末级缓存减半时，DuoArch-Nano 消除了基线总体性能下降8.6%的影响，并且总体表现略胜于完整末级缓存基线（+0.88%），表明此类智能体框架能够成功生成预取器，在每个核心上节省数兆字节的 LLC SRAM，同时总体每周期指令数完全不受影响。

1 引言

在过去十年中，服务器片上系统（SoCs）上的末级缓存（LLC）容量迅速增长，以满足不断增长的需求，同时占用了越来越大比例的宝贵计算硅片。例如，共享式32MB三级缓存约占八核AMD Zen 4核心复合体的44%，以及完整芯粒的37% [6, 22]。这种对大型物理缓存的严重依赖正面临一个关键的扩展瓶颈：尽管逻辑密度随着每一代新工艺节点持续提升，高密度静态随机存取存储器的扩展却基本停滞不前 [35, 38]。因此，服务器架构师面临着核心与缓存之间的艰难权衡。他们要么限制核心数量，以维持足够的每核心缓存容量；要么增加处理核心数量，但代价是大幅削减每个核心可用的缓存 [3, 6]。

这种硅片税引出了一个根本性的架构问题：更智能的数据预取器可以替代多少片上缓存容量而不损失性能

*两位作者对本研究贡献相同。

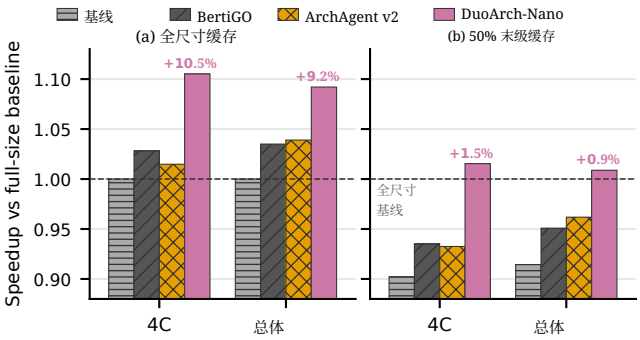


图 1：相对于全尺寸基线的加速比：(a) 使用全尺寸缓存；(b) 使用 50 % 末级缓存。

? 原则上，预取器可以隐藏片外存储器的延迟，并达到大得多的末级缓存的性能。实际上，要在不增加带宽需求的情况下，为现代缓存层次结构构建这样的预取器极其困难 [5, 31]。高性能的预取器必须在严格的元数据预算下协调多级缓存层次结构 [26, 34]。此外，在末级缓存容量和 DRAM带-宽都稀缺 [28, 33] 且片外存储器总线达到饱和时 [18]，它还不能导致性能下降。

自主大语言模型智能体为探索多级数据预取器的广阔设计空间提供了一种很有前景的方法。最近，ArchAgent v2 [11] 展示了编程智能体能够演化出在单核基准测试中击败冠军级、人工设计的 BertiGO [29] 的预取器。然而，智能体硬件设计面临着一个严峻的瓶颈：周期级仿真比软件基准测试慢几个数量级，对一个设计进行全四核（四核）评估需要超过一天的时间。这一模拟瓶颈严重限制了以往智能体系统能够探索的搜索空间。为节省模拟时间，ArchAgent v2 一次只在单核上演化一个缓存层级，并且直到最后阶段才引入多核配置。这种级联式方法使其设计在四核共享末级缓存争用下比 BertiGO 落后约 1.3%，同时四核 DRAM 流量增加了 7 %（比基线高 14%）。

为了让智能体驱动的探索突破多核系统的这一模拟瓶颈，我们提出了**DuoArchitect**。DuoArchitect [1],继承了DuoDiscover的多保真度范式，调度成群的沙箱化智能体，同时协同演化全部三个缓存层级。智能体无需在每次迭代中等待超过一天的完整模拟，而是在各自的沙箱中协同演化预取器设计和快速本地评估器（称为标尺），从而大幅加快迭代，并实现跨全部三个缓存层级的联合多核优化。随后，在分布式仿真集群上，使用公开的搜索集对有潜力的候选项进行评估，最后再在完整的官方集合上进行最终评估，其中还加入了留出工作负载。

我们将 DuoArchitect 应用于第四届数据预取锦标赛（DPC4）设置，[7], 在所有运行模式下均取得最先进的结果（§5.1），并以大幅优势超越现有工作。在为期 ~13 天的搜索中，我们的旗舰运行（**DuoArch-Nano**）在本地评估基准上筛选了约 ~19,800 个候选项，在搜索集上评估了 1,521 个候选项，并仅对 22 个新的最佳设计进行了官方评估。最终得到的三级预取器在全尺寸 DPC4 缓存下实现了 9.2% 的总体几何平均 IPC 加速比（图 1a），其在全带宽下为 9.6%，在有限带宽下为 7.5%，在四核（4C）下为 10.5%（图 6）。这使 BertiGO（3.5%）和 ArchAgent v2（3.9%）的加速比均增加了一倍以上。更重要的是，这些性能提升并未以增加内存流量为代价。事实上，与基线相比，DuoArch-Nano 将 4C 的 DRAM 流量降低了 11.4%。

关键发现。鉴于云端高密度服务器片上系统中每核末级缓存的缩减，[3, 6],我们还通过在缩减的缓存预算下评估和演化预取器（§5.1，§5.4），探索快速多保真搜索是否能够以预取器智能换取末级缓存芯片面积，并得出了两个核心发现：

• **预取器智能可以取代一半的末级缓存**。 将末级缓存缩减50%（从每个核心3 MB降至1.5 MB）会使基线速度降低8.6%。DuoArch-Nano完全消除了这一损失，并且总体上略胜于全尺寸末级缓存基线（+0.88%；图 1b），在每个核心回收1.5 MB的静态随机存取存储器，同时总体每周期指令数没有损失。即使末级缓存大小减半，DuoArch-Nano仍具竞争力的关键原因在于其高准确率和带宽效率。具体而言，在单核工作负载上，DuoArch-Nano所使用的DRAM 流量比全尺寸末级缓存基线少5–8%，而在四核DRAM流量方面与其持平（差异在0.4%以内）。

• **缓存稀缺会在演化过程中孕育更强的预取器**。DuoArch-Nano是在我们最严格的缓存约束下演化得到的，但在所有机器规模上总体都胜过在更大缓存上演化的预取器，包括全尺寸12 MB末级缓存配置（+9.2%对 7.1–8.0%）。这表明，宽裕的末级缓存会掩盖预取污染，而严重的缓存稀缺则会充当正则化器，迫使系统实施严格的效率和公平性控制。

微体系结构洞见。对 DuoArch-Nano 的分析表明，其性能提升源于多种预取引擎的组合，以及巧妙的协调与缓存管理。我们总结了性能提升背后的三点洞见（§3、§5.4）：

•**多样化的预取引擎**。在三个缓存层级中，DuoArch-Nano 使用了多种预取引擎，涵盖步长、空间和时间预取器。这些预取器会根据拥塞反馈进行调节，并覆盖全局和局部模式。

• **跨层级通信**。三种预取器并非各自独立运行，
而是协同工作。
• **缓存管理**。 DuoArch-Nano 的设计认识到，如果不协同设计缓存管理，预取器就无法发挥作用。由于 DPC 配置不允许直接修改替换状态，预取引擎通过向驻留于缓存的缓存行发出预取请求，间接调节慢速核心或关键缓存行（例如页表项）的替换优先级。

主要贡献。本文的主要贡献如下：
• **将多保真智能体发现扩展至周期级模拟——**
lation.我们提出 DuoArchitect，这是一个采用智能体构建的快速排序代理和诊断硬件预言器进行协同进化的系统
三级预取器（§2，§5.2）。
•**一个全新的、达到最先进水平的三级预取器**。DuoArch-Nano 在 DPC4 上实现了 9.2% 的总体每周期指令数加速比（在 +10.5% 的四核系统上），同时将四核 DRAM 流量降低了 11.4%，增幅超过一倍先前的人工作业和智能体加速比（§5.1、§3）。
• **用预取器智能换取末级缓存容量**。我们展示
进化预取器能够弥补末级缓存减半的影响
总体（+0.88%，相较于完整末级缓存基线），回收 1.5 MB 的
每个核心的 LLC SRAM，同时总体每周期指令数没有损失（§5.1）。

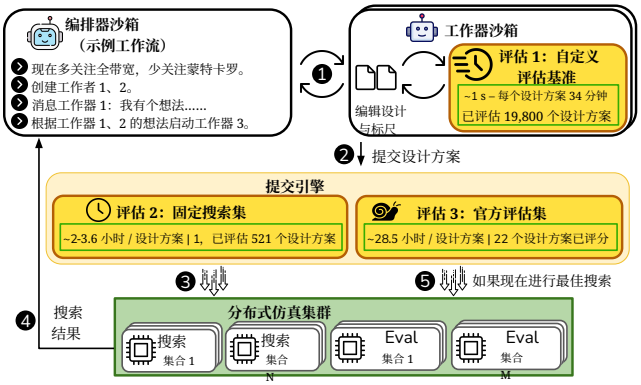


图 2：DuoArchitect 及其 3 个评估层级概览。

• **证明缓存稀缺会催生更强的预取器。** 在缩减缓存的处理器上进行多次进化运行后，我们发现，最严格的缓存约束产生了在我们测试的各种缓存容量（§5.4）之间总体泛化效果最佳的驻留与共享末级缓存 QoS 机制。

2 DuoArchitect

DuoArchitect 基于 DuoDiscover [1]，——一个用于自主智能体发现的多保真度框架，并将其改造为由硬件仿真反馈驱动的框架（图 2）。由于一次完整的 DPC4（详情见第4节）评估需要一天以上，比 DuoDiscover 面向的软件基准测试慢一到两个数量级，DuoArchitect 将评估划分为三级漏斗。1 在每个集群中，协调器通过决定要追求的目标来引导探索，并通过启动、沟通以及合并并行工作者的预取器设计来落实这些目标。在各自的沙箱内，每位工作者会迭代修改三级预取器设计，并使用快速评估器（我们称之为评估基准，每个设计耗时 ~1 s–34 分钟）筛除表现不佳的方案，然后 2 将有希望的候选方案提交给提交引擎。引擎 3 在分布式仿真集群上安排对固定搜索集的评估（包含 DPC4 中 131 条已披露的单核跟踪和 14–15 个 4C 混合负载，规模为 5M/20M，每个设计耗时 ~2–3.6 小时），并 4 将经验证的搜索结果返回给协调器。最后，每当搜索产生新的最佳设计时，都会在离线状态下异步地使用官方评估集对其进行评估：DPC4 中全部 610 条单核跟踪（规模为 50M/200M）以及 483 个 4C 混合负载（每个核心规模为 20M/50M；每个设计耗时 25–28.5 小时）。该评估集包含搜索集中的 131 条跟踪和 14–15 个混合负载，以及用于衡量对未见数据泛化能力的 479 条留出跟踪和 468–469 个留出混合负载。下文将详细介绍我们如何在整个流程中基于 DuoDiscover 进行构建。协调器—工作器循环。每个工作器沙箱都会在严格的存储预算内，以三级 C++ 预取器种子初始化，并获得对 131 条已披露跟踪的只读访问权限，以及一个并行化的自定义评估基准模板；工作者可以自由定制该模板，在提交前于本地筛选候选修改（第5.2节）。每个沙箱还包含一份可写的 ChampSim 模拟器源代码私有副本，使工作者能够对模拟器本身进行插桩，将其改造成自定义诊断预言机（第5.2节）。为适配基础版 DuoDiscover，我们在四种机制上修改了 ~1，共涉及 8k 行框架代码：(1) 为协调器提供一个只读 Docker 工具沙箱，将其视野限制在运行目录内，使其无法查看离线官方分数，也无法在工作者之间复制未经评分的代码；(2) 建立多目标账本，分别存储全带宽、limitBW 和 4C 加速比，使协调器能够动态调整目标权重，并对所有过往设计重新排序，而无需重新仿真；(3) 采用低于百分之一的确定性噪声门控，取代基础版 DuoDiscover 1% 的统计噪声下限，从而保留确定性的 0.1–3% 每周期指令数增益，而不会将其当作噪声丢弃；(4) 设置轮次与预算护栏。护栏将工作者的轮次限制在 2 小时，并优先为合并其他工作者设计的工作者分配空闲工作者名额和评分队列。若某个阶段剩余时间不足以完成评分，护栏还会阻止开启新轮次或合并设计。在每个 24 小时阶段，框架仅依据已披露的搜索集结果自动生成阶段简报，并通过自动检查从提示和笔记中剔除任何官方分数。

为协调器提供只读 Docker 工具沙箱，将其视野限制在运行目录内，使其无法查看离线官方分数，也无法在工作者之间复制未经评分的代码；(2) 建立多目标账本，分别存储全带宽、limitBW 和 4C 加速比，使协调器能够动态调整目标权重，并对所有过往设计重新排序，而无需重新仿真；(3) 采用低于百分之一的确定性噪声门控，取代基础版 DuoDiscover 1% 的统计噪声下限，从而保留确定性的 0.1–3% 每周期指令数增益，而不会将其当作噪声丢弃；(4) 设置轮次与预算护栏。护栏将工作者的轮次限制在 2 小时，并优先为合并其他工作者设计的工作者分配空闲工作者名额和评分队列。若某个阶段剩余时间不足以完成评分，护栏还会阻止开启新轮次或合并设计。在每个 24 小时阶段，框架仅依据已披露的搜索集结果自动生成阶段简报，并通过自动检查从提示和笔记中剔除任何官方分数。

提交引擎。 工作者 提交的候选设计会交给提交引擎。为避免浪费或[illegible]持续数小时的集群运行，引擎会在收到提交时冻结一份快照（因此工作者后续的修改不会影响排队中的提交），将每位工作者同时进行的评估限制为一个，并在相同设计和机器配置已评估过时立即返回存储的分数。对于每个新提交，引擎会在无网络容器内的 DPC4 模拟器干净副本上编译并检查该快照，依次通过四道合法性检查门：(1) 采用失败即拒绝策略的源代码检查，禁止访问私有缓存状态（缓存的标记数组和 MSHR 数组）或使用外部包含文件；(2) 采用失败即拒绝策略的构建检查，禁止调用特定文件、环境、进程或汇编相关功能；(3) 采用失败即拒绝策略的存储探测，确保一级数据缓存 32 KB、二级缓存 128 KB 和末级缓存 256 KB 的元数据预算不超标；(4) 与 8 个已发布的 DPC4 提交进行精确匹配的剽窃检查。通过检查的构建 会分发到仿真集群，作为 ~277 个任务，覆盖全部 131 条已披露跟踪（在全带宽和 limitBW 下）以及 14–15 个已披露的 4C 混合负载。为使搜索集评估比正式的 50M/200M 评估快一个数量级，同时不扭曲候选者排名，搜索集会缩短仿真窗口，将指令数减至 10× 5M/20M（Google 跟踪数据为 15M/60M，因为较短的预热阶段容易造成误导），而不是减少仿真的跟踪数量。在我们针对 8 个已发布 DPC4 提交进行的校准中，将窗口缩短至 15M/60M 后，仍保持了与全长度评估相同的排名（肯德尔秩相关系数 $\tau = 0.857$ [15]，其中 1 表示顺序完全相同，0 表示毫无相关性）；5M/20M 的排名与 15M/60M 的顺序一致（$\tau = 1.00$），而使用更小的跟踪子集则会改变获胜者（$\tau = 0.50$）。每个基线 IPC 都按缓存配置和指令窗口冻结（搜索使用 5M/20M；正式评估使用单核 50M/200M，以及 4C 每核 20M/50M），确保加速比始终是在匹配的缓存容量和窗口之间进行比较；同时，集群数组任务会落实每个站点的内存预留，并在发生内存不足终止或出现掉队任务时自动重试。如果任一跟踪或混合工作负载崩溃，整个评分即告失败；否则，引擎 会将各部分加速比返回账本（同时在冻结快照上重新运行工作者的标尺，以记录其一致性 τ）。

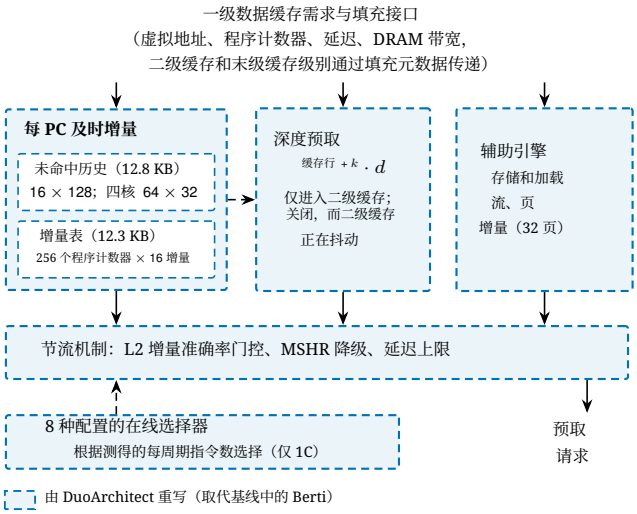


图 3：DuoArch-Nano 的一级数据缓存：将深度预取分阶段传递至二级缓存，并配备流式和页增量助手以及在线配置选择器。

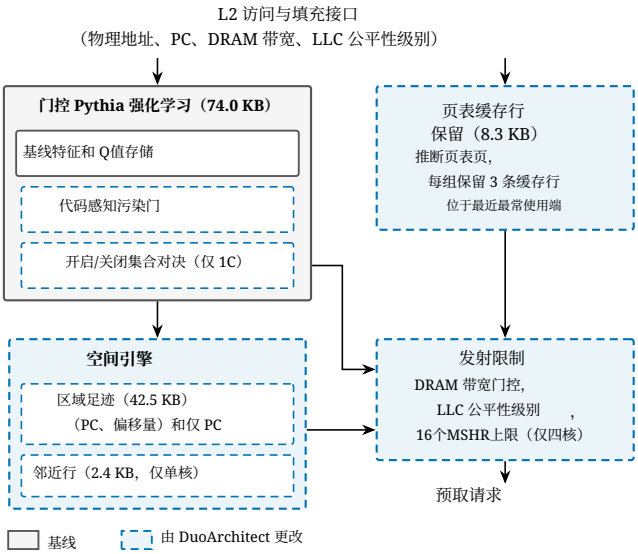


图 4：DuoArch-Nano 的二级缓存：在代码感知门控器之后部署 Pythia，并配备区域足迹引擎、邻近引擎和页表线保留引擎。

3 DuoArch-Nano 设计

图 3–5 展示了 DuoArch-Nano 的三级架构（分别使用了 DPC4 一级数据缓存、二级缓存和末级缓存存储预算的 99.2%、99.4% 和 99.8%），说明了每个缓存层级如何将更强的模式覆盖率与明确的缓存污染控制和带宽控制相结合。

一级数据缓存（每 PC 及时增量和二级缓存前瞻暂存）。基线中的 Berti 预取器（§4）以页为键，但同一页通常会混合步幅各异、互不相关的加载指令。因此，DuoArch-Nano 将 Berti 的键改为加载指令的程序计数器（图 3），并跟踪

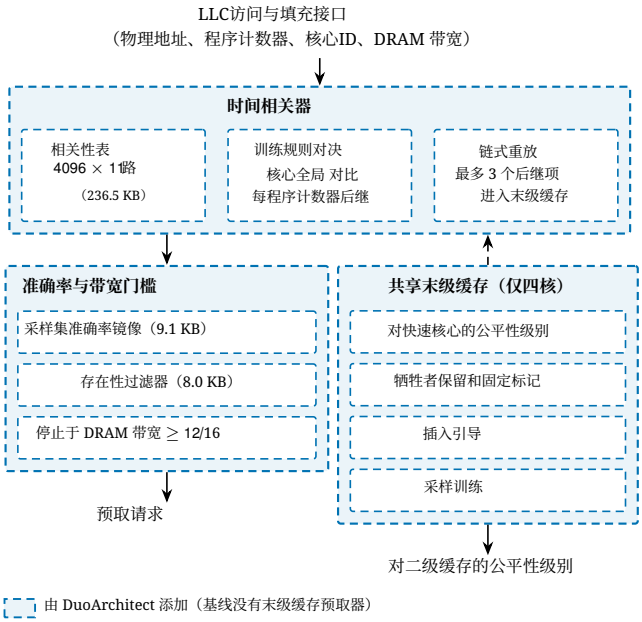


图 5：DuoArch-Nano 的末级缓存：一个配备准确率和带宽门控以及四核共享控制的时间相关器。

每条指令的未命中时间戳，以学习及时的地址增量，并利用每个增量的历史覆盖率来决定预取是直接填入一级数据缓存，还是仅填入二级缓存。置信度最高的增量会将更深层的前瞻步骤暂存到二级缓存中，因此后续的一级数据缓存预取会命中二级缓存。每当二级缓存报告预取污染严重，或末级缓存传递 4C 公平性退避信号时，这种前瞻机制都会自动暂停。

为捕获单 PC 增量表无法识别的访问模式，两个辅助引擎与 Berti 并行运行：(1) 流引擎根据一级数据缓存中未由此前任何加载或预取触发的填充，预测存储未命中；(2) 页增量引擎跟踪包含多个加载 PC 的循环中的空间步幅。流引擎会在准确率较低或 DRAM 带宽使用率较高时暂停；页增量引擎会在 DRAM 带宽使用率较高时暂停。

所有 L1D 引擎都使用虚拟地址，且不会在页边界处停止，因此增量和深度前瞻可以跨入下一页。发生拥塞时，在 MSHR 占用率较高（16 个 MSHR 中有 ≥10 个处于忙碌状态）时，L1D 预取会降级为二级缓存填充；当平均需求未命中延迟超过 350 个周期时，则会将每次触发的发出速率限制为一次。预取度和前瞻深度由在线纪元仲裁器动态确定，该仲裁器会定期测试不同的预取度和前瞻深度，仅在某种配置能提升测得的每周期指令数时才切换。

二级缓存（污染门控 Pythia、空间足迹引擎和 PTE 保留）。在容量减半至 1 MB 的二级缓存中（图4），Pythia 激进的强化学习驱动的请求经常驱逐有用的指令和数据，或耗尽有限的 DRAM 带宽。DuoArch-Nano 将 Pythia 置于一个 36.5 KB 的影子标签监控器之后（2,048 组 × 8 路、每路采用 10 位部分标签，计入二级缓存预取器预算），该监控器会跟踪二级缓存数据驱逐，并检测 Pythia 的填充何时取代指令缓存行（当某次填充驱逐的地址不在仅含数据的影子中时，据此推断）

）或再次请求的数据。对于中度污染（有害预取与有用预取之比为 >50%），DuoArch-Nano 会将 Pythia 限制为每次访问最多进行一次预取；对于严重污染（>90%），则限制为每 8 次访问进行一次预取（保留足够的预取，以便检测 Pythia 何时再次变得有用；在 1C 上，还配合采用集合对决式开关，并在总线利用率较高时实施更严格的准确率门控器）。

为覆盖 Pythia 未能捕获或会降低预取力度的空间访问模式，系统增加了两种空间预测器：(1) 区域足迹引擎 [30]，用于记录并重放 2KB 区域内已访问缓存行的位向量；(2) 邻近引擎，用于学习每个加载 PC 在其至少一半的未命中中，哪些相邻缓存行偏移量（±1 到 ±4 缓存行）会很快发生未命中，并对其进行预取。

由于将两级缓存的容量都减半会导致叶级页表（PTE）缓存行在地址转换之间被逐出，二级缓存引入了一个 8.3 KB 的**页表行保留引擎**。该引擎通过反复出现的页表遍历填充来识别 PTE 缓存行（即填入二级缓存、此前没有加载或预取请求过、且从未被任何数据访问触及的缓存行），并在影子标签中为每个二级缓存组最多标记三个反复出现的 PTE 缓存行。每次访问缓存组时，该引擎都会向每个已标记的缓存行发出预取请求（生成一次二级缓存命中，使其刷新至最近最常使用位置，且不产生内存流量）。所有二级缓存引擎的发出速率均由实时 DRAM 带宽阈值、末级缓存的四核公平性信号，以及针对四核的 MSHR 占用上限共同控制。

LLC（时间相关器和四核 QoS 协调器）。由于基线方案将 256KB LLC 预取器槽位留空，因此逃过一级数据缓存和二级缓存的不规则指针追逐缺失无法得到预取。DuoArch-Nano 使用片上时间（马尔可夫）地址相关表填充该槽位，将每个 LLC 缺失与其后继项关联起来，并在再次发生缺失时重放最多包含三个后继项的链（图 5）[34]。该引擎使用集合对决，在每核心后继项训练和每程序计数器后继项训练之间进行选择。预取发出受 DRAM 总线利用率和采样准确率镜像控制；后者跟踪预取的缓存行是否在每 16 个采样 LLC 组中有 1 个被使用。

在四核执行（4C）期间，四个核心共同争用减半后的 6 MB LLC，LLC 预取器还充当中央 QoS 协调器，通过四种方式保护混合工作负载中的最慢核心，使其免受共运行任务抖动的影响。首先，当其每 64K 个周期进行的检查检测到某个内存受限、不规则的受害核心（<0.5 每周期指令数、到达 LLC 的 ≥5 二级缓存需求每千条指令未命中数，以及每次需求缺失对应的 <2 二级缓存预取）时，它会通过二级缓存转发退避信号，使快速攻击核心的一级数据缓存预取器静默。其次，在 4C 模式下，它重新利用相关表的 11 个路中的两个，用于**为最慢受害核心固定每组一条 LLC 命中缓存行（<0.5 每周期指令数）**：每当共运行任务向该组填充数据时，预取器就会向存储的受害核心地址发出预取，从而生成一次 LLC 命中并刷新其替换年龄。第三，对于流式共运行程序（<15% LLC 命中率），它将 DRRIP 的集合对决选择器 [14] 引导至远端（即将被驱逐）插入方式：将它们的填充应答到远端插入主导集合，并向近端插入主导集合发出预取。最后，对于缺失序列很少重复的核心（写入中有 <1/16 与现有条目匹配，或准确率为 <31%），将其写入共享相关表的比例限制为每 8 次写入 1 次，使其无法覆盖受害核心学习到的链。

4 方法论

DPC4 基准测试。DPC4 在 ChampSim [10] 中评估三级数据预取器，涵盖两种单核带宽运行模式

表1：DPC4处理器层级结构与评估运行模式。

组件	配置
L1D缓存	48 KB/核心，12 路，5 周期，16 个 MSHR；Berti基线，32 KB 预取预算
L2缓存	每核心 2 MB，16 路，10 个周期，32 个 MSHR；Pythia 基线，128 KB 预取预算
共享末级缓存	每核心 3 MB，12 路，35 个周期，每核心 64 个 MSHR，DRRIP；无基线预取器，256 KB 预取预算
DRAM	1 个通道，4,800 MT/s（全带宽，四核）或 800 MT/s（limitBW）
单核配置	610 条跟踪记录（50M/200M），速率为 4,800 MT/s（全带宽）和 800 MT/s（limitBW）
四核配置	483 个四轨迹混合组（每个核心 20M/50M），4,800 MT/s，调和平均每周期指令数

（**fullBW**和**limitBW**）以及四核共享末级缓存配置（**4C**），如表 1 所示（4C 在 484 个混合工作负载中的 483 个上进行评估，排除了一个会导致基线发生死锁的图计算混合组）。该基准测试涵盖四个工作负载套件：SPEC CPU 2017、Google 服务器、人工智能/机器学习和图计算。为防止设计过程中过拟合，搜索阶段公开了 131 条单核跟踪记录和 14–15 个 4C 混合负载，其余 479 条跟踪记录和 468–469 个混合负载则严格留作最终评估。

基线和已有预取器。DPC4 **基线** 在一级数据缓存上搭配 Berti [24]，并在二级缓存上搭配 Pythia2 [5]，同时将 256KB 的末级缓存槽位留空。Berti 为一级数据缓存未命中和填充记录时间戳，以测量取数延迟并学习具有高覆盖率且及时的地址增量，再根据覆盖率和 MSHR 占用率选择一级数据缓存或二级缓存的填充位置（不过，ChampSim 的 DPC4 基线按页而非加载 PC 对 Berti 建立索引，从而混淆了无关的指令）。Pythia 将二级缓存预取建模为在线表格型强化学习（SARSA），将上下文特征（PC+增量和近期增量序列）映射到页内偏移操作；当内存带宽繁忙时，对不准确的预取施加更重的惩罚。然而，由于缺少置信度门控，它在不规则工作负载上仍会发出低准确率的预取（DPC4 上的准确率为 45–55%，表 2）。

除基线外，我们还与两种表现强劲的既有三级设计进行比较：(1)**BertiGO** [29], , 即 DPC4 人工评选的优胜方案，它将按程序计数器区分的及时增量一级数据缓存预取器与全局步幅跟踪器以及二级缓存中的空间/增量表结合起来；(2) **ArchA-gent v2** [11], , 一种进化式 LLM 代理系统，它依次在相互隔离的情况下演化一级数据缓存（在 Berti 的基础上加入全局步幅与增量表以及强化学习仲裁器）、二级缓存（五特征求和池化 Pythia）和末级缓存（步长比时序/步幅预取器）。

缓存配置与指标。 我们评估采用大容量缓存的 DPC4 配置（表 1；每个核心配备 2 MB 二级缓存和 3 MB 末级缓存），并将其与两种缓存受限配置进行比较。这两种配置均保持 48 KB 一级数据缓存和所有预取器存储预算不变：(1) 50% 末级缓存（每个核心配备 2 MB 二级缓存和 1.5 MB 末级缓存，或四核共享 6 MB，减少每个核心 1.5 MB 的静态随机存取存储器），以及 (2) 50% L2 + 50% 末级缓存（每个核心配备 1 MB 二级缓存和 1.5 MB 末级缓存，减少每个核心 2.5 MB 的静态随机存取存储器）。我们的旗舰预取器 DuoArch-Nano，是在限制最严格的机器上从未经修改的基线演化而来（其二级缓存和末级缓存容量均减半）；§3 介绍其三级架构，§5 介绍其性能，而 §5.4 则通过与在 50% 末级缓存配置下演化的方案进行比较，分析静态随机存取存储器匮乏为何能带来更高性能。

我们从两个互补尺度报告官方增益。第一个是相对于全尺寸基线，即将加速比归一化为相对于采用

全尺寸缓存的基线（因此， ≥ 0% 表示缩减缓存配置达到或超过全缓存基线）。第二个是相对于同尺寸基线，即将加速比归一化为相对于同一缩减缓存机器上的基线，以衡量预取对特定缓存配置的提升幅度。

实验设置。 在我们的进化运行中（§5），每次运行都会以 24 小时为步长，花费 ~13 天进化预取器；整个过程中最多使用 8 个并行工作组，每组有 4 名工作者，在 44 个 CPU 核心上运行，工作者和协调器均使用 Claude Opus 5.5。即使采用自定义基准筛选，每次运行也会在搜索集上评估 800–1,521 个候选项（每次运行消耗 50,000–133,000 个搜索核心小时），而每次 离线官方评估还需额外消耗 ~3,000 个核心小时。我们的所有运行总计消[redacted]超过 300,000 个核心小时，因此每次进化运行都需要大量集群投入。

5 评估

由于二级缓存和 LLC SRAM 占据了处理器芯片面积的很大一部分，架构师面临一个根本性问题：更智能的预取器能够在不损失性能的前提下，替代多少片上缓存容量？更耐人寻味的是，在缓存极度稀缺的情况下进化预取器，是否会促使我们发现更强大的通用机制，即使缓存容量较大时也能表现出色？

5.1 跨机器尺寸的结果

图 6 评估了 DuoArch-Nano 在全部三种处理器配置下的设计表现，并与固定基线预取器、DPC4 人类冠军（BertiGO）以及此前大语言模型智能体的最先进方案（ArchAgent v2）进行了对比。从全尺寸缓存逐步缩减到限制最严格的机器，可以得到三项主要结果：

(1) **凭借全尺寸缓存实现最先进的性能：** 尽管 DuoArch-Nano 完全在最小缓存上进化而成，但在使用全尺寸 DPC4 缓存时，相较于基线实现了 9.21% 的总体加速比（+9.58% fullBW、 +7.54% limitBW 和 +10.52% 四核），其收益超过 BertiGO（3.50%）和 ArchAgent v2（3.90%； [11] 中报告为 3.8% ）的两倍。

(2) **仅使用一半的末级缓存，却击败全尺寸末级缓存基线：** 每个核心减少 1.5MB 的末级缓存会使基准处理器的性能下降 −8.56% （ fullBW 下为 −7.07%、limitBW 下为 −8.80%、四核下为 −9.80%）。固定的 BertiGO 和 ArchAgent v2 相较于同尺寸基线实现的相对加速比略高于全尺寸时（分别从 3.50% 升至 4.11%，以及从 3.90% 升至 5.32%），证实缓存越小，预取的重要性越高。然而，在有限带宽和四核争用下，它们针对全尺寸缓存的调优无法弥补缺失的末级缓存容量，使其性能分别比使用全尺寸缓存的基线低 −4.91% 和 −3.82%。相比之下，DuoArch-Nano 克服了缺失的末级缓存容量，性能比使用全尺寸缓存的基线高 +0.88%（比同尺寸基线高 +10.16%），表明进化预取器可以在不牺牲总体每周期指令数的情况下替代一半的末级缓存。

(3) **在缓存容量减半时挽回大部分损失：** 当二级缓存和末级缓存都减半时，基线处理器的性能下降了 −15.28%（−13.58% / −15.66% / −16.56%），而固定的 BertiGO 和 ArchAgent v2 则骤降至相对于全尺寸基线的 −12.06% 和 −11.46%。虽然同时缩减两级缓存带来的硬件代价过大，无法完全弥补，但 DuoArch-Nano 挽回了官方双缓存损失的 59%，

表2：全尺寸机器上的缓存计数器。

设计	L1D 每千条指令未命中数	L2 每千条指令未命中数	LLC 每千条指令未命中数	L1D 每千条指令	二级缓存 二级缓存预取 DRAM		
	未命中数	准确率%	准确率 %			PKI	PKI
1C FullBW							
基线	22.33	10.09	6.53	70.7	47.2	49.0	21.0
ArchAgent v2	**18.36**	**8.14**	5.56	81.1	67.3	**35.4**	18.5
DuoArch-Nano	22.15	8.35	**5.15**	**83.3**	**69.2**	53.7	**18.0**
1C 带宽限制							
基线	23.26	10.20	6.07	70.9	45.1	53.9	20.9
ArchAgent v2	**20.00**	**8.39**	5.60	83.4	51.1	**30.7**	18.1
DuoArch-Nano	23.79	10.26	**5.52**	**83.6**	**76.1**	50.0	**17.4**
4C							
基线	76.02	21.21	12.25	84.4	54.9	185.9	52.1
ArchAgent v2	**62.53**	17.06	11.61	90.7	**70.1**	179.5	59.3
DuoArch-Nano	71.53	**16.24**	**8.50**	**91.3**	66.8	**110.1**	**46.1**

每千条指令未命中数统计需求未命中。预取：发出的二级缓存预取。DRAM：DRAM 请求。每千条指令：每千条指令。均值按跟踪（610 个）或四核混合核心（483 个混合工作负载）计算。

（相对于全尺寸基线提升至 −6.20%），并且比同尺寸基线高出 +10.52%；在这台机器上，它比 BertiGO（+3.95%）和 ArchAgent v2（+4.66%）高出 5.9–6.6 个百分点。在最严苛的缓存约束下演化出的单个预取器竟能在所有机器上占据主导地位，这表明演化过程中的缓存稀缺会孕育出强大的

ger, more g可泛化的 p重新预取器（§5.4）.

在每种评估运行模式下，DuoArch-Nano在全带宽、limitBW和四核配置下的所有机器规模中均优于此前的所有设计方案。

加速比的来源。 表 2 比较了全尺寸机器上基线、ArchAgent v2和DuoArch-Nano的需求未命中、预取准确率、二级缓存预取量和DRAM 流量。在三种运行模式下，DuoArch-Nano都将最低的末级缓存需求未命中与最高的一级数据缓存预取准确率（83.3–91.3%）、较高的二级缓存准确率（66.8–76.1%）以及最低的DRAM 流量（17.4–46.1 PKI）结合起来。在四核配置下，虽然此前的预取器会发出大量推测性L2请求，争用共享末级缓存和内存总线（185.9和179.5 PKI），DuoArch-Nano却将33.7 PKI的高准确率L1D前瞻分阶段送入二级缓存，并将四核L2预取减少41%（降至110.1 PKI），同时仍将四核末级缓存需求未命中减少30.6%（12.25 → 8.50 每千条指令未命中数）。它通过在所有三个级别上限制低置信度推测来实现这种选择性；当无法准确预测非规则访问时，则保留驻留缓存行，并引导流式填充进行提前驱逐（§3）。

四核内存子系统在全尺寸缓存下就已受到队列限制（~1，平均末级缓存未命中延迟为 800 个周期，而 ~200 周期（单核）），但其他所有评估的预取器都会使四核 DRAM 流量高于基线（52.1 每千条指令；图 7），其中 DuoArch-Micro（预热）达到 53.7 每千条指令（+3%），BertiGO 和 DuoArch-Micro 达到 55.4 每千条指令（+6%），ArchAgent v2 则达到 59.3 每千条指令（+14%）。相比之下，DuoArch-Nano 是唯一一种能将四核片外流量降至全尺寸基线以下的设计，可将 DRAM 请求减少 11.4%，降至 46.1 每千条指令。消除浪费的片外流量，使 DuoArch-Nano 能够取代物理缓存容量。共享末级缓存减半后，基线的四核 DRAM 流量上升 24%，达到 64.7 每千条指令；BertiGO 和 Ar- chAgent v2 的流量更高，分别为 66.6 和 67.6 每千条指令。相比之下，DuoArch-Nano 将末级缓存减半时的四核 DRAM 流量控制在 52.3 每千条指令，比同尺寸基线低 19%，且与全尺寸基线的 52.1 每千条指令相差不到 0.4%；两级缓存均减半时，其流量仍比同尺寸基线低 26%

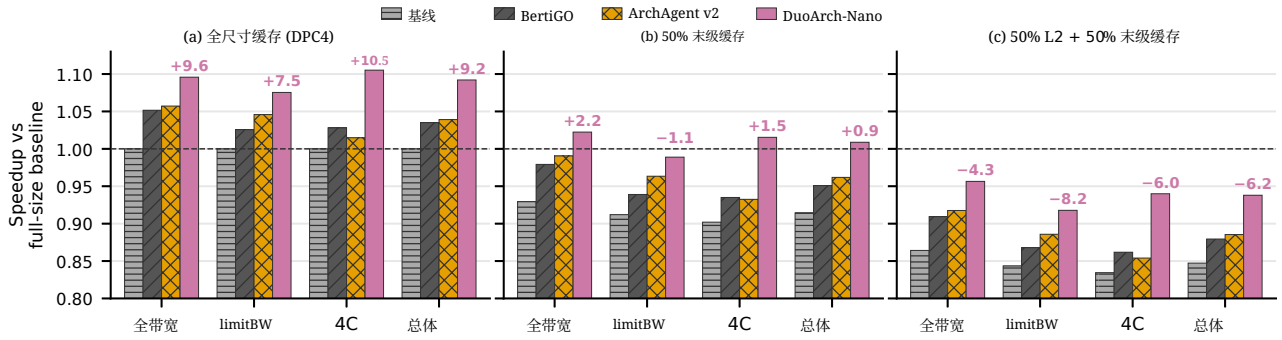


图 6：各机器在每种配置下，相对于采用全尺寸缓存的基线所实现的官方加速比。

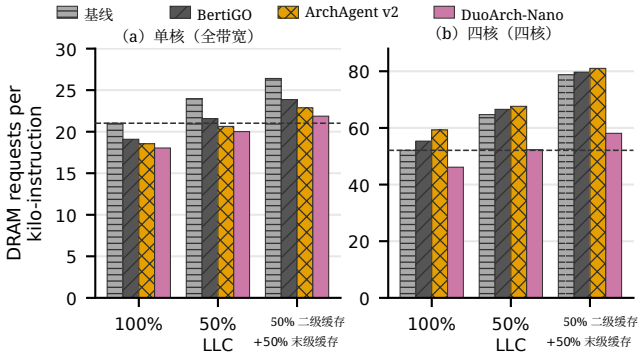


图 7：每台机器每千条指令的 DRAM 请求数。虚线表示采用全尺寸缓存的基线。

（58.1 每千条指令）。这缓解了内存总线排队，因此即使末级缓存减半，DuoArch-Nano 在四核场景下仍胜过全尺寸基线（+1.54%）。

量产设计还可以将节省下来的缓存硅片面积用于增加核心或加速器。例如，根据 AMD Zen 4 核心复合体面积分解（其中 32MB 的三级缓存占八核复合体的~44% [6, 22]），将每核 LLC 容量减半可使每个核心的+缓存瓦片缩小 ~16%，从而释放足够的硅片面积，约可增加 20% 的核心。

逐轨迹鲁棒性和留出泛化。在严重缓存稀缺的情况下进行演化，是否会导致预取器对搜索跟踪记录过拟合，或在缓存空间紧张时损害异常工作负载的表现?

答案是否定的。图 8 显示，在最受限的 50% L2 + 50% 末级缓存配置下，针对全部跟踪记录（包括未见跟踪记录），DuoArch-Nano 在全带宽下仅使 610 条单核跟踪中的 2.5% 变慢，在 limitBW 下则使 7.2% 变慢。BertiGO 分别使 24.6% 和 14.8% 变慢，ArchAgent v2 则分别使 7.5% 和 11.1% 变慢。在全带宽下，DuoArch-Nano 在 610 条跟踪记录中的 522 条和 546 条上分别胜过 BertiGO 和 ArchAgent v2，而且在 359 条 Google 跟踪数据中没有一条变慢。在四核场景下，其稳健性优势更加明显。DuoArch-Nano 仅使 483 个 4C 混合负载中的 14 个变慢，而 BertiGO 和 ArchAgent v2 分别使 111 个和 136 个变慢；它在 437 个和 425 个混合负载上分别胜过它们。在 4C 图计算混合负载上，它提升了 +12.1%，且 90 个混合负载无一变慢；相比之下，BertiGO 和 ArchAgent v2 分别下降 −0.03% 和 −3.0%，并分别使其中 42 个和 54 个变慢。最后，DuoArch-Nano 的泛化能力能够很好地适应未见工作负载：在 479 条留出的单核轨迹上，它相对 ArchAgent v2 和 BertiGO 的领先优势保持不变或进一步扩大（全带宽下分别为 +4.2% 和 +5.5%；limitBW 下分别为 +3.7% 和 +6.1%）；在 469 个未见的 4C 混合负载上，它比同尺寸基线高出 +12.0%（接近其在 14 个已搜索混合负载上取得的 +14.0% 增益）。

5.2 评估基准如何助力设计演化

为了沿着图 9 所示的轨迹演化 DuoArch-Nano，而无需在集群上为每个想法等待数小时，工作线程使用自定义评估基准，其运行 3× 比搜索集评估 65× 的实际运行时间更短。它们以两种互补方式使用这些评估基准：一是作为瓶颈诊断标尺，在编写预取器代码之前隔离硬件瓶颈；二是作为候选项排名基准，在提交至集群之前筛选完整的 C++ 预取器修改。

瓶颈诊断标尺。在其沙箱中，每个工作线程都会获得 ChampSim 源代码树、131 条已披露跟踪和随附的评估基准模板。由于工作线程可以完全写入其专用的模拟器副本，因此能够通过对本地模拟器进行插桩或修改来构建目标专用评估基准，例如模拟零周期页表遍历、将缓存行固定在选定级别，或添加自定义延迟探针。这些测量会显示某个瓶颈造成了多少每周期指令数损失，以及应在哪个缓存层级修复它，然后工作线程才会投入轮次编写完整的 C++预取器（表 3）。这些本地模拟器修改不会影响集群评估，因为提交引擎只会将已提交的预取器模块针对模拟器的干净副本进行编译（§2）。DuoArch-Nano 的大多数关键机制都源自代理自行设计的此类诊断实验，推动了图 9中的演化轨迹：早期步骤确立了三级预测器骨干和初始公平性中继，后续步骤则在单核心二级缓存容量突破与四核共享末级缓存争用控制之间交替推进。以下两个示例说明了代理如何在编写预取器代码之前使用预言机测试来隔离小型缓存瓶颈：

• **诊断二级缓存页表驱逐（+1.24% 搜索）：**在开始编码之前，一名工作线程在 Google 跟踪数据上运行了预言机模拟，结果显示，免于页表遍历可带来 +7–19% 的每周期指令数提升，而让 PTE 缓存行常驻二级缓存可带来 +6–17% 的提升，这证明叶级 PTE 缓存行在两级缩减后的缓存之间的页表遍历间隔内发生了抖动。第二个预言机确定了正确的缓存层级：将 PTE 缓存行固定在共享末级缓存中可带来 ~0% 的提升，而将其固定在 1MB L2 中则带来 +2.2–8.3% 的提升。由于预取器

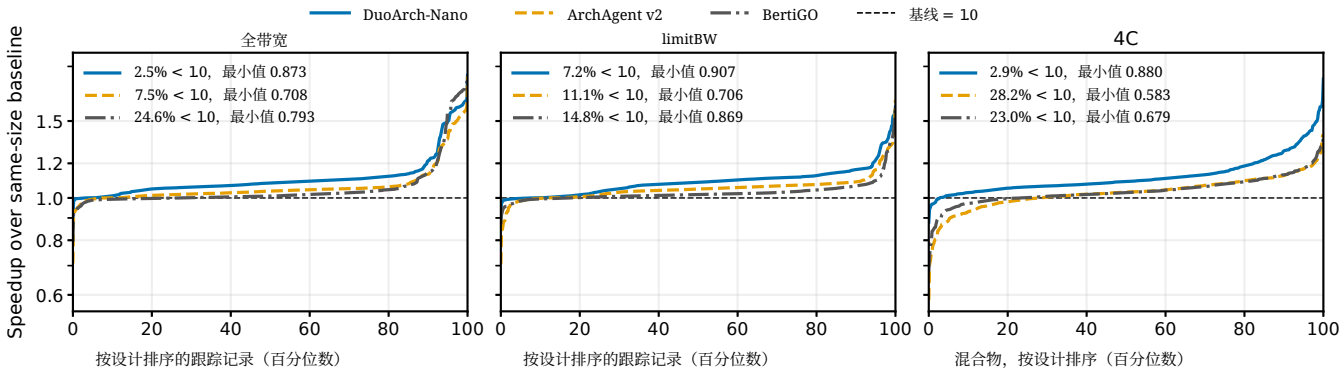


图8：在 50%L2 + 50% 末级缓存机器上，对于每条跟踪，在全带宽/limitBW 下以及每种 4C 混合配置中，相对于同尺寸基线的加速比；按设计排序。

表 3：DuoArchitect 在 50% L2 + 50% 末级缓存机器上构建的瓶颈诊断评估基准及其推动形成的机制。

级别	诊断标尺	测量内容与主要发现	所得机制（搜索增益）
L1D	缺失延迟探针	测量 1C 与四核下的一级数据缓存缺失延迟：逐程序计数器 Berti 胜过页键控 Berti（1C 下为 +2–3%，四核下为 +2.4%），但其激进流将四核下的缺失延迟推高至 1,000–1,200 个周期（而 1C 下为 ~250）。	带高延迟节流器的逐程序计数器及时增量 Berti（超过 350 个周期时，每次触发预取 ≤ 1 ）。
L2	逐轨迹消融	逐条跟踪禁用基线的二级缓存 Pythia 在有限带宽下：在一条 SPEC 轨迹上提升了 +20.0%，在一条图轨迹上提升了 .2%，但在广度优先搜索上下降了 −9.3%（因此排除了静态切换）。	可动态启用或禁用 Pythia 的在线集合对决门控在运行时。
L2	理想化 页表遍历与固定预言机	模拟 0 周期页表遍历，并固定 PTE 二级缓存与末级缓存中的缓存行：免费页表遍历在 Google 跟踪数据上带来 +7–19% 的收益，而在 1 MB 二级缓存中固定 PTE 可带来 +2.2–8.3% 的收益（而在末级缓存中为 ~0%）。	二级缓存 PTE 保留引擎（固定标记不产生额外存储）推断 PTE 缓存行并刷新它们 在最近最常使用的位置（+1.24%）。
LLC	离线缺失流分析器	分析残余末级缓存缺失的马尔可夫覆盖率 缺失：在指针追踪型 SPEC 跟踪记录中覆盖了 33–43%（在 Google 上为 2–20%），而四核共运行程序导致受害者 30% 的链路驱逐。	256 KB LLC 时间相关器（+0.65%），以及四核（+0.46% 四核）下的采样表写入。
LLC 和一级数据缓存	单混合替换预言机	测试抑制快速核心的一级数据缓存（+9–17% 在两个混合工作负载上，另外两个上为 −27%），并插入共运行任务的 LLC 以最低优先级进行填充（使 IPC 最低的受害核心的 IPC 提高 +34%。）。	受害者条件公平中继（+1.5% 四核）和借用路受害者固定标记。

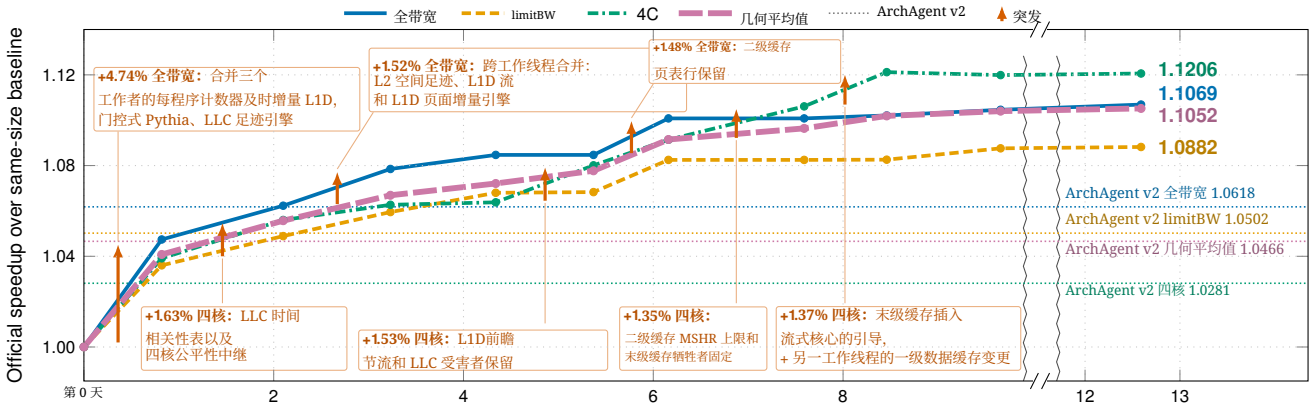


图9：DuoArch-Nano 在 50% L2 + 50% 末级缓存机器上的演化轨迹（相较于同尺寸基线的官方加速比）。标注框突出显示了主要阶段性增益及其引入的机制（横轴省略了第 9.9 至 11.7 天之间的空闲间隔）。

API 既不公开页遍历标志，也不公开替换状态，因此该工作线程设计了未解释填充推断和预取命中

MRU刷新机制（搜索增益为+1.24%，后续每次 L2 访问时刷新又带来 +0.15%的增益）。

表4：DuoArch-Nano 中不同候选项排名基准的速度–保真度权衡，按轨迹数量分组。

轨迹数量	典型窗口	快多少	排序正确
56（模板）	2M/8M	3.0×	86% (19/22)
16–20	1M/4M–3M/10M	26×	77% (55/71)
7–15	1M/2M–2M/2M	65×	57% (142/250)
≤ 6	0M/1M–2M/3M	59×	50% (67/135)

• **诊断四核拖尾者饥饿和末级缓存受害者固定标记：**观察到每种混合工作负载的四核分数受最慢核心制约后，一名早期工作者发现，无条件关闭快速核心的一级数据缓存预取器，挽救了两种混合工作负载（+9% 和+17%），但损害了另外两种（−27%）；根据低每周期指令数、高每千条指令未命中数、低实用性的受害核心来触发一级数据缓存退避，使四核搜索集的分数提高了+1.5%。后来，另一名工作者运行了单一混合工作负载的替换预言机，结果显示，降低共运行任务的末级缓存填充优先级，使受害核心的每周期指令数提高了 +34%，由此启发了借用路受害者固定标记（结合二级缓存 MSHR 上限后，四核评估分数提高+0.9%）；又过了一步，工作者们在没有预先运行预言机的情况下，加入了流式核心插入引导（四核评估分数提高+2.5%）。**候选项排名基准**。工作者实现候选预取器后，会先使用候选项排名基准在本地筛选，再提交到集群。每名工作者都从一个模板开始；该模板在搜索前经过离线校准。模板以全带宽运行 56 条已披露的单核跟踪记录，并按照各套件在 131 条已披露跟踪记录中的占比进行分层抽样（9 条 SPEC17、18 条 Google、13 条人工智能/机器学习和 16 条图）；同时限制每条跟踪的运行时间，以便并行批处理能在 14–15 分钟内完成。模板还会以 limitBW 运行其中 27 条（即在 800 MT/s 下基线每周期指令数下降最大的那些跟踪）。所有跟踪均使用 200 万条指令预热和 800 万条指令窗口；在我们的校准中，这是能够保持已知 DPC4 预取器排名的最短窗口。之后，工作者可以自由自定义此模板，在整个运行过程中生成了 202 种不同的跟踪集和窗口配置。为此，工作者们独立地重新发现了抽样模拟，通过调整跟踪数量和指令窗口参数，在模拟时间与排名保真度之间进行权衡。每次搜索集评估时，提交引擎都会在沙箱外对冻结快照重新运行工作者的评估基准，以记录其候选项排序是否与搜索集相同（§2）。表4 汇总了 DuoArch-Nano 上的这些检查结果，并比较搜索集评估的中位墙钟时间加速比与

在相差 ≥ 0.1% 的候选项上，两两排序准确率。56条轨迹模板（实际运行时间快 3.0×，模拟工作量少 9.6×）能正确排列 86% 的候选项对（跨工作线程为 91%）。经过适度剪枝、保留 400 万至 1000 万个窗口的 16–20 条轨迹子集（速度快 26×，工作量少 25–50×）能正确排列 77% 的候选项对。将轨迹数量减至 15 条以下，同时把窗口缩小到 100 万至 300 万条指令（快 59–65×，受构建和启动开销限制），会使一致率降至 57%，继而降至随机猜测水平的 50%。

搜索集验证可保障最终质量。即使工作者在每轮中使用更小的（≤ 15 条跟踪）评估基准进行快速合理性检查，局部代理指标的噪声也不会影响运行结果：评估基准只会筛选工作者提交的候选项，而所有通过筛选的设计都严格依据完整搜索集（131 条跟踪记录和 14–15 种混合工作负载）进行选择。在 DuoArch-Nano 中，每当工作中的

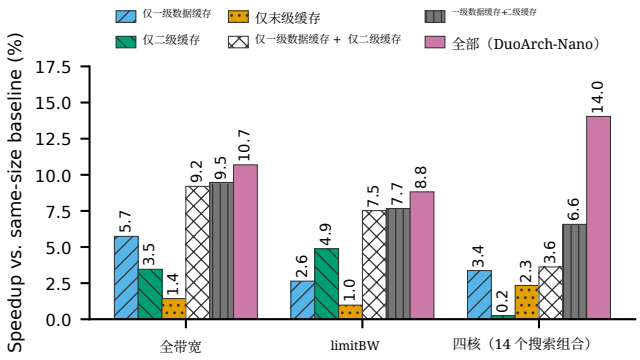


图 10：在二级缓存为 50%、末级缓存为 + 50% 的机器上，对 DuoArch-Nano 进行逐级消融（相对于同尺寸基线的加速比）。

评估基准预测性能会提升时，搜索集有 87% 的情况（60/69）也得出了相同结论；对于过度剪枝的评估基准产生的误报，搜索集也能准确剔除（同时，在各步骤边界将工作者重置为 56条轨迹模板，可防止过度剪枝的评估基准影响后续步骤）。类似地，在四核场景中，由于共享末级缓存争用需要更长的预热时间，短时的 1M/4M 运行结果与 20M 评分的相关性较低（而在完整的 20M 窗口下进行 2 种混合工作负载的 4C 检查，Kendall $\tau = 0.71$–0.78），因此，在集群搜索集上对全部 14–15 种 4C 混合负载进行评分，避免了单核评估基准为了提升单核速度而牺牲 4C 公平性。总的来说，表 4 展示了模拟器支持的智能体搜索的理想平衡点：适度剪枝（在 4M–10M 窗口下使用 16–56 条跟踪）可实现 3–26× 更快的局部筛选，排名准确度达到 77–86%，而集群搜索集则能捕捉任何带噪声的局部估计。**编排相较于独立智能体的价值。**为单独评估多智能体群编排和跨工作者合并的收益（而非底层模型本身的影响），我们将使用全尺寸缓存的 DuoArchitect 编排运行，与三个独立、未经编排的 Claude Code 智能体进行了比较。所有运行均从基线预取器（§4）开始，使用 Claude Opus 5 和 131 条已披露跟踪，持续运行 15 个预算日。即使将计算预算设为相同的 72.7k 忙碌核心小时（即独立智能体运行至第 13.5 天时的用量），我们的全尺寸运行在 fullBW 上也已领先于表现最好的独立智能体（6.4% 对 5.8%）。由于缺少协调器来剪除停滞分支或合并工作者之间互补的发现，表现最好的独立智能体在第 11 天后便停滞在 4.4% 的总体几何平均值（fullBW 为 5.8%，limitBW 为 4.3%，4C 为 3.0%；另外两个智能体分别达到 2.0% 和 3.1%）；相比之下，我们的全尺寸运行达到 6.3% 的总体几何平均值（fullBW 为 9.1%，limitBW 为 5.9%，4C 为 3.9%）。

5.3 哪个级别重要

为量化各缓存层级的贡献，以及在缓存容量减半时，演化后的一级数据缓存、二级缓存和末级缓存模块是独立运行，还是依赖跨级协调，我们评估了 DuoArch-Nano 在 50% L2 + 50% LLC 处理器上的逐级组合，并与同尺寸基线进行比较（图 10）。四核柱状图采用 14 种搜索组合；L1D + L2 limitBW 柱使用了 610 条跟踪记录中的 608 条，因为两条图 PageRank 跟踪记录的作业运行时间超过了 12 小时。在单核工作负载上，L1D 和 L2 的收益几乎可以相加，

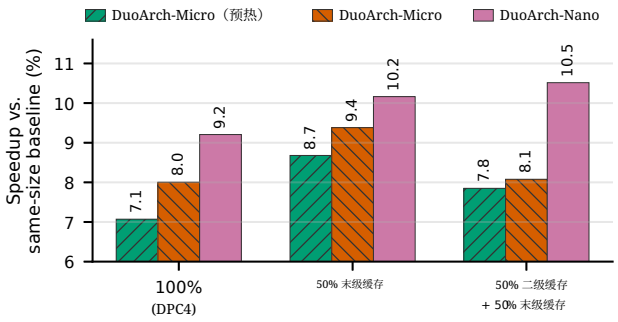


图 11：同一台机器上，三种小缓存设计相对于基线的总体官方加速比（几何平均值）。

不过，两者的相对重要性会随可用内存带宽而变化。在全带宽下，重构后的逐 PC L1D 带来了大部分加速比（单独使用时为 5.7%，而 L2 单独使用时为 3.5%）。在 limitBW 下，污染门控 L2 单独带来的收益更大（4.9%，而 L1D 为 2.6%），因为它减少了不准确的 Pythia 流量，并在 1MB L2 中保留了页表缓存行。组合使用演化后的 L1D 和 L2，在全带宽下可带来 9.5% 的收益，在 limitBW 下可带来 7.7% 的收益，与各自单独使用时收益之和（分别为 9.2% 和 7.5%）非常接近；再加入 LLC 时间相关器，则可额外提升 ~1%，使收益达到 10.7% 和 8.8%。然而，在四核执行（4C）下，各级之间表现出显著的跨层协同效应：单独使用任何一级的收益都不超过 3.4%；组合 L1D 和 L2 可达到 6.6%，而加入 LLC 模块后，四核加速比提升至原来的两倍以上，达到 14.0%，这是通过协调跨级攻击者退避，并在发生争用的 6 MB LLC 中保护受害者缓存行实现的。

5.4 资源匮乏是否能孕育强大?

为了检验 DuoArch-Nano 的强大表现究竟是否源于在严重缓存稀缺条件下进行演化，还是说在限制较少的小缓存环境中也能取得同样的结果，我们在全部三种处理器配置上，将 DuoArch-Nano 与在 50% LLC 机器（2MB L2、6 MB 共享 LLC）上演化出的两种方案进行比较。**DuoArch-Micro（预热）**以一个先前使用全尺寸缓存演化出的设计作为预热启动种子 [1], , 而**DuoArch-Micro** 则从未经修改的基线开始演化（官方评测见图11）。

从基线冷启动优于预热启动，并且达到 LLC 容量翻倍时的基线表现。首先，演化小缓存预取器并不需要预热启动种子。在 50% LLC 机器上，从未经修改的基线冷启动（DuoArch-Micro）在搜索集上的表现几乎赶上预热启动方案（DuoArch-Micro（预热）），并在官方评估集上略胜一筹（图11）。DuoArch-Micro 相比同尺寸基线取得 9.38% 的加速比，相比全尺寸基线则取得 –0.02% 的加速比（+3.56%（全带宽）、 –1.09%（limitBW），以及 –2.42%（四核））；而 DuoArch-Micro（预热）相比同尺寸基线取得 8.68% 的加速比（相比全尺寸基线为–0.68%）。即使尚未缩减二级缓存，从基线直接演化也能弥补 LLC 容量减半造成的总体每周期指令数损失。

更紧凑的缓存约束能让各种规模的机器都拥有更强的预取器。其次，与其针对小缓存过度专门化预取器，不如在更严格的缓存容量限制下进行演化，这能提升三种机器上的总体性能（图11）。DuoArch-Nano 在搜索集上的起点表现最差（–14.6%，而全尺寸基线），随后升至 –4.1%，弥补了搜索集损失的 72%。然而，其设计在每种机器上总体均优于两种 50% 末级缓存配置。在两者均减半的处理器上，其性能比同尺寸基线高 10.52%，而 DuoArch-Micro 高 8.08%，DuoArch-Micro（预热）高 7.85%。在各自的 50% 末级缓存适用场景下，其性能比同尺寸基线高 10.16%，比全尺寸基线高 +0.88%（四核高出 1.5%），领先于 DuoArch-Micro 的 –0.02% 和 DuoArch-Micro（预热）的 –0.68%。使用全尺寸缓存时，两种 50% 末级缓存配置的性能分别比全尺寸基线高 7.07%（DuoArch-Micro（预热））和 8.00%（DuoArch-Micro），而 DuoArch-Nano 高 9.21%。

为什么更紧凑的缓存能造就更强的预取器。在全尺寸缓存（2MB L2 和 12MB 共享末级缓存）上，充裕的静态随机存取存储器掩盖了预取低效：准确率一般的预取或缺乏协调的四核突发流量，都能由多余的缓存路吸收，而不会触发抖动，因此搜索会停滞在局部最优。缩小缓存会消除这层缓冲，使每个浪费的缓存槽位或未受限制的激进核心都立即导致容量驱逐，并带来四核调和平均值惩罚。因此，对两次 50% 末级缓存运行中的各阶段增益进行分解后可以发现，基础预测器就位后，大部分增益来自四核得分，也就是说，增益来自解决核间竞争，而非提高单核激进性。

50% 末级缓存运行构建了什么。从没有设计提示的基线出发，DuoArch-Micro（与 DuoArch-Nano 一样，§3）首先在所有三个级别上收敛到已知的预测器系列（+6首个阶段的搜索增益为 .18%）：一个按指令指针的及时增量一级数据缓存预取器、一个配有感知器有用性过滤器的二级缓存 Pythia，以及一个带有空间足迹的 256KB LLC 时间相关性表 [2, 4, 30, 34], , 之后又加入了存储流重构。在这个初始阶段之后，DuoArch-Micro 阶段性增益的 56–100% 以及 DuoArch-Micro（预热）早期阶段性增益的 83–100% 都来自四核得分：

- **代码压力降级与每核后进先出队列（DuoArch-Micro）：**由于指令取指不会调用预取器钩子，采样二级缓存组会跟踪仅数据影子，并在数据填入驱逐一个未见过的地址时推断发生了指令行驱逐，从而在代码压力下将较弱的数据预取降级为仅预取至末级缓存 [23, 31]（这是一个 +1.00% 搜索阶段的一部分，该阶段还加入了存储流重构；此前的公平性调优阶段增益为 +1.91%）。在共享末级缓存处，后续阶段（搜索总体增益为 +0.54%，4C 搜索分数增益为 +0.92%，同时加入页遍历行保留器）将 4C 上的先进先出发射改为每核最新优先（后进先出）栈；当末级缓存预取队列中的条目少于 4 个时，按轮询顺序从栈中取出预取，从而避免突发请求使共运行任务饥饿。

- **三级防掉队者调节器（DuoArch-Micro（预热））：**鉴于 4C 调和平均分数主要由最慢核心决定，DuoArch-Micro（预热）保留了初始设计，并在全部三个级别加入协同 QoS 调节器 [8]：在末级缓存处，4C 性能提升 +0.5%；在一级数据缓存处，4C 性能提升 +2.3%（当领先者/掉队者退休指令比超过 24/8 时，为最快的 3 个核心限制前瞻）；在二级缓存处，4C 性能提升 +1.3%；整步增益分别为 +0.75% 和 +0.46%。

DuoArch-Nano 的机制为何更进一步。 虽然三种小缓存运行都会保留页表缓存行 [19]（在全尺寸配置下达到相近的单核二级缓存页遍历缺失，即每千条指令 0.51–0.54 次），并限制四核竞争，但它们管理共享末级缓存的方式各不相同。具体而言，DuoArch-Micro（预热）的调节器只会在快速核心领先时降低其速度，而 DuoArch-Micro

只会限制预取激进程度，并按照最新优先的顺序安排每个核心的末级缓存预取；二者都不会直接保护最慢受害核心驻留在末级缓存中的缓存行。面对前置于 6 MB 共享末级缓存、容量减半为 1 MB 的二级缓存，DuoArch-Nano 构建了类似缓存分区和插入控制的机制 [27, 36]，无需修改替换硬件即可直接保护这些缓存行（§3）。具体而言，它通过借鉴相关性表的两种方式，即预取命中刷新，将最慢核心的末级缓存命中固定下来；引导 DRRIP 插入，使流式核心的填充首先被驱逐；并限制嘈杂共运行任务向共享相关表的写入。因此，DuoArch-Nano 相对于 DuoArch-Micro 的全尺寸优势几乎全部来自四核场景。在全尺寸机器上，DuoArch-Nano 在 483 个四核混合负载中的 327 个上优于 DuoArch-Micro，使四核 SPEC17 从 +13.9% 提升至 +21.6%，并将四核 Graph 从 −1.9% 的损失扭转为 +6.1% 的增益。同时，它的四核末级缓存需求未命中减少了 14%，每条指令发出的 DRAM 请求减少了 17%（比基线少 11%）；各混合工作负载的每周期指令数增益与末级缓存未命中减少幅度高度吻合（斯皮尔曼 $\rho = 0.69$）。

5.5 差距仍在哪里

Fig. 12 将全尺寸和 50% 末级缓存的结果按套件进行细分。为什么 evolution 能在 50% 末级缓存下完全弥合总体容量差距，但在两个缓存都减半时，仍然比全尺寸基线低 −6.20%？将官方结果按工作负载套件细分后，可以看出常规工作负载（SPEC17 和人工智能/机器学习）与不规则或大工作集工作负载（Graph 和 Google）之间存在明显分化。末级缓存减半后，原本能够在片上复用的工作集变成了容量未命中，其不规则访问模式超出了 256 KB 片上预取器的覆盖范围：在各条跟踪记录中，因缩小缓存而造成的每周期指令数损失与末级缓存未命中数的增加高度一致（斯皮尔曼 $\rho = 0.92$–0.94，在 Google 和 Graph 内）。在单核运行且末级缓存为 50% 时，两个 50% 末级缓存运行配置中的预取器消除了 SPEC17 和人工智能/机器学习中 100% 的额外容量未命中，在 Google 中消除了 46–82%（使 DuoArch-Micro 在 +0.2% 的 fullBW Google 上达到与全尺寸基线相当的水平），在 Graph 中消除了 37–55%。然而，在四核执行（四核）下，共享末级缓存减半后共同运行的核心之间会产生竞争，使 Graph 的额外容量未命中放大八倍，并使原本已受队列限制的内存总线上的 DRAM 流量几乎翻倍；当两个缓存都减半时（图 12中未显示），相对于全尺寸基线，DuoArch-Nano 在 −17.0% 的四核 Graph 和 −13.0% 的四核 Google 上仍能完成执行。

6 相关工作

数据预取器。空间预取器会学习程序访问某个区域中哪些缓存行，依据可以是区域足迹 [4, 30] 或指令指针类别 [26]。偏移量与增量预取器会学习预取距离，可以采用全局方式 [21],，也可以针对每条加载指令 [24],，或将页内增量特征作为签名并通过前瞻进行推测性跟踪 [16]；最新的 Caerus [32], 会学习多个全局偏移量，并针对每条加载指令进行过滤。在其自身的 SPECspeed 2017 配置中，该方法报告的成绩领先 BertiGO 1.5%（将作者的代码移植到 DPC4，并按照 DPC4 规则移除其专用标记阵列过滤器，将其作为仅用于一级数据缓存的 Berti 替代方案后，与基线相比总体提升 1.0%）。时间预取器会重放记录的缺失序列 [2, 13,34],，而 Pythia [5] 则通过在线强化学习选择预取操作学习。这些设计中有几种赢得了数据预取锦标赛，该赛事在 ChampSim 中比较预取器 [10]并采用固定存储预算：经过调优的 BOP 赢得了 DPC-2，IPCP 赢得了 DPC-3，而基于 Berti 构建的 BertiGO [29], 赢得了 DPC4 [7],；其 limitBW 和多核配置会惩罚带宽需求高的设计。**预取节流、共享缓存管理与缓存容量配置**。为防止激进预取耗尽内存带宽或污染缓存，先前的研究会依据运行时准确率、污染情况和带宽反馈来调节预取度 [8, 31]，或在内存控制器处调度预取 [18]。其他方案则修改缓存替换和分区策略，以遏制预取污染 [28, 33],，在核心之间划分或伪划分共享末级缓存容量 [27, 36],，保留指令缓存行 [23],，或在数据与地址转换之间划分缓存空间 [19],；而 Pythia 等研究 [5] 则评估不同末级缓存容量下的固定预取器。DuoArch-Nano 并不修改缓存替换硬件，也不只在较小缓存上评估设计，而是在未修改的 DRRIP 下运行 [14]，并通过标准预取器接口工作。我们的实验发现，在缓存资源匮乏的情况下进化预取器，在我们测试的三种缓存容量下都能产生总体上更强的设计（§5.4）。

用于硬件设计的大语言模型智能体。以往由大语言模型驱动的架构研究，要么针对其他结构，例如缓存替换 [12, 20], 参数级设计空间探索 [9, 37],；要么在 DPC4 设置之外演进单级预取器，且未提供代码 [17]。因此，我们将其与 ArchAgent v2 [11], 进行比较；后者是 DPC4 上最先进的智能体系统，基于其前代系统 [12, 25]构建，并一次演进一个缓存层级的 DPC4 预取器。

7 结论

我们介绍了**DuoArchitect**，这是一种多保真度自主发现系统，能够克服周期级仿真的瓶颈，协同演进三级数据预取器。DuoArchitect 发现了 **DuoArch-Nano**，这是一种在二级缓存和末级缓存容量减半的条件下演进出的智能预取器。在 DPC4 基准测试上评估时，DuoArch-Nano 在全尺寸缓存下实现了 9.2% 的总体加速比，增益超过 BertiGO（3.5%）和 ArchAgent v2（3.9%）的两倍。移除一半末级缓存（1.5MB/核心）时，DuoArch-Nano 消除了基线的 8.6% 性能下降，并优于全缓存基线（+0.88%），这表明微架构智能能够直接替代物理缓存容量。

回收这些缓存硅片面积可直接提升计算密度。例如，在 AMD Zen 4 核心复合体中（32 MB 三级缓存占八核复合体的 44%），将每核 LLC 减半可腾出足够的硅片面积，容纳约 20% 更多的物理核心。以预取器驱动的主动智能取代被动的缓存容量，使硬件设计人员能够将面积性能帕累托前沿推至技术缩放极限之外。

更广泛地说，这些结果预示着智能体计算机架构领域的一场范式转变。我们表明，基于大语言模型的智能体并非只是设计自动化助手，还能够驾驭复杂的多维权衡，而这类权衡的规模通常远超人类工程师的推理能力。通过共同演进策略及其自身的多保真度仿真环境，智能体框架得以用智能化的软件驱动微架构取代庞大且扩展受阻的物理结构。

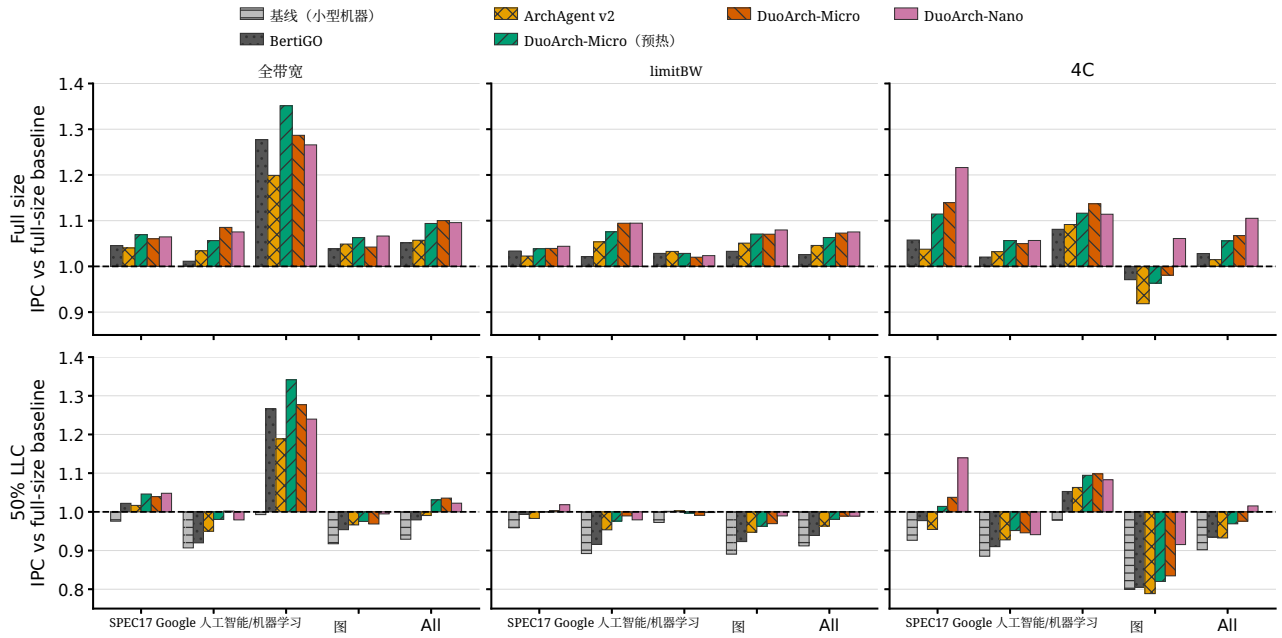


图12：全带宽、limitBW 和四核配置下，各套件相对于全尺寸基线的官方每周期指令数。

致谢

Gemini 4 Neon 曾用于协助撰写、编写绘图脚本和核对数字；作者审核了所有内容，并对此承担全部责任。

参考文献

[1] Shubham Agarwal、Akshat Gupta、Mike Heddes、Dilara Soylu、Rui Meng、Patrick Tser Jern Kon、Jiefeng Chen、Mert Cemri、Shu Liu、Abe Gonzalez、Akanksha Jain、Bhavana Dalvi Mishra、Linfeng Song、Vilobh Meshram、Xiangru Tang、Jinsung Yoon、Aditya Parameswaran、Ion Stoica、Vahab Mirrokni、Burak Gokturk、Parthasarathy Ranganathan、Tomas Pfister、Nanyun Peng 和 Chun-Liang Li。2026。DuoDiscover：突破评估瓶颈，扩展自主发现。arXiv: 2610 17184 。https://arxiv.org/abs/ 2610 17184 。[2] Sam Ainsworth 和 Lev Mukhanov。 2024 。Triangel：高性能、精准且及时的片上时序预取器。载于第 51 届年度国际计算机体系结构研讨会（ISCA）论文集。1202–1216。doi:10.1109/ISCA59077.2024.00090[3] 亚马逊云科技。2026。AWS Graviton 技术指南。https://github.com/aws/aws-graviton-getting-started。访问日期：2026-11-15。[4] Mohammad Bakhshalipour、Mehran Shakerinava、Pejman Lotfi-Kamran 和 Hamid Sarbazi-Azad。2019。Bingo 空间数据预取器。载于 IEEE 高性能计算机体系结构国际研讨会（HPCA）论文集。 399 411 10 1109 2019 00053 – 。doi: 。/ HPCA。 。[5] Rahul Bera、Konstantinos Kanellopoulos、Anant V. Nori、Taha Shahroodi、Sreenivas Subramoney 和 Onur Mutlu。2021。Pythia：使用在线强化学习的可定制硬件预取框架。载于第 54 届年度 IEEE/ACM 微体系结构国际研讨会（MICRO）论文集。1121–1137。doi:10.1145/3466752.3480114[6] Thomas Burd、Srividhya Venkataraman、Wilson Li、Timothy Johnson、Jerry Lee、Srikirti Velaga、Mark Wasio、Thomas Yiu、Franklin Bodine、Michael McCabe、Udin Salim、Santosh Kumar Thouta、Michael Golden、Sowmya Ramachandran、Gokul Subramani Lakshmi Devi、John Wuu、Yarek Kuszczak、Gaurav Singla、Carson Henrion、Andy Robison、Sabeesh Balagangadharan、Umesh Nair、Naveen Srivastava、Hari Prasad、Mohini Polimetla、Phaneendra Chennupati、Eshwar Gupta、Mahesh Vykuntam、Sumantra Sarkar、Praveen Kumar Duvvuru、Theja Mardi 和 G Swetha。2024。“Zen 4c”：AMD 5纳米面积优化型 x86-64 微处理器核心。载于 2024 IEEE 国际固态电路会议（ISSCC）。IEEE, 38 40 10 1109 49657 2024 10454507 – 。doi: 。/ISSCC 。 。DPC 。[7] 4 2026 4 第 th 届数据预取冠军赛（DPC4）。https: //sites.google.com/view/dpc4-2026/home。与第 32 届 IEEE 高性能计算机体系结构国际研讨会（HPCA）联合举办。[8] Eiman Ebrahimi、Onur Mutlu、Chang Joo Lee 和 Yale N. Patt。2009。多核系统中的多预取器协调控制。载于第 42 届年度 IEEE/ACM 微体系结构国际研讨会（MICRO）论文集。316–326。doi:10.1145/1669112.1669154[9] Zuoming Fu、Alex Manley 和 Mohammad Alian。2025。gem5 Co-Pilot：用于体系结构设计空间探索的人工智能助理代理。载于第 3 届计算机体系结构建模与仿真研讨会（CAMS）。

计算机体系结构建模与仿真（CAMS）。arXiv：2510.19577。[10] Nathan Gober、Gino Chacon、Lei Wang、Paul V. Gratz、Daniel A. Jiménez、Elvira Teran、Seth Pugsley 和 Jinchun Kim。2022。冠军赛模拟器：面向教育与竞赛的体系结构仿真。arXiv：2210.14324 [cs.AR] https://arxiv.org/abs/2210 14324 。Abraham Gonzalez、Raghav Gupta、Akanksha Jain、Hanna Alam、Alexan- der Novikov、Po-Sen Huang、Matej Balog、Marvin Eisenberger、Sergey Shirobokov、Ngân V˜u、Hank Levy、Borivoje Nikolić、Sagar Karandikar、Mar- tin Dixon 和 Parthasarathy Ranganathan。2026。ArchAgent v2：数据预取冠军赛案例研究。arXiv：2608.09874 [cs.AI] https://arxiv.org/abs/2608 09874 。Raghav Gupta、Akanksha Jain、Abraham Gonzalez、Alexander Novikov、Po-Sen Huang、Matej Balog、Marvin Eisenberger、Sergey Shirobokov、Ngân V˜u、Martin Dixon、Borivoje Nikolić、Parthasarathy Ranganathan 和 Sagar Karandikar。2026。ArchAgent：智能体人工智能驱动的计算机体系结构发现。arXiv：2602 22425 [ ] 2602 22425 。cs.AI https://arxiv.org/abs/ 。Akanksha Jain 和 Calvin Lin。。不规则内存访问线性化，以改进相关预取。载于第 46 届年度 IEEE/ACM 微体系结构国际研讨会（MICRO）论文集。247–259。doi:10.1145/2540708.2540730[14] Aamer Jaleel、Kevin B. Theobald、Simon C. Steely Jr. 和 Joel S. Emer。2010。使用再引用间隔预测（RRIP）实现高性能缓存替换。载于第 37 届年度国际计算机体系结构研讨会（ISCA）论文集。60 71 10 1145 1815961 1815971 – 。doi：。/。M. G. Kendall。。秩相关性的新度量。Biometrika 30，1/2（1938），81–93。[16] Jinchun Kim、Seth H. Pugsley、Paul V. Gratz、A. L. Narasimha Reddy、Chris Wilkerson 和 Zeshan Chishti。2016。基于路径置信度的前瞻预取。载于第 49 届年度 IEEE/ACM 微体系结构国际研讨会（MICRO）论文集。1–12。doi:10.1109/MICRO.2016.7783763[17] Vasilis Kypriotis、Alexander Blasberg、Alexandros Moiras 和 Dimitrios Skarlatos。2026。Agentic Architect：用于体系结构设计探索与优化的智能体人工智能框架。arXiv：2604.25083 [cs.AI] https://arxiv.org/abs/2604 25083 。Chang Joo Lee、Onur Mutlu、Veynu Narasiman 和 Yale N. Patt。2008。预取感知型 DRAM 控制器。载于第 41 届 IEEE/ACM 微体系结构国际研讨会（MICRO）论文集。200 209 10 1109 2008 4771791 – 。doi：。/MICRO。。Yashwant Marathe、Nagendra Gulur、Jee Ho Ryoo、Shuang Song 和 Lizy K. John。2017。CSALT：上下文切换感知型大型TLB。载于第 50 届年度 IEEE/ACM 微体系结构国际研讨会（MICRO）论文集。449–462。doi:10.1145/3123939.3124549[20] Kaushal Mhapsekar、Bita Aslrousta、Brijesh Kumar Bhayana、Paula Contreras、Azam Ghanbari、Ethan Goodman、Anna Andriiko 和 Samira Mirbagher Ajorpaz。2026。CACHEFORGE：大语言模型引导的端到端生成式缓存替换策略，面向性能与硬件效率。arXiv：2610.07668 [cs.AR] https://arxiv.org/abs/2610 07668 。Pierre Michaud。。最佳偏移硬件预取。载于 IEEE 高性能计算机体系结构国际研讨会（HPCA）论文集。469–480。doi:10.1109/HPCA.2016.7446087[22] Benjamin Munger、Kathy Wilcox、Jeshuah Sniderman、Chuck Tung、Brett Johnson、Russell Schreiber、Carson Henrion、Kevin Gillespie、Tom Burd、Harry Fair、David Johnson、Jonathan White、Scott McLelland、Steven Bakke、Javin Olson、Ryan McCracken、Matthew Pickett、Aaron Horiuchi、Hien Nguyen 和 Tim H Jackson。2023。“Zen 4”：AMD 5纳米5.7GHz x86-64微处理器核心。载于 2023 IEEE 国际固态电路会议（ISSCC）。IEEE，38–39。doi:10.1109/ISSCC42615.2023.10067540[23] Nayana Prasad Nagendra、Bhargav Reddy Godala、Ishita Chaturvedi、Atmn Patel、Svilen Kanev、Tipp Moseley、Jared Stark、Gilles A. Pokam、Simone Campanoni 和 David I. August。2023。EMISSARY：用于 L2指令缓存的增强型缺失感知替换策略。载于第 50 届年度国际计算机体系结构研讨会（ISCA）论文集。1 13 10 1145 3579371 3589097 – 。doi：。/。Agustín Navarro-Torres、Biswabandan Panda、Jesús Alastruey-Benedé、Pablo Ibáñez、Víctor Viñals-Yúfera 和 Alberto Ros。2022。Berti：精准的局部增量数据预取器。载于第 55 届 IEEE/ACM 微体系结构国际研讨会（MICRO）论文集。975– 991。doi:10.1109/MICRO56248.2022.00072[25] Alexander Novikov、Ngân V˜u、Marvin Eisenberger、Emilien Dupont、Po-Sen Huang、Adam Zsolt Wagner、Sergey Shirobokov、Borislav Kozlovskii、Francisco J. R. Ruiz、Abbas Mehrabian、M. Pawan Kumar、Abigail See、Swarat Chaudhuri、George Holland、Alex Davies、Sebastian Nowozin、Pushmeet Kohli 和 Matej Balog. 2025. AlphaEvolve：用于科学与算法发现的编程代理。arXiv：2506 13131 [ ] 2506 13131 . cs.AI https://arxiv.org/abs/ 。Samuel Pakalapati 和 Biswabandan Panda. 。指令指针花束：基于指令指针分类器的空间硬件预取。载于第47届年度国际计算机体系结构研讨会（ISCA）论文集。118 131 10 1109 45697 2020 00021 – 。doi: 。/ISCA 。 。Moinuddin K. Qureshi 和 Yale N. Patt. 。基于效用的缓存分区：一种低开销、高性能的共享缓存运行时分区机制。载于第39届年度IEEE/ACM微体系结构国际研讨会（MICRO）论文集。423 432 10 1109 2006 49 – 。doi: 。/MICRO. 。Vivek Seshadri、Samihan Yedkar、Hongyi Xin、Onur Mutlu、Phillip B. Gibbons、Michael A. Kozuch 和 Todd C. Mowry. 2015. 使用针对预取数据块的信息驱动缓存策略缓解预取器导致的污染。《ACM体系结构与代码优化汇刊》11, 4 (2015), 51:1–51:22. doi:10.1145/2677956[29] Simranjit Singh、Agustín Navarro-Torres 和 Alberto Ros. 2026. Berti预取器的极限探索。载于第四届数据预取锦标赛（DPC-4），与IEEE高性能计算机体系结构国际研讨会（HPCA）同期举行。https://webs.um.es/aros/papers/pdfs/ssingh-dpc26.pdf[30] Stephen Somogyi、Thomas F. Wenisch、Anastassia Ailamaki、Babak Falsafi 和 Andreas Moshovos. 2006. 空间内存流。载于第33届年度国际计算机体系结构研讨会（ISCA）论文集。252–263. doi:10.1109/ISCA.2006.38[31] Santhosh Srinath、Onur Mutlu、Hyesoon Kim 和 Yale N. Patt. 2007. 反馈引导预取：提升硬件预取器的性能与带宽效率。载于第13届高性能计算机体系结构国际会议（HPCA）论文集。IEEE计算机学会，63–74. doi:10.1109/HPCA.2007.346185[32] Jacky Wong、Jacob Alexandrou、Yinting Huang、Luke Panayi、Lev Mukhanov、Sam Ainsworth 和 Paul H. J. Kelly. 2026. Caerus：全局思考，局部行动的预取。arXiv：2610 39797 [ ] 2610 39797 . cs.AR https://arxiv.org/abs/ 。Carole-Jean Wu、Aamer Jaleel、Margaret Martonosi、Si mon C. Steely Jr. 和 Joel S. Emer. 2011. PACMan：面向高性能缓存的预取感知缓存管理。载于第44届年度IEEE/ACM微体系结构国际研讨会（MICRO）论文集。442–453. doi:10.1145/2155620.2155672[34] Hao Wu、Krishnendra Nathella、Joseph Pusdesris、Dam Sunwoo、Akanksha Jain 和 Calvin Lin. 2019. 无需片外元数据的时间预取。载于第52届年度IEEE/ACM微体系结构国际研讨会（MICRO）论文集。996 1008 10 1145 3352460 3358300 – 。doi: 。/。Shien-Yang Wu、C.H. Chang、M.C. Chiang、C.Y. Lin、J.J. Liaw、J.Y. Cheng、J.Y. Yeh、H.F. Chen、S.Y. Chang、K.T. Lai、M.S. Liang、K.H. Pan、J.H. Chen、V.S. Chang、T.C. Luo、X. Wang、Y.S. Mor、C.I. Lin、S.H. Wang、M.Y. Hsieh、C.Y. Chen、B.F. Wu、C.J. Lin、C.S. Liang、C.P. Tsao、C.T. Li、C.H. Chen、C.H. Hsieh、H.H. Liu、P.N. Chen、C.C. Chen、R. Chen、Y.C. Yeo、C.O. Chui、W. Chang、T.L. Lee、K.B. Huang、H.J. Lin、K.W. Chen、M.H. Tsai、K.S. Chen、X.M. Chen、Y.K. Cheng、C.H. Wang、W. Shue、Y. Ku、S. M. Jang、M. Cao、L.C. Lu 和 T.S. Chang. 2022. 一种3纳米CMOS FinFlexTM平台技术，具备更高的功耗效率和性能，适用于移动SoC与高性能计算应用。载于2022年国际电子器件会议（IEDM）。IEEE，27 5 1 27 5 4 10 1109 45625 2022 10019498 . 。 – 。 。 。doi: 。/IEDM 。 。Yuejian Xie 和 Gabriel H. Loh. 。PIPP：多核共享缓存的提升/插入伪分区。载于第36届年度国际计算机体系结构研讨会（ISCA）论文集。174–183. doi:10.1145/1555754.1555778[37] Jia Xiong、Runkai Li、Chenxu Niu、Guangyuan Gao、Changwen Xing、Yifan Zhang、Xinlai Wan、Jieran Cui、Chen Bai、Yusheng Hua、Ying Wang、Ming Ling、Xi Wang 和 Tao Xie. 2026. MicroEvo：用于高效微体系结构设计空间探索的知识引导大语言模型采样。载于IEEE/ACM计算机辅助设计国际会议（ICCAD）论文集。arXiv:2608.06183. [38] Geoffrey Yeap、S. S. Lin、Y. M. Chen、H. L. Shang、P. W. Wang、H. C. Lin、Y. C. Peng、J. Y. Sheu、M. Wang、X. Chen、B. R. Yang、C. P. Lin、F. C. Yang、Y. K. Leung、D. W. Lin、C. P. Chen、K. F. Yu、D. H. Chen、C. Y. Chang、H. K. Chen、P. Hung、C. S. Hou、Y. K. Cheng、J. Chang、L. Yuan、C. K. Lin、C. C. Chen、Y. C. Yeo、M. H. Tsai、H. T. Lin、C. O. Chui、K. B. Huang、W. Chang、H. J. Lin、K. W. Chen、R. Chen、S. H. Sun、Q. Fu、H. T. Yang、H. T. Chiang、C. C. Yeh、T. L. Lee、C. H. Wang、S. L. Shue、C. W. Wu、R. Lu、W. R. Lin、J. Wu、F. Lai、Y. H. Wu、B. Z. Tien、Y. C. Huang、L. C. Lu、Jun He、Y. Ku、J. Lin、M. Cao、T. S. Chang 和 S. M. Jang. 2019. 5纳米CMOS生产技术平台，采用成熟的极紫外光刻技术，并配备具有最高密度0.021m^2 的高迁移率沟道FinFET SRAM单元，适用于移动SoC与高性能计算应用。载于2019年IEEE国际电子器件会议（IEDM）。IEEE，36.7.1–36.7.4. doi:10.1109/IEDM19573.2019.8993577