Cognitive SSD: A Deep Learning Engine for In-Storage Data Retrieval

摘要

数据分析和检索是现有人工智能系统中广泛使用的组件。然而，每个请求都必须通过 I/O 堆栈的每个层次，将大量无关的数据在二级存储、DRAM 和芯片内缓存之间移动。这导致响应延迟高，能耗不断上升。为了解决这个问题，文章提出了认知 SSD，这是一种用于基于深度学习的非结构化数据检索的高能效引擎。在认知 SSD 中，一个名为 DLG-x 的闪存访问加速器被放置在闪存旁边，实现了近数据的深度学习和图搜索。这种在 SSD 内部进行深度学习和图搜索的功能通过扩展 NVMe 命令并且以库 API 的形式暴露给用户。基于 FPGA 的原型的实验结果显示，所提出的认知 SSD 与基于 CPU 的传统 SSD 解决方案相比，平均减少了 69.9% 的延迟，并且与提供相当性能的基于 CPU 和 GPU 的解决方案相比，分别减少了 34.4% 和 63.0% 的整体系统功耗。最终我复现了此篇文章，并使用另一款神经网络实现了更低的平均延迟，以及更低的整体功耗。

关键词： SSD；近数据处理；深度学习加速器

1 引言

近年来，非结构化数据，特别是视频和图像等，呈爆发式增长。据报道，商业数据中心中非结构化数据占据了高达 80% 的存储容量。海量非结构化数据一旦在云中存储和管理，就会导致用户发出密集的检索请求，这对数据中心的处理吞吐量和功耗构成重大挑。因此，在云服务基础设施中支持快速、节能的数据检索对于降低数据中心的总拥有成本 (TCO) 至关重要。

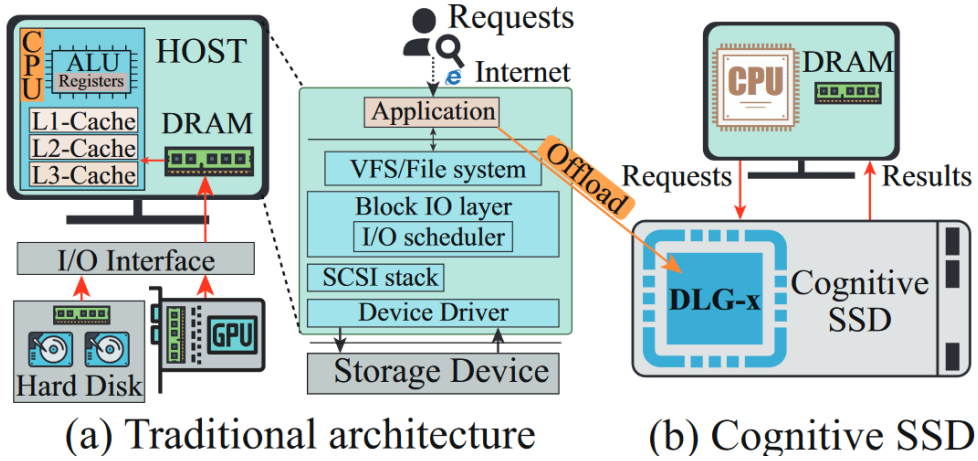

图 1. 传统架构 (a) vs. 可认知的 SSD(b)

不幸的是，传统的基于内容的多媒体数据检索系统存在不准确、功率低效和成本高的问题，尤其是对于大规模非结构化数据。图 1(b) 简要描述了一个典型的基于内容的数据检索系统，由 CPU/GPU 和传统存储设备组成，基于以计算为中心的架构 [5]。当数据检索请求从互联网或中央服务器到达时,CPU 必须将大量潜在数据从磁盘重新加载到临时 DRAM 中，并将查询特征与加载的非结构化数据进行匹配，以找到相关目标。这种以计算为中心的架构面临着几个关键的开销和低效率来源。(1) 当前的 I/O 软件堆栈简单地根据检索请求从存储设备获取数据时，会给数据检索系统带来很大的负担 [60]。情况甚至更糟，因为据报道，随着传统 HDD 被非易失性存储器取代，性能瓶颈已从硬件 (75-50us) 迁移到软件 (60.8us) [31,33]。(2) 在传统的内存层次结构中，大量数据移动会产生能量和延迟开销。随着查询数据规模的增加，这个问题变得更加严重，因为低级存储中的相关数据必须经过慢速 I/O 接口（例如 SATA）、主内存和多级缓存，然后才能到达计算单元 CPU 或 GPU [3]，如图 1(a) 所示。

为了解决这些问题，如 1(b) 所示，本工作旨在在紧凑型存储设备内定制统一的数据存储和检索系统，并消除主要的 IO 和数据移动瓶颈。在该系统中，检索请求直接发送到存储设备，目标数据的分析和索引完全在非结构化数据所在的地方进行。基于所提出的认知 SSD 构建这样的数据检索系统具有以下设计目标：(1) 提供紧凑型 SSD 中可承受的高精度、低延迟和节能的查询机制，(2) 利用闪存设备的内部带宽在 SSD 中进行基于节能深度学习的数据处理，(3) 使开发人员能够针对不同数据集定制数据检索系统。下面详细阐述这些要点。

首先，考虑到 SSD 的外形尺寸和成本，必须拥有计算效率高且准确的数据检索架构，而不是依赖图 1(a) 中的通用 CPU 或 GPU 设备。传统的数据检索框架不准确或计算成本太高，无法在资源受限的 SSD 中实现。在这项工作中，文章首次提出了一种结合深度学习和图搜索算法（DLG）的整体数据检索机制，其中前者可以提取非结构化数据的语义特征，后者可以提高数据库搜索效率。DLG 解决方案实现了更高的数据检索精度，并通过深度学习模型定制实现了用户可定义的计算复杂度，使得在 SSD 中实现灵活高效的端到端非结构化数据检索系统成为可能。

其次，虽然 DLG 是一种简单而灵活的端到端数据检索解决方案，但将其嵌入 SSD 仍然需要相当大的努力。文章设计了一种同时支持深度哈希和图搜索的特定硬件加速器 DLG-x，以构建目标认知 SSD，而无需使用电力不可持续的 CPU 或 GPU 解决方案。然而，SSD 内有限的 DRAM 主要用于缓存闪存管理的元数据，没有为深度学习应用程序留下可用空间。幸运的是，文章已经证明，在典型的 SSD 中，内部闪存接口的带宽超过了外部 IO 接口的带宽，这通过适当的数据布局映射满足了 DLG-x 的带宽需求。通过重建 SSD 中的数据路径，并刻意优化 NAND 闪存上与深度学习模型和图形相关的数据布局，DLG-x 可以充分利用内部并行性，绕过板载 DRAM 直接访问 NAND 闪存中的数据。

最后，当文章将深度学习技术引入 SSD 时，必须将认知 SSD 的软件抽象暴露给用户和开发人员，以使用不同的深度学习模型处理不同的数据结构。因此，文章利用 NVMe 协议进行命令扩展，将底层的深度学习机制、特征分析和数据结构索引机制抽象为用户可见的调用。用户的请求不仅可以触发 DLG-x 加速器在目标数据集中搜索查询相关的结构，而且系统开发人员可以针对不同的数据集和性能要求自由配置具有不同表示能力和开销的深度哈希架构。与传统的临时解决方案相比，认知 SSD 允许系统开发人员通过提供的 API 来调整检索精度以及数据检索服务的实时性能。同时，认知 SSD 还支持灵活组合特殊命令来实现不同的数据检索相关任务，例如存储内数据分类和仅哈希函数。

2 相关工作

2.1 非结构化数据检索系统

基于内容的非结构化数据检索系统旨在通过分析大规模数据集中的视觉或音频内容来搜索某些数据条目。图 2描述了典型的基于内容的检索过程，由两个主要阶段组成：特征提取和数据库索引。特征提取生成查询数据的特征向量，数据库索引在存储中搜索相似的数据结构，并在语义空间中编码该特征向量。

特征提取和深度学习：深度学习的兴起将研究的重点转移到基于深度卷积神经网络(DCNN) [19] 的特征，因为它提供了更好的中级表示 [11,29]。图描绘了一个典型的 DCNN，其中包含四种关键类型的网络层：(1) 卷积层，通过在组织为 3D 张量的输入数据上移动和卷积多维滤波器，从输入中提取视觉特征；(2) 激活、非线性文章对输入信号进行的变换，(3) 池化层，它对输入通道进行下采样以获得尺度和其他类型的不变性，以及 (4) 全连接 (FC) 层，它在特征和特征之间执行线性操作学习权重来预测输入数据的分类或其他高级特征。这种神经网络非常灵活，可以设计为具有不同的超参数，例如堆叠在一起的卷积层和池化层的数量以及卷积滤波器的维度和数量。更改这些参数将影响神经网络的泛化能力和计算开销，神经网络通常可以针对不同的数据集或应用场景进行定制。一些先前的工作直接采用 FC 层的高维输出向量进行数据检索，并且在内存占用和计算复杂性方面被认为过于昂贵。因此，文章采用深度哈希 [19] 来实现有效而简洁的数据表示。图 2举例说明了深度哈希架构 Hash-AlexNet，其中哈希层位于 AlexNet [2] 的最后一层之后，将从 AlexNet 学习到的数据特征投影到哈希空间中，生成的哈希码可以直接用于索引相关的数据结构并摆脱复杂的数据预处理阶段。

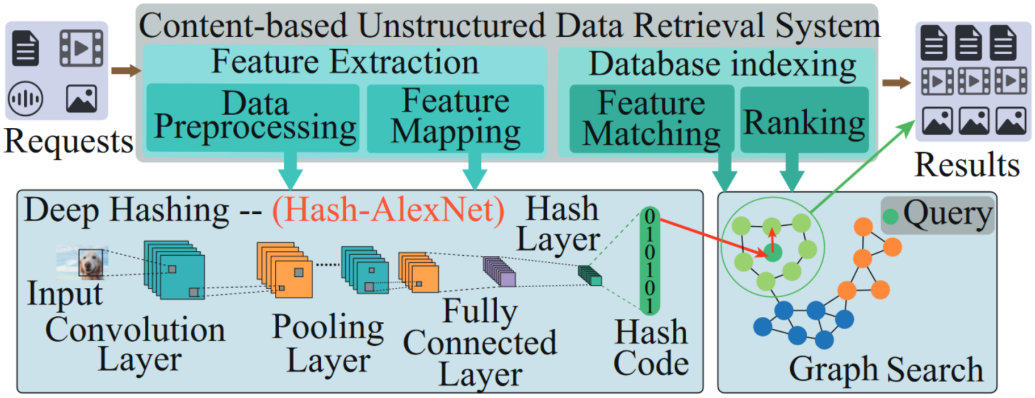

图 2. 基于内容的多媒体数据检索系统

数据库索引：基于图的近似最近邻搜索 (ANNS) 方法称为 NSG [10]，作为深度哈希的补充，实现了准确和快速的数据检索结果，正如之前的工作 [4,9] 所证明的那样。NSG 的主要思想是将查询哈希码映射到图中。图的顶点代表一个实例，边代表实体之间的相似度，边的值代表相似度的强弱。最重要的是，NSG 可以迭代地检查图中邻居的邻居，根据邻居的邻居也可能是邻居概念来找到查询的真正邻居。通过这种方式，NSG 可以避免不必要的数据检查，从而减少检索延迟。综上所述，与使用暴力搜索或哈希算法的传统解决方案相比，深度哈希加图搜索可以实现低延迟和高精度的检索性能。同时，由于相似的计算模式和数据流，它也使得检索框架更加紧凑和高效，从而可以适合紧凑且功率有限的 SSD。

2.2 近数据处理和深度学习加速器

对于软硬件协同设计，SSD 研究有两个方向：开放通道 SSD 和近数据处理（NDP）。虽然开放通道 SSD 可通过系统软件实现直接闪存访问 [15,16,22,32]，但近数据处理 (NDP) 将计算从系统的主处理器转移到内存或存储设备中 [1,7,8,14,21,23,27]。

在 NDP 中，Morpheus [12] 提供了一个框架，用于将计算转移到 NVMe SSD 上的通用嵌入式处理器。FAWN [6] 使用低功耗处理器和闪存来处理数据，并专注于键值存储系统。SmartSSD [30] 引入了智能 SSD 模型，它将设备内处理与强大的主机系统配对，能够在不修改操作系统的情况下处理面向数据的任务。 [18] 利用三星 SmartSSD 支持基本的数据库操作，包括排序、扫描和列表交集。[13] 通过模拟研究了在 SSD 中使用嵌入式 ARM 处理器来运行 SGD 的可能性，SGD 是神经网络训练的关键组成部分。然而，由于嵌入式处理器的性能限制，它们都无法处理深度学习处理。因此， [26] 提出了智能固态硬盘 (iSSD)，它将流处理器嵌入到闪存控制器中以处理线性回归和 kmeans 工作负载。 [17] 将可编程逻辑集成到 SSD 中，以实现网络规模数据分析的能效计算。同时， [25] 还使用 FPGA 构建 BlueDBM，该 BlueDBM 使用闪存存储和店内处理来对大型数据集进行经济高效的分析，例如图遍历和字符串搜索。GraFBoost [24] 专注于闪存内计算平台上图算法的加速，而不是像这项工作那样的深度学习算法。

认知 SSD：先前的主动磁盘要么集成了无法处理高吞吐量数据的通用处理器，要么集成了仅支持扫描和排序等简单功能的专用加速器。这些盘内计算引擎无法满足高吞吐量深度神经网络（DNN）推理的要求，因为计算密集型 DNN 在数据分析和查询任务中通常依赖于耗电的 CPU 或 GPU。为了实现节能 DNN，之前的工作提出了各种节能深度学习加速器。例如，Diannao 和 C-Brain 将大型 DNN 映射到矢量化处理阵列上，并采用数据平铺策略来利用神经参数的局部性 [20,28]。Eyeriss 将经典的脉动阵列架构应用于 CNN 的推理中，在能效方面大幅优于 CPU 和 GPU [34]。然而，这些研究的重点是优化加速器的内部结构，并依赖于大容量的 SRAM 或 DRAM 而不是外部非易失性存储器。与这些工作和之前的主动 SSD 设计相比，文章提出了认知 SSD，这是第一个使存储设备能够利用深度学习来进行存储内数据查询和分析的工作。它旨在取代传统的数据检索系统，并包含用于深度学习和图形搜索的闪存访问加速器（DLG-x）。DLG-x 被特意重新塑造，以利用大闪存容量和高内部带宽的优势，并且还重新架构以实现图形搜索到目标索引。

3 本文方法

3.1 本文方法概述

如图 2所示，这项工作结合了深度哈希和图搜索技术（DLG）的优点，在高精度的前提下降低了检索系统的复杂性，这使得将检索任务从 CPU/GPU 卸载到检索系统成为可能。资源约束认知 SSD。在此基础上，文章构建了一个端到端的数据检索系统，支持音频、视频、文本等多媒体数据检索。例如，音频可以通过认知 SSD 上的循环或卷积神经网络模型进行处理，以生成哈希码，该哈希码充当检索 SSD 内相关音频数据的索引。在本文中，图像检索被用作展示。如图 3所示，Cognitive SSD 旨在支持 DLG 框架中的主要组件，允许开发人员定制和实施数据检索解决方案。这样的近数据检索系统由两个主要组件组成：运行在管理用户请求

的轻量级服务器中的 DLG 库，以及通过 PCIe 接口插入主机服务器的认知 SSD。如表 1 所示，作为认知 SSD 系统的接口，DLG 库是利用 NVMe 协议 I/O 命令集中的 Vendor Specific Commands 建立的。它包含一个配置库和一个用户库。配置库使管理员能够根据应用需求在认知 SSD 上快速选择和部署不同的深度学习模型。在认知 SSD 上部署特征提取深度哈希模型后，到达主机服务器的数据处理请求可以通过调用用户库提供的 API 向其发送并建立查询会话。然后，认知 SSD 嵌入式处理器上的运行时系统接收并解析请求以激活与用户创建的会话关联的相应 DLG-x 模块。接下来文章详细阐述 Cognitive SSD 的软硬件设计细节。

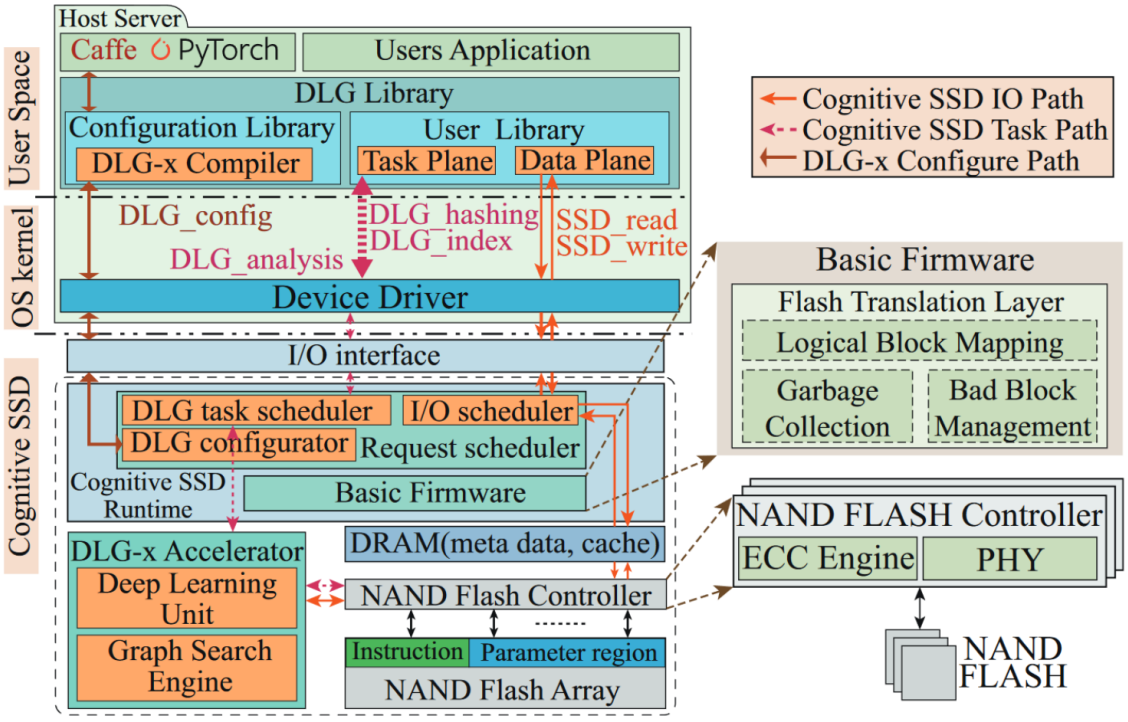

图 3. 可认知 SSD 的系统架构

3.2 可认知 SSD 的软件部分：DLG 库

更新深度学习模型：由于深度学习模型的选择会极大地影响数据检索系统的性能，因此系统管理员必须能够根据数据库的复杂性和容量以及通过响应衡量的服务质量来定制特定的深度哈希模型延迟或请求吞吐量。因此，配置库提供了与流行的深度学习框架（即 Caffe）兼容的 DLG-x 编译器，以允许管理员离线训练新的深度学习模型并生成相应的 DLG-x 指令。然后，管理员可以通过更新 DLG-x 指令来更新认知 SSD 上运行的学习模型。更新后的指令被发送到 NAND 闪存中分配的指令区域并停留在那里，直到发出模型更改命令（图 3中的 DLG_config）。同时，DLGx 编译器还根据神经网络模型和图的结构，重新组织 DLG 算法的数据布局，以充分利用内部闪存带宽，然后将深度学习模型和图的参数写入 NAND 闪存。权重和图结构信息的物理地址记录在 DLG-x 指令中。通过这种方式，DLG-x 在运行时直接获取所需数据的物理地址，而不是采用产生额外开销的地址转换或查找操作。

3.3 用户库

数据部分：数据平面提供 SSD_read 和 SSD_write API，供用户控制主机服务器和认知 SSD 之间的数据传输。这两个命令绕过闪存转换层直接对物理地址进行操作。用户可以调用这些 API，根据数据地址和数据大小参数，将用户发送的数据注入到认知 SSD 上的数据缓存区域或 NAND 闪存中。之后，用户可以使用这些地址来指向其他 API 中的操作数。任务部分：为了提高支持深度哈希神经网络和图搜索算法的 DLG-x 加速器的可扩展性，我们将 DLG-x 的功能抽象为用户库任务平面中的三个 API：DLG_hashing、DLG_index 和 DLG_analysis 。这些 API 分别使用 NVMe I/O 协议的 C0h、C1h 和 C2h 命令建立。它们都拥有 NVMe 协议双字携带的两个基本参数：表示数据在认知 SSD 中位置的数据地址，以及以字节为单位的数据大小。

首先，DLG_hashing API 旨在提取输入数据的压缩特征并将其映射到哈希或语义空间，这是数据检索系统的基础，并且对于图像分类或分类等其他分析功能很有用。该命令包含一个扩展参数：hashcode length，它决定了携带信息的容量。例如，与具有 500 个对象类型的数据库相比，具有 1000 个对象的数据库需要更长的哈希码以避免信息丢失。其次，DLG_index API 是从 DLG-x 的图搜索功能中抽象出来的。它还包括一个扩展参数：T，表示用户根据应用场景配置的搜索结果的数量。最后，DLG_analysis API 允许用户利用深度神经网络的数据分析和处理能力来分析输入数据，并且它还拥有用于用户定义函数的保留字段。这些任务 API 是 Cognitive SSD 提供的关键近数据处理内核的抽象，可以单独或组合调用它们来开发不同的 SSD 内数据处理功能。例如，用户可以结合 DLG_hashing 和 DLG_index API 来完成大规模数据库上的数据检索，其中 DLG_hashing 将查询数据的特征映射为哈希码，DLG_index 使用它来搜索 top-T 相似实例。

3.4 认知 SSD 的 Runtime

部署在认知 SSD 内部嵌入式处理器上的认知 SSD 运行时负责管理通过 PCIe 接口传入的扩展 I/O 命令。它还将 API 相关命令转换为 DLG-x 加速器的机器指令，并处理 NAND 闪存的基本操作。它包括请求调度程序和基本固件。请求调度器包含三个模块：DLG 任务调度器、I/O 调度器和 DLG 配置器。DLG 配置器从主机接收 DLG_config 命令，并更新编译器生成的指令和认知 SSD 指定深度学习模型的参数。DLG 任务调度程序响应任务平面中支持的用户请求，并在认知 SSD 中启动相应的任务会话。I/O 调度程序将 I/O 请求分派给基本固件或 DLG-x。基本固件包括闪存转换层，用于逻辑块映射、垃圾收集、坏块管理功能，并与 NAND 闪存控制器通信以获取一般 I/O 请求。

值得注意的是，DLG-x 加速器一旦激活就会占用闪存带宽的显着部分，这可能会降低正常 I/O 请求的性能。为了缓解这个问题，DLG 任务调度器不是让任务或 I/O 调度器等待请求完成（表示为方法 A），而是接收带有门铃机制的主机发送的 NVMe 命令，并主动轮询任务的完成状态 DLG-x 定期（表示为方法 B）来决定是否可以调度下一个请求。我们在 DLG-x 加速器操作占用所有认知 SSD 通道的最坏情况影响下，使用灵活 IO 测试仪 (fio) 基准测试了第 5.1 节中描述的认知 SSD 原型的正常读/写带宽。实验表明，采用方法 B 仅导致正常 I/O 带宽下降 27%-44%，而采用方法 A 在 DLG-x 加速器繁忙时平均减少近 91% 的读/写带宽处理过度承担的检索任务。

3.5 硬件架构

图 3描绘了认知 SSD 的硬件架构。它由运行认知 SSD 运行时的嵌入式处理器、DLG-x 加速器和连接到闪存芯片的 NAND 闪存控制器组成。每个 NAND 闪存控制器连接一个通道的 NAND 闪存模块，并使用 ECC 引擎进行纠错。当每个通道中的设备锁步运行并并行访问时，内部带宽超过 I/O 接口。更重要的是，虽然 SSD 通常具有紧凑的 DRAM 来缓存数据或元数据，但内部 DRAM 容量很难满足因神经网络参数众多而臭名昭著的深度学习的需求。更糟糕的是，像 FTL 这样的基础固件和其他组件也占用了主要的内存资源。因此，NAND 闪存控制器暴露给 DLG-x 加速器，使 DLGx 能够绕过内部 DRAM，直接从 NAND 闪存读写相关工作数据。

3.6 可认知 SSD 中数据检索的流程

图 3还描绘了用户执行非结构化数据检索任务时认知 SSD 的整体流程。首先，假设 Hash-AlexNet 模型的硬件指令和参数已经通过 DLG-x 编译器和 DLG_config 命令生成并写入相应的区域，如图 3所示。Hash-AlexNet 是开发者指定的神经网络用于输入数据的特征提取。然后，当主机 DLG 库捕获检索请求时，它通过 SSD_write API 将用户输入数据从指定主机内存空间打包并写入到认知 SSD。同时，携带输入数据地址的 DLG_hashing 命令被发送到 Cognitive SSD 以生成哈希码。接收到该命令后，认知运行时的请求调度程序对其进行解析，并通知 DLG-x 加速器启动哈希特征提取会话。然后，DLG-x 自动从命令指定的数据地址获取输入查询数据，然后从 NAND 闪存加载深度学习参数。同时，另一个命令 DLG_index 由任务调度程序发送并排队。生成哈希码后，调度 DLG_index 来调用 DLG-x 中的图搜索功能，并使用哈希结果在数据图中搜索相关数据条目。在这种情况下，DLG-x 不断从 NAND 闪存中获取图形数据，并在任务完成后将最终检索结果发送到主机内存。

4 复现细节

4.1 与已有开源代码对比

由于文章的工作是设计了一个 SSD（固态硬盘），并且在 SSD 在融入了一个深度学习加速器实现图片的检索功能，因此代码需要涉及到硬件设计部分与软件部分。硬件部分基于 FPGA 进行验证，其中包含了 PCIe 控制器、BRAM 控制器、Nand Flash 控制器、DMA 控制器、神经网络加速器、向量搜索加速器；而软件部分是运行在 arm 处理器上，其中包含了与 Flash 关联的 FTL（闪存转换层）、ECC 校验，软硬件协同构成了整个 SSD。文章的作者使用 Cosmos plus OpenSSD 平台完成了整个系统的设计，该平台是开源的 SSD 项目，同样包含了硬件与软件部分，作者在此平台的基础上对硬件和软件部分代码进行修改，最终实现了 Cognitive SSD。根据作者给的 Github 链接，作者目前只开源了硬件部分的 BRAM Controller，向量搜索加速器，而像神经网络加速器，Nand Flash Controller 均未开源；并且软件部分均未开源。作者的硬件实物图如图 4所示。

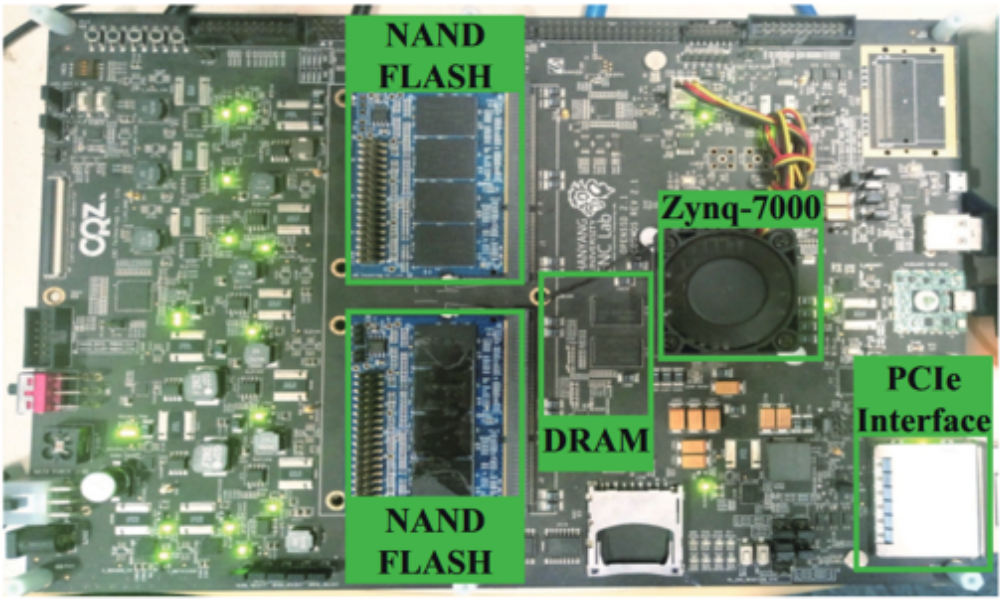

图 4. 原作者的硬件实物图

由于我们实验室并无 Cosmos plus OpenSSD 这个平台，所以我使用了另一款资源更多、性能更高的的芯片 Xilinx ZYNQ UltraScale+ ZU7EG 自己实现 SSD 的硬件部份和软件部份。并在此基础上，自己设计了一个高效的神经网络加速器，并把作者开源的向量搜索加速器融入硬件系统，由此复现了此篇论文。此外，我们还新提出了一个非侵入式的存储内计算框架，该框架包括一个建立在未修改的主机 NVMe 驱动程序上的易于使用的存储计算编程接口，以及在存储固件中的轻量级内核，用于加速存储中的 CNN 推理。具体而言，我们首先提出了一种基于超地址的新颖方法，供程序员选择不同的 CNN 加速器并执行存储中的 CNN 推理。其次，提出了一种面向 CSD 文件感知的页面数据布局，以弥合主机文件系统和设备固件之间的语义差距。

4.2 实验环境搭建

我使用的硬件实物图如图 5所示，与图 4原作者的硬件图有较大差异。其中红色框内是 PCIe 插槽，用于将此 SSD 插入主机主板上，最终能在 Ubuntu 中查看到并使用此块 SSD。绿色框内是 Xilinx ZYNQ UltraScale+ ZU7EG 芯片，它是一款 SOC，其中包含了 FPGA 和 ARM（Cortex-A53）两部分，硬件代码是用 FPGA 实现的，而软件代码是运行在 ARM 处理器上的，我使用此芯片实现了整个 SSD 系统。SSD 的硬件部份代码使用 vivado 2020.2 进行仿真、综合、实现。并最终导出 xsa 硬件信息文件给软件使用。软件部份使用 vitis 2020.2 设计和编译，并将最终的二进制文件烧录进 SOC 中。蓝色部份是 MLC Nand FLash 闪存芯片，用于存储 SSD 接收到来着主机用户的数据，具有掉电不丢失的功能。

跟原文作者一样，我们也使用了 CIFAR-10 以及 ImageNet-1000（224x224x3）这两个数据集去评估 SSD 内进行深度学习以及数据检索的能力。

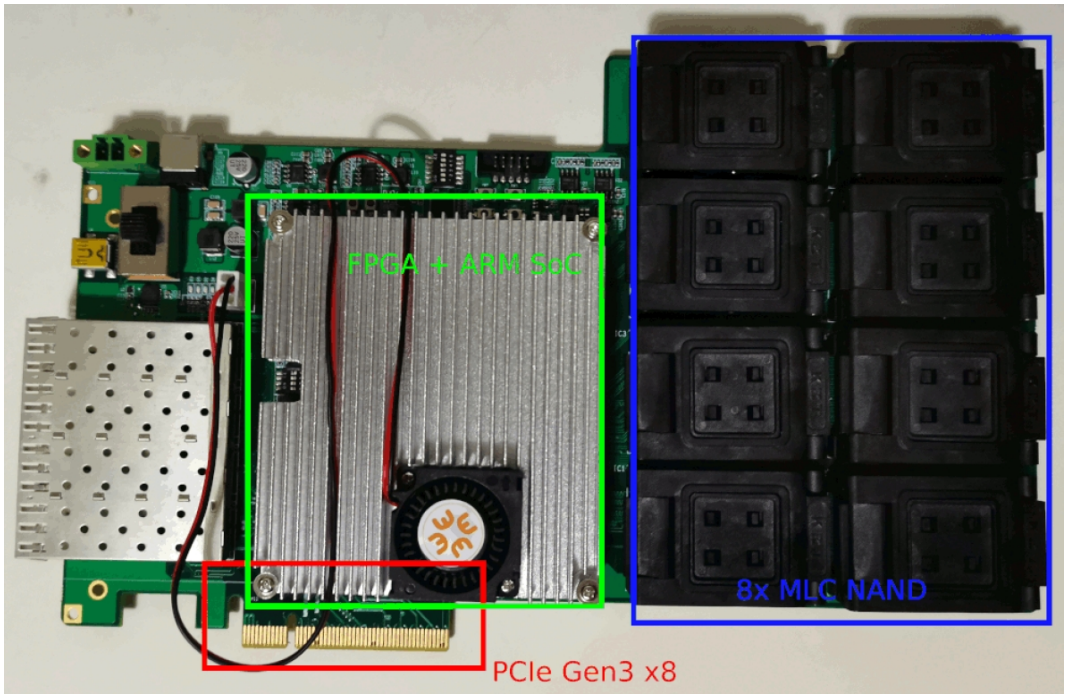

图 5. 我的硬件实物图

4.3 界面分析与使用说明

为了方便用户使用 SSD 的存储内计算功能，我基于 Qt 在 Ubuntu 上开发了一款上位机软件，其界面如图 6所示。用户想使用此 SSD 时，需将 SSD 的 PCIe 接口插入到主机主板的 PCIe 插槽，然后启动主机，进入 Ubuntu 系统运行此款上位机软件，即可进行 SSD 的普通读写命令以及存储内计算命令的发送。

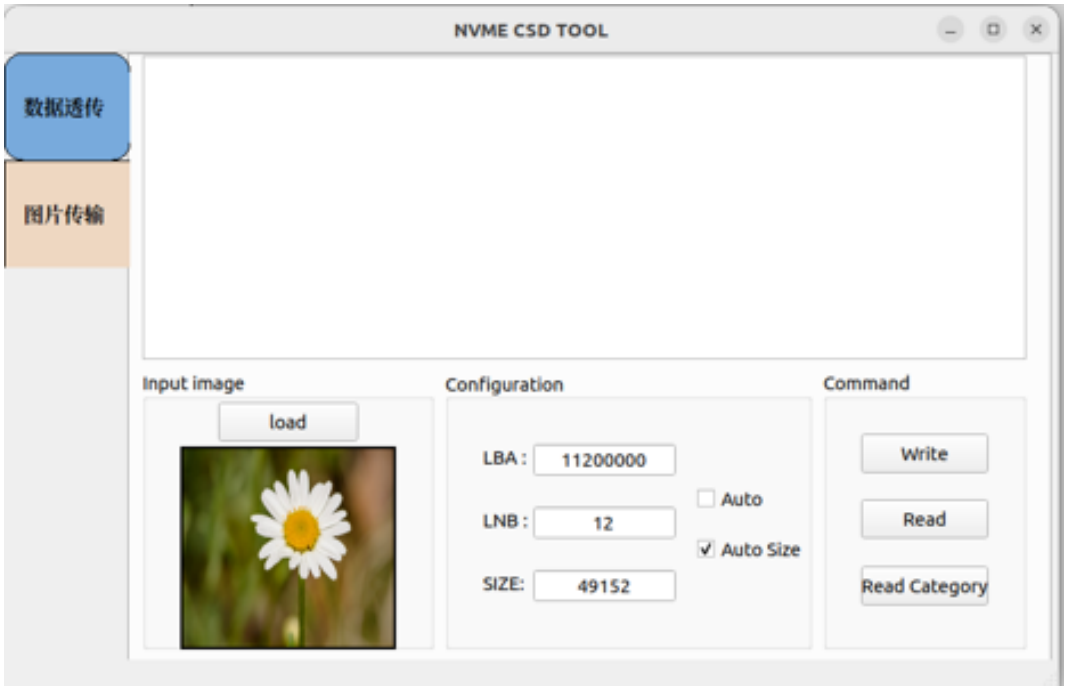

图 6. 上位机软件界面

此图形界面软件一共有两个页面，图中展现的是“数据透传”页面，用户可以直接在“Input

data” 输入框中输入要发送给 CSD 的字符串数据。中间 “Configuration” 框内的 LBA 是逻辑地址，即需要设置要将此字符串数据写入的位置；LNB 是 Logic Number Block，表示要写入字符串所占的 block 个数，CSD 中一个 block 为 4096Bytes；SIZE 则表示要写入数据的长度，如果勾选了 Auto Size 按钮，则会软件自动根据 “Input data” 处输入的字符串来自动计算处 LNB 和 SIZE。右侧的 “Command” 表示要下发给 CSD 的 NVMe 命令，点击 “Write” 按钮则会根据 “Configuration” 的配置将 “Input data” 发送给 CSD 中；同理，点击 “Read” 按钮则会读取 CSD 中逻辑地址为 LBA 的数据到窗口里。

第二个页面是 “图片传输” 页面，在这个页面中用户可以自行选择要写入的图片，写入完毕后可将图片读取到程序所在目录下，同时也可以利用 CSD 存储中计算的能力，读取该图片的分类结果，以及读取与该张图片海明距离最近的那些图片。其中 “Input Image” 框内用户可点击 load 按钮选择本地的任意一张图片，之后配置 “Configuration” 内的信息，点击 “Write” 按钮即可将图片写入。右图展示了选择一张花的图片，点击 “Auto Size” 按钮，自动计算出此图片的 LNB 和 SIZE。之后点击 “Write” 或 “Read” 进行写入读取，点击 “Read Category” 读取图片类别或在 SSD 内部直接检索与该张图片类型最相近的那些图片，并只返回那些匹配的图片。

4.4 创新点

4.4.1 神经网络加速器设计与优化

原论文中神经网络网络使用 2012 年的 Alexnet 模型，其 top-1 准确率为 57.1%, top-5 识别率达到 80.2%。并且其加速器基于 DeepBurning [55] 实现。对此，我进行改进，使用了 MobileNet V2（ =1.0， =1.0）网络模型，top-1 准确率为 72%, top-5 识别率达到 90% 以上，以此来实现更高精度的识别率以及更低的推理延迟和更低的功耗。网络加速器基于自主实现，有更低的推理延迟以及更小的硬件资源开销。如图所示，我使用 Pipeline、Loop Unroll、Array Partition 等技术对神经网络加速器进行设计和优化，因此系统性能的提升基本上都来源于神经网络加速器性能的提升。

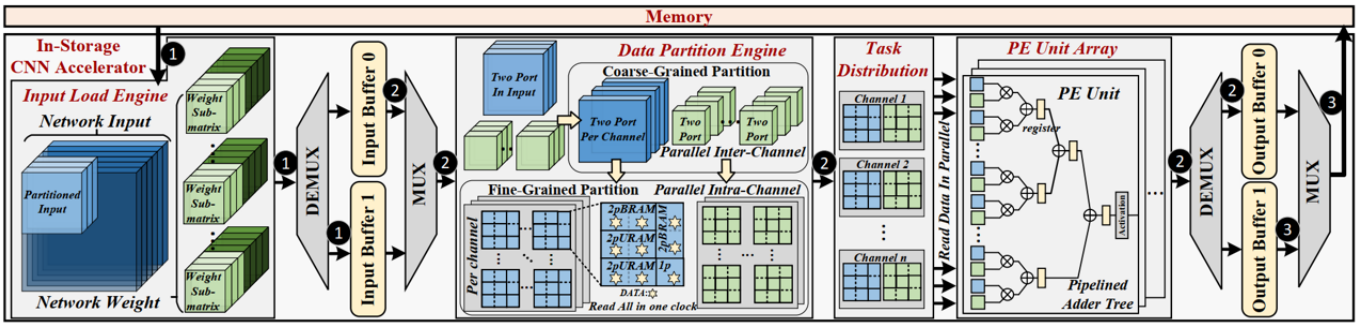

图 7. 神经网络加速器框架

我充分挖掘了 FPGA 在计算级别和任务级别上的并行性。首先是计算级别的并行性：由于 FPGA 中的资源有限，将所有输入数据和权重预加载到加速器中是不可能的。因此，需要输入加载引擎来选择并加载网络输入和权重的一部分。读取的数据暂时存储在 FPGA 的 RAM 中（例如，BRAM、URAM、LUTRAM），可以由程序员配置。然而，为了在 PE 阵列中实现最大计算级别的并行性，需要更高的 RAM 带宽，这受到每个 RAM 块的有限读/写端口的限

制。因此，我们需要一个数据分区引擎将输入数据分成不同的 RAM 块。我们首先对特征图和权重矩阵进行粗粒度的划分，将每个图像通道存储在不同的 RAM 块中，使数据传输受益于更多独立的读/写端口。通过这种方式，多个通道的数据可以同时被提取到 PE 阵列中。为了实现更高的吞吐量，我们进一步引入了细粒度的划分，其中每个通道的数据基于 CNN 卷积核的大小进行划分，并存储在不同的 RAM 块中。这允许一个图像通道中具有多个读/写端口，使得 PE 阵列可以在一个周期内获取一个图像通道中的所有数据。然后，由于数据已经被划分并存储在不同的 RAM 块中，任务分配引擎将划分的数据调度到空闲的 PE 单元，使它们能够同时运行，以实现最大计算级别的并行性。用于结果累积的流水线加法树，最终将结果存储在输出缓冲区并传输回来。

任务级别的并行性：为了重叠数据传输延迟和计算，我们通过在输入/输出路径中利用两个乒乓缓冲区进一步探索任务级别的并行性。这允许同时执行三个阶段：(1) 从内存加载数据的部分，(2) 对数据进行分区并使用 PE 阵列执行并行计算，以及 (3) 将计算结果写回内存。通过对这三个阶段的延迟进行慎重考虑，开发人员可以权衡第 2 阶段的延迟，以匹配第 1 和第 3 阶段的延迟，从而改善整体推断性能，而且可以选择使用更多或更少的硬件资源。

4.4.2 非侵入式的存内计算框架

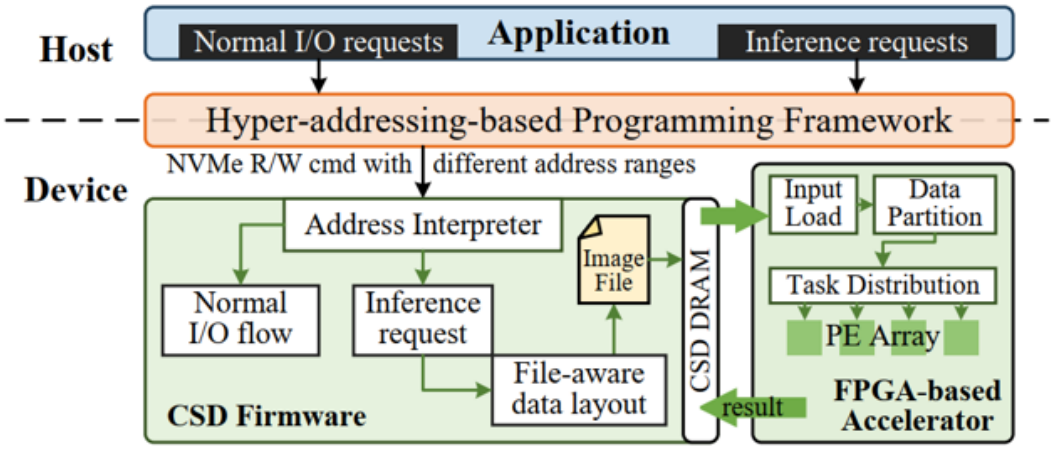

图 8. 非侵入式存内计算框架

图 8展示了非侵入式框架设计的整体架构。应用程序员可以通过我们易于使用的库发出存储推断请求，在其中我们采用超寻址来区分普通的 NVMe I/O 请求和存储中的 CNN 推断请求。在 CSD 固件内部，我们引入了一个地址解释器来识别具有不同 LPA 范围的 NVMe 请求。然后，我们根据 NVMe 请求中的 LPA 范围决定是处理为普通 I/O 请求还是存储中的推断请求。对于推断请求，我们进一步需要通过我们的面向文件的页面数据布局检索所有相关的文件页面，然后将它们整合为存储中 CNN 加速器的输入。接下来，基于 FPGA 的加速器将加载图像，将其划分为多个子块，并将它们分派到不同的处理引擎（PE）以实现最大并行性。最后，CSD 仅将图像识别的结果返回给主机，同时通过减少原始图像数据的移动开销来实现更高的性能和能效。

5 实验结果分析

首先我评估了自主设计的神经网络加速器所带来的性能提升，跟原文的神经网络延迟、功耗对比表格如表 ??所示。可以看到，新改进的神经网络加速器以及选用的新模型与原论文中的两个网络相比，能够带来更低的推理延迟，以及更低的功耗。此外，整个 SSD 系统所用的硬件资源如表 2所示。

表 1. 神经网络优化的性能和功耗对比表

Model	-	Latency(ms)	Power(Watt)
Hash-AlexNet	DLG-x	38	9.1
Hash-ResNet-18	DLG-x	94	9.4
MobileNet V2	DLG-x	30.466	2.898

表 2. FPGA 硬件资源的使用情况

Module	**LUT**	**FF**	**BRAM**	**DSP**
NVMe Controller	11034	31287	46.5	0
DDR Controller	15415	17919	25.5	3
Flash Controller	61042	33770	28	0
CNN Accelerator	71862	68193	165	351
Interconnect and others	55565	140445	15	0
In Total	**214918**	**223586**	**280**	**354**
Percent(%)	**93.28**	**48.52**	**89.74**	**20.49**

此外，我分析了神经网络加速器跟 Jetson NX GPU 相比的性能关系，分割 (breakdown) 图如图 9所示。可以看到，由于我加入了计算级别并行性、任务级别的并行性以及两个 IP 核协同工作，最终使得加速器最终在某些 batchsize 上的性能比 GPU 还要好，而功耗在所有的 batchsize 上都远远比 GPU 的功耗低。

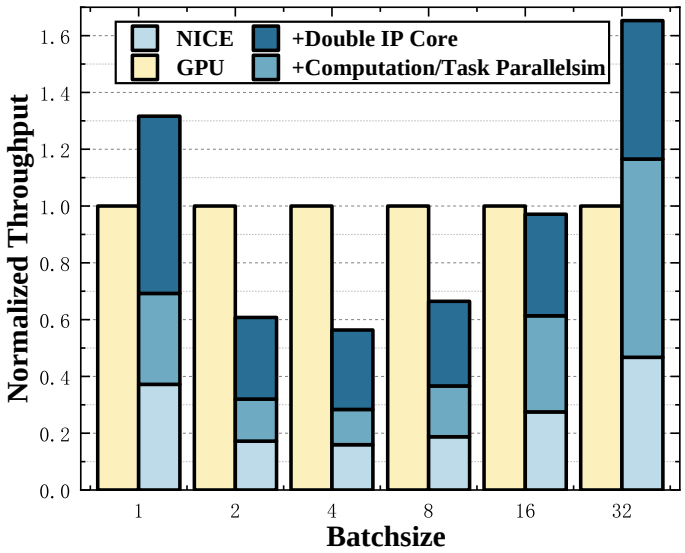

图 9. 性能 Breakdown

最后，评估了系统在检索图片上的性能，如图 10所示。图中的左侧是原文的实验结果，右侧是使用我设计的神经网络加速器跑出来的实验结果，可以看到，自己设计的加速器实现的性能比原文中的性能更好，并且功耗更低。

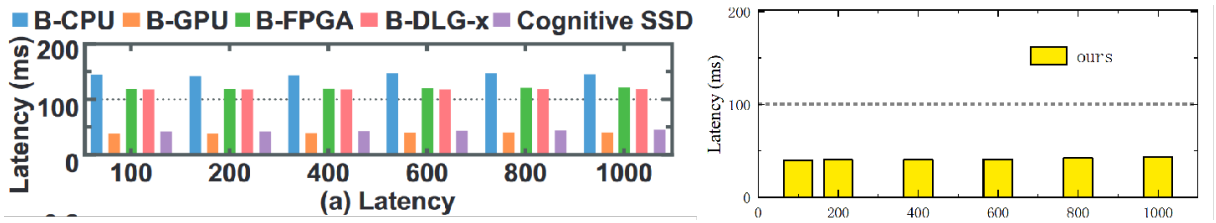

图 10. 性能 Breakdown

6 总结与展望

本篇文章首先介绍了传统了数据检索架构，以及使用存储内计算的新型数据检索架构，并且分析了两者的利弊。原文章实现了可认知的 SSD，用于做存储内的数据检索系统，因此本文详细阐述了与其相关的背景，例如非结构化数据、近数据处理、深度学习加速器。接着，详细分析了原文作者设计的可认知 SSD 的系统架构。最后本文复现了原作者的工作，原作者在 CosMos+ Openssd 平台上实现了它的 SSD，而本文用更高性能和更多硬件资源 Zynq UltraScale+ ZU7EG 芯片同样实现了可用于存储内计算的 SSD。由于神经网络的加速器未开源，因此本文自主设计了一个更高效的神经网络加速器，并且选用准确度更高的神经网络，最终实现了比原文章更低的推理延迟以及更低的系统功耗。此外，本文还提出了一种非侵入式的存储内计算框架，可在不修改主机 NVMe 驱动的情况下，使用 hyper-address 空间来发起存储内计算的命令。而对于进一步的研究方向，我们会将更多的 AI 应用用存储内计算这种框架来加速，减少大部分无效的数据移动来实现更高的性能以及更低功耗。

参考文献

[1] Joel Coburn Todor I. Mollow Rajesh K. Gupta Adrian M. Caulfield, Arup De and Steven Swanson. Moneta: A high-performance storage array architecture for nextgeneration, non-volatile memories. In *In Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO ' 43*, pages 385–395, 2010.

[2] Ilya Sutskever Alex Krizhevsky and Geoffrey E. Hinton. imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, 2017.

[3] Rajesh Gupta Arup De, Maya Gokhale and Steven Swanson. Minerva: Accelerating data analysis in next-generation ssds. In *In Proceedings of the 2013 IEEE 21st Annual International Symposium on FieldProgrammable Custom Computing Machines, FCCM ' 13*, pages 9–16, 2013.

[4] Deng Cai. A revisit of hashing algorithms for approximate nearest neighbor search. *CoRR*, abs/1612.07545, 2016.

[5] Near data processing: Insights from a micro 46 workshop. R. balasubramonian, j. chang, t. manning, j. h. moreno, r. murphy, r. nair, and s. swanson. *IEEE Micro*, 34(4):36–42, 2014.

[6] Michael Kaminsky-Amar Phanishayee Lawrence Tan David G. Andersen, Jason Franklin and Vijay Vasudevan. Fawn: A fast array of wimpy nodes. In *In Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles, SOSP ' 09*, pages 1–14, 2009.

[7] Sudharshan S. Vazhkudai-Youngjae Kim Xiaosong Ma Peter J. Desnoyers Devesh Tiwari, Simona Boboila and Yan Solihin. Active flash: Towards energy-efficient, in-situ data analytics on extreme-scale machines. In *In Proceedings of the 11th USENIX Conference on File and Storage Technologies, FAST' 13*, pages 119–132, 2013.

[8] Youngjae Kim-Xiaosong Ma Simona Boboila Devesh Tiwari, Sudharshan S. Vazhkudai and Peter J. Desnoyers. Reducing data movement costs using energy efficient, active computation on ssd. In *In Proceedings of the 2012 USENIX Conference on Power-Aware Computing and Systems, HotPower' 12*, pages 4–4, 2012.

[9] Cong Fu and Deng Cai. Efanna : An extremely fast approximate nearest neighbor search algorithm based on knn graph. *CoRR*, abs/1609.07228, 2016.

[10] Cong Fu, Changxu Wang, and Deng Cai. Fast approximate nearest neighbor search with navigating spreading out graphs. *CoRR*, abs/1707.00143, 2017.

[11] S. Shan H. Liu, R. Wang and X. Chen. Deep supervised hashing for fast image retrieval. In *In 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2064–2072, 2016.

[12] Yuxiao Zhou-Mark Gahagan Hung-Wei Tseng, Qianchen Zhao and Steven Swanson. Morpheus: Creating application objects efficiently for heterogeneous computing. *SIGARCH Comput. Archit. News*, 44(3):53–65, 2016.

[13] Seongsik Park Sei Joon Kim Eui-Young Chung Hyeokjun Choe, Seil Lee and Sungroh Yoon. Near-data processing for machine learning. *CoRR*, abs/1610.02273, 2016.

[14] Jignesh M. Patel Chanik Park-Kwanghyun Park Jaeyoung Do, Yang-Suk Kee and David J. DeWitt. Query processing on smart ssds: Opportunities and challenges. In *In Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, SIGMOD ' 13*, pages 1221–1230, 2013.

[15] Jiwu Shu Jiacheng Zhang and Youyou Lu. Parafs: A log-structured file system to exploit the internal parallelism of flash devices. In *In 2016 USENIX Annual Technical Conference (USENIX ATC' 16)*, pages 87–100, 2016.

[16] Song Jiang Zhenyu Hou Yong Wang Jian Ouyang, Shiding Lin and Yuanzheng Wang. Sdf: Softwaredefined flash for web-scale internet storage systems. In *In Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS ' 14*, pages 471–484, 2014.

[17] Zhenyu Hou Peng Wang Yong Wang Jian Ouyang, Shiding Lin and Guangyu Sun. Active ssd design for energy-efficiency improvement of web-scale data analysis. In *In Proceedings of the 2013 International Symposium on Low Power Electronics and Design, ISLPED ' 13*, pages 286–291, 2013.

[18] Yang-Suk Kee Yannis Papakonstantinou Jianguo Wang, Dongchul Park and Steven Swanson. Ssd in-storage computing for list intersection. In *In Proceedings of the 12th International Workshop on Data Management on New Hardware, DaMoN ' 16*, pages 4:1–4:7, 2016.

[19] J. Hsiao K. Lin, H. Yang and C. Chen. Deep learning of binary hash codes for fast image retrieval. In *In 2015 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 27–35, 2015.

[20] Yinhe Han Xin Zhao Bosheng Liu Lili Song, Ying Wang and Xiaowei Li. C-brain: A deep learning accelerator that tames the diversity of cnns through adaptive data-level parallelization. In *In Proceedings of the 53rd Annual Design Automation Conference, DAC ' 16*, pages 123:1–123:6, 2016.

[21] Zsolt István Louis Woods and Gustavo Alonso. Ibex: An intelligent storage engine with support for advanced sql offloading. *Proc. VLDB Endow.*, 7(11):963–974, 2014.

[22] Javier González Matias Bjørling and Philippe Bonnet. Lightnvm: The linux open-channel ssd subsystem. In *In 15th USENIX Conference on File and Storage Technologies (FAST' 17)*, pages 359–374, 2017.

[23] S. S. Vazhkudai P. Desnoyers S. Boboila, Y. Kim and G. M. Shipman. Active flash: Out-of-core data analytics on flash storage. In *In 012 IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–12, 2012.

[24] Sizhuo Zhang Shuotao Xu Sang-Woo Jun, Andy Wright and Arvind. Grafboost: Using accelerated flash storage for external graph analytics. In *In Proceedings of the 45th Annual International Symposium on Computer Architecture, ISCA ' 18*, pages 411–424, 2018.

[25] Sungjin Lee Jamey Hicks John Ankcorn Myron King Shuotao Xu Sang-Woo Jun, Ming Liu and Arvind. Bluedbm: An appliance for big data analytics. In *In Proceedings of the 42Nd Annual International Symposium on Computer Architecture, ISCA ' 15*, pages 1–13, 2015.

[26] Hyunok Oh Sungchan Kim Youngmin Yi Sangyeun Cho, Chanik Park and Gregory R. Ganger. Active disk meets flash: A case for intelligent ssds. In *In Proceedings of the 27th International ACM Conference on International Conference on Supercomputing, ICS ' 13*, pages 91–102, 2013.

[27] Sundaram Bhaskaran Trevor Bunker Arup De Yanqin Jin Yang Liu Sudharsan Seshadri, Mark Gahagan and Steven Swanson. Willow: A userprogrammable ssd. In *In Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation, OSDI' 14*, pages 67–80, 2014.

[28] Ninghui Sun Jia Wang Chengyong Wu Yunji Chen Tianshi Chen, Zidong Du and Olivier Temam. Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. In *In Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS ' 14*, pages 269–284, 2014.

[29] Sheng Wang Wu-Jun Li and Wang-Cheng Kang. Feature learning based deep supervised hashing with pairwise labels. In *In Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI' 16*, pages 1711–1717, 2016.

[30] Ethan L. Miller Yangwook Kang, Yang-Suk Kee and Chanik Park. Enabling cost-effective data processing with smart ssd. In *2013 IEEE 29th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–12, 2013.

[31] Hyuck Han Hyeonsang Eom Yongseok Son, Nae Young Song and Heon Young Yeom. A user-level file system for fast storage devices. In *In Proceedings of the 2014 International Conference on Cloud and Autonomic Computing, ICCAC ' 14*, pages 258–264, 2014.

[32] Jiwu Shu Youyou Lu and Weimin Zheng. Extending the lifetime of flash-based storage through reducing write amplification from file systems. In *In Presented as part of the 11th USENIX Conference on File and Storage Technologies (FAST' 13)*, pages 257–270, 2013.

[33] Youmin Chen Youyou Lu, Jiwu Shu and Tao Li. Octopus: an rdma-enabled distributed persistent memory file system. In *n 2017 USENIX Annual Technical Conference (USENIX ATC' 17)*, pages 773–785, 2017.

[34] Joel Emer Yu-Hsin Chen and Vivienne Sze. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. In *In Proceedings of the 43rd International Symposium on Computer Architecture, ISCA ' 16*, pages 367–379, 2016.