国产 AI 芯片
软件生态白皮书

主编单位：湖南大学

联合编写单位：北京智谱华章科技有限公司、清华大学

北京邮电大学、新华三技术有限公司、南方科技大学

2025 年 11 月

编写人员

湖南大学：

陈果、曾一夫、陈禹澎、陈泽宇、徐方林、何冬

北京智谱华章科技股份有限公司：

鄢兴雨

清华大学：

翟季冬

北京邮电大学：

张宇超

新华三技术有限公司：

汪卫国

南方科技大学：

李卓钊

柏林云科技自媒体：

柏林

说明：中国计算机学会（CCF）青年计算机科技论坛（YOCSEF）25-26 年度“智算基础设施”系列技术论坛活动的所有嘉宾和参与者对本文亦有重要贡献，特此感谢。

目录

一、背景和意义概述 1
1.1 背景 1
1.2 目的和意义 1
二、AI 芯片软件生态核心组成与功能解析 3
2.1 基础支撑层：硬件算力的“翻译与调度中枢” 4
2.2 核心工具层：算力释放的“性能优化引擎” 5
2.3 框架适配层：开发者友好的“应用接口桥梁” 7
2.4 管理监控层：系统稳定的“运维保障屏障” 9
三、国产 AI 芯片软件生态资源现状 13
3.1 国产 AI 芯片分类及代表性厂商 13
3.2 国产 AI 芯片软件生态各环节资源汇总与完善度对比 14
3.3 国产 AI 芯片软件社区活跃度对比数据表 24
3.4 本章小结 28
四、结语 30
附录一 AI 芯片硬件基础：理解软件生态所“指挥”的对象 31
参考文献 34

一、背景和意义概述

1.1 背景

在科技竞争日益激烈的国际大背景下，以构建自主可控的 AI 芯片及其软件生态战略为指引，我国 AI 芯片近些年在技术创新与市场拓展方面均收获颇丰。以华为昇腾、寒武纪、地平线、沐曦、燧原科技、海光信息、壁仞科技、摩尔线程及天数智芯等为代表的一批本土企业，已成功推出一系列具有市场竞争力的 AI 芯片产品，在国内市场形成了多厂商、多技术路线并行的活跃竞争格局。

随着国产 AI 芯片在算力、能效比等硬件指标上的突破，用户关注点已从“有没有”转向“好不好”——即软件生态的成熟度、兼容性与易用性。这里的“好不好”，其核心指向的已不再仅仅是芯片的理论峰值性能，而是其背后支撑的软件生态是否成熟、完善与开放。

一个成熟的软件生态，是决定芯片价值能否充分释放的关键。它体现在很多方面，包括基础软件栈的完备性与稳定性、算子库的丰富度与高性能实现、编译工具链的智能化与高效性、以及对 PyTorch 等业界主流 AI 框架的无缝兼容与深度适配能力、开发社区的活跃度等。对于广大的 AI 开发者和企业用户而言，一个完善的软件生态意味着其现有的 AI 应用、算法模型与开发工作流，能够以极低的迁移成本、甚至实现“无感”地部署到新的国产硬件平台上，从而避免大规模的代码重构和漫长的适配调试周期。因此，软件生态的构建水平，不仅是衡量国产 AI 芯片核心竞争力的关键标尺，更直接决定了其商业化落地的广度、深度以及最终能否赢得用户信任与市场份额。

1.2 目的和意义

本白皮书的核心目的在于系统性地梳理和评估国产 AI 芯片软件生态的发展现状，为产业界、学术界及政府部门提供一份客观的技术参考与决策依据。AI 芯片软件生态主要由"四层架构"组成，包括基础支撑层、核心工具层、框架适配层与管理监控层，各模块通过"技术依赖-功能协同"形成闭环，共同作用于 AI 模型的训练与推理过程。然而，不同厂商在生态建设上呈现出显著差异：例如，华为昇腾通过

自研软件栈，构建出一套完整的自主软件生态体系；摩尔线程通过高度对标 NVIDIA CUDA 生态，实现了极高的兼容性。

本白皮书的意义体现在三个层面：（1）深度剖析 AI 芯片软件生态，形成系统性介绍。将 AI 芯片软件生态分为"四层架构"，包括基础支撑层、核心工具层、框架适配层与管理监控层，剖析其概念与作用，介绍具体案例。（2）汇总国产 AI 芯片软件生态资源，形成资源指南。详细调研多款代表性国产 AI 芯片，汇总介绍其软件生态并给出资源链接，帮助企业和开发者根据自身应用场景（如训练、推理、边缘计算等）和技术栈基础，选择最适合的解决方案，避免盲目追求"算力峰值"而忽视生态兼容性的误区。（3）为政策制定提供参考。通过客观评估国产软件生态，为相关产业政策的制定提供数据支撑，助力我国 AI 芯片产业实现从"基础可用"向"场景好用"的关键跨越。

湖南大学

二、AI 芯片软件生态核心组成与功能解析

AI 芯片软件生态是衔接硬件算力与上层应用的“技术枢纽”，其本质是通过分层设计实现“硬件能力抽象化、算力调用标准化、开发流程便捷化”。参考 CPU（如飞腾）、AMD、英伟达等成熟软件生态的“底层支撑 - 核心优化 - 上层适配 - 运维保障”逻辑，AI 芯片软件生态可划分为基础支撑层、核心工具层、框架适配层与管理监控层四大模块。各模块通过“技术依赖 - 功能协同”形成闭环，共同作用于 AI 模型的训练与推理过程。

为了方便有一定 GPU 编程经验的读者理解，以下使用 NVIDIA 生态为例进行类比讲解一个任务在 GPU 上的处理流程。当用户在 PyTorch 中指定 NVIDIA GPU 开始执行任务，流程从框架适配层开始：框架把高层算子映射到 cuDNN/cuBLAS 等实现，并做必要的数据格式转换。接着进入核心工具层，编译器将计算图编译成 PTX 或机器指令，并在需要时调用 NCCL 完成多卡通信。生成的指令再交由基础支撑层执行：CUDA Runtime 和 CUDA Driver 合作负责调度与显存管理，CUDA Driver 将上层指令翻译成可在 GPU 上运行的底层操作，并通过 GPU 的 ECC 硬件、Watchdog（超时检测）等机制保证稳定性。整个执行过程中，管理监控层通过 nvidia-smi/NVML 监控状态，Kubernetes 分配 GPU 资源，驱动在异常时进行隔离与恢复。四层协同完成了从模型代码到 GPU 指令的转换与可靠执行。而对于 GPU 编程经验较少、对 AI 芯片与 CPU 区别理解较少的读者，可以阅读“附录一 AI 芯片硬件基础：理解软件生态所‘指挥’的对象”进行了解。

湖南大学

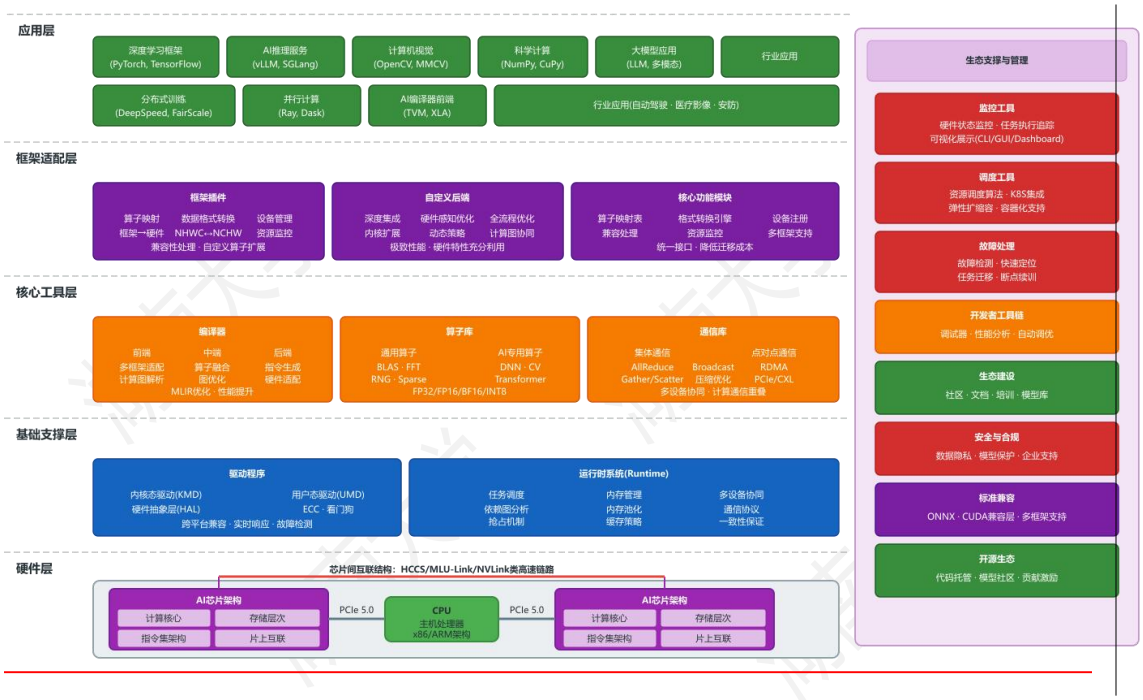

图 2.1 AI 芯片生态各个层级的结构

上图介绍了 AI 芯片生态各个层级的结构。接下来，结合国产 AI 芯片（如华为昇腾、摩尔线程、寒武纪）的技术实践，深入解析各模块的核心组成、支撑技术及逻辑关系。

2.1 基础支撑层：硬件算力的“翻译与调度中枢”

基础支撑层是 AI 芯片软件生态的“地基”，负责把底层硬件算力翻译为上层可用的形式，并对资源进行底层调度。它主要包括芯片驱动、底层库和系统运行时等组件，相当于 AI 芯片的操作系统。

在这一层，软件通过抽象硬件复杂性，让上层开发者无需直接处理寄存器、DMA 等细节。例如，摩尔线程的 MUSA SDK 提供了底层编译器和运行时库，屏蔽了 GPU 硬件细节，开发者可以像使用 CUDA 那样调用 GPU 加速计算。又如，华为昇腾提供的 CANN (Compute Architecture for Neural Networks) 就包含基础支撑层部分，这一部分封装了昇腾 AI 处理器的指令集和算子，实现对硬件的抽象和使能，并已全面开源以方便开发者直接调度底层资源。

基础支撑层还承担着任务调度和资源管理职责：在 AI 芯片上运行的模型计算任务，会通过底层运行时被合理地分配到各计算核心上执行，并管理显存分配、数

据传输等。这类似于 CUDA Driver 在 NVIDIA GPU 生态中的作用——确保每个 GPU 核心高效执行分配给它的计算。在国内 AI 芯片生态中，各大厂商均构建了自己的基础软件平台：寒武纪的 NeuWare 基础系统就同时支持云端、边缘和终端各类芯片，提供统一的底层接口，方便智能应用在不同算力设备间迁移和调优。总体来说，基础支撑层作为“翻译与调度中枢”，将硬件能力抽象化并提供标准接口，使上层工具和应用能够“不感知”硬件复杂性地调用算力。这是国产 AI 芯片从功能可用走向生态好用的第一步基础。

除了指令翻译和资源调度外，基础支撑层还需提供健康检测、故障隔离与版本兼容的能力。类似于 NVIDIA NVML，华为昇腾提供了 npu-smi 作为 GPU 信息查询接口；而通过 K8S 设备插件等工具，上层调度器也可感知 AI 芯片资源状态。另外，运行时需与驱动/固件保持接口契约与版本协商，确保编译期与运行期一致性，降低算子库、通信库升级带来的回归风险。这些“可运维”能力与“可用性”直接正相关，是国产生态由“能用”迈向“好用”的关键支点。

2.2 核心工具层：算力释放的“性能优化引擎”

核心工具层是 AI 芯片软件生态的“性能核心”，汇集各种让算力真正高效发挥的优化工具链。主要涵盖模型编译器、算子库、性能分析和调优工具等，它们相当于为芯片配备的“引擎和涡轮增压器”。这一层的首要组成是 AI 编译器/执行引擎：它负责将上层训练好的模型转换为适配芯片的高效执行方案，包括计算图优化、算子融合和指令调度[1]。

典型流程是：模型通过解析器导入（支持 ONNX、TensorFlow、PyTorch 等格式），转换为内部中间表示 IR，然后进行算子融合、常量折叠、子图划分等图优化，再映射到具体硬件资源（如 PE 阵列、缓存），最后生成对应芯片 ISA 的指令流。为了进一步提升开发效率与跨平台兼容性，业界已出现一批“编译兼容”工具，它们以统一的高层语义描述算子，再向下翻译成不同芯片的后端代码。例如:Triton 采用类 Python 的 Kernel 语言，开发者只需写一次矩阵乘、卷积等核心逻辑，其编译器即可自动生成面向 NVIDIA GPU、AMD GPU 乃至多家国产加速器的高效机器码，显著降低手写 CUDA/ROCm 内核的门槛[18]。TileLang 则把“张量分块”作为一等

公民，通过 tile-level 的声明式语法描述数据搬运与计算，编译期自动完成流水线调度、双缓冲和局部性优化，同一套 TileLang 源码可无缝编译到寒武纪 MLU、华为昇腾、 Apple Metal 等多种架构，实现“一次编写、处处加速”。

国产厂商方面，以寒武纪 Cambricon 为例，其 MagicMind 推理加速引擎可将用户在 PyTorch、TensorFlow 等框架训练好的模型一键编译为思元芯片可执行的代码，内部集成了基于 MLIR 的图编译技术，可自动完成功能算子的解析、后端代码生成和优化。借助 MagicMind，用户只需极少的工作即可将模型部署到寒武纪全系列芯片上，并获得接近 CUDA 生态的性能。同样地，华为昇腾提供的全流程编译工具链（如昇腾编译器、算子开发工具等）能够保证模型在昇腾 NPU 上高效运行；壁仞、燧原等公司也在开发自有的编译器来优化其芯片上的深度学习计算。

核心工具层的另一关键要素是高度优化的算子库和性能库。这些库提供基础数学运算和深度学习常用算子的实现，并针对特定芯片架构做了极致优化。例如，NVIDIA CUDA 有 cuBLAS、cuDNN 等库，摩尔线程 MUSA SDK 也包含 muBLAS（基础线性代数）、muDNN（深度神经网络）等加速库，每一种都在摩尔线程 GPU 上做了深度优化。寒武纪的 NeuWare 同样提供 CNNL（寒武纪神经网络库）以加速卷积、矩阵乘等算子运算[2]。这些优化算子库相当于为 AI 芯片配备“标准部件”，让常用操作可以直接调用并运行在接近手工优化的最快速度上。此外，核心工具层通常还包括调试与 Profiling 工具，帮助开发者分析模型在芯片上的运行效率瓶颈。比如寒武纪的 MagicMind 提供了丰富的调试和性能调优工具，方便用户检查计算图、跟踪指令执行、分析硬件利用率。

同样在核心工具层中的通信库是实现多芯片协同计算的关键工具，被誉为 AI 计算集群中的“数据传输桥梁”。在单卡无法容纳超大模型或需要加速训练时，往往需要多块加速卡甚至跨服务器协同，这就要求高效的通信机制在各设备间传递梯度和参数。成熟的通信库可以极大降低多设备同步的开销，NVIDIA 的 NCCL（集群通信库）便是业界广泛使用的例子，它针对 GPU 的拓扑和 NVLink 架构做了优化，可实现 GPU 间广播、归约等操作的高吞吐通信。国产 AI 芯片也在积极构建自己的通信加速能力。例如，寒武纪 NeuWare 软件栈内置了 CNCL 通信库和 Horovod 分布式框架适配，支持数据并行、模型并行以及混合并行等多种训练模式，实现从单

卡到集群的顺畅扩展[2]。摩尔线程 MUSA SDK 包含自研的 MCCL（Moore Collective Communication Library），可用于其多 GPU 系统的数据传输与同步；沐曦也联合开源社区提供拓扑感知的通信方案，在 HAMi 资源调度中充分利用 MetaXLink 高速互联来优化多卡通信延迟[4]。需要指出的是，通信库不仅服务于训练过程，在大规模推理场景（如多节点在线推理）中也扮演重要角色，保障模型副本间或模型与数据节点间的高效数据交换。

通信库作为“数据传输桥梁”填平了多芯片算力之间的沟壑。对于国产 AI 芯片生态，拥有自主可控且高效的通信库意味着大规模 AI 任务可以运行在本土算力集群上而不减速。这也是近年来国内厂商积极投入这一领域的重要原因：从自行开发通信栈，到兼容开源方案（如高性能 MPI、Gloo），再到针对国产互连技术（如 PCIe Gen 5、国产高速互联总线）优化，都体现出对“多设备一体”计算体验的追求[2]。随着通信库的性能提升和标准化，多芯片协同的复杂度将进一步降低，AI 算法工程师能更专注于模型本身而非数据搬运，从而在软硬件协同下实现实现高效的多卡并行。

2.3 框架适配层：开发者友好的“应用接口桥梁”

框架适配层是 AI 芯片软件生态与开发者之间的“最后一公里”，核心任务是让开发者在 不改变主流开发习惯 的前提下，把模型顺畅迁移到国产算力之上。其基本思路是：一方面，通过对 PyTorch、TensorFlow 等国际主流框架进行插件式适配或后端扩展，让这些框架可以直接在昇腾、寒武纪、摩尔线程等国产芯片上运行；另一方面，通过发展 PaddlePaddle、计图（Jittor）等国产深度学习框架和推理软件栈，从软件侧原生支持多家国产加速硬件，形成“国产软硬协同”的一体化方案。在这一层，生态建设的评价标准不仅要关注“能不能跑”，还要关注“迁移成本多高、性能损耗多大、开发体验是否一致”。

对于已经大量使用 PyTorch 的开发者，框架插件 提供了一条最平滑的迁移路径：通过扩展现有框架的设备类型和算子实现，让原有模型代码以极少改动跑在国产芯片上。典型做法是为框架增加新的设备标识（如 npu、mlu、musa），并在内

部完成算子映射和数据格式转换——对用户而言往往只需把 cuda 换成 npu / mlu / musa 即可。

华为昇腾社区维护的 torch_npu 仓库，就是一个面向 Ascend NPU 的 PyTorch 扩展；它将 PyTorch 的张量和算子执行通过 CANN 软件栈映射到昇腾 AI 处理器，使 PyTorch 用户可以直接复用 Ascend 的算力。寒武纪则提供了 CATCH / Torch-MLU 等扩展，使 PyTorch 支持其 MLU 加速卡后端，在训练和推理场景下都能获得高性能加速。摩尔线程开源的 torch_musa 项目，则在 PyTorch 中新增了 MUSA 设备类型，并提供与 CUDAExtension 类似的 MUSAExtension 接口，方便第三方算子与 CUDA 版本一键移植到 MUSA 平台。配合其 Musify 工具链，已有项目（如 llama.cpp）可以较为顺畅地从 CUDA 生态迁移到摩尔线程生态。

在推理与部署层面，国产通用软件栈也通过插件方式支持多家国产芯片。以飞桨生态为例，PaddlePaddle 及其部署组件 FastDeploy、PaddleX 已支持在 NVIDIA GPU、百度昆仑 XPU、华为 Ascend NPU、寒武纪 MLU、海光 DCU 等多种硬件上无缝切换。对应用开发者而言，只需配置不同的后端即可在不同国产算力平台上部署同一模型，大幅降低了“换芯片就重写代码”的门槛。

与“轻量封装”的插件模式相比，自定义后端与国产自研框架 更强调深度耦合与性能极致。其路径大致有两种：一是像 MindSpore + Ascend、MagicMind + MLU 这样，将框架执行后端与国产芯片紧密绑定，在计算图优化、算子调度、混合精度策略等方面做全栈协同；二是像 PaddlePaddle、计图这类国产通用框架，从设计之初就面向多家国产加速硬件，形成“一个框架、多种国产芯片”的统一开发体验。

华为在 MindSpore 中提供了面向 Ascend 的专用后端，将计算图直接交由 CANN 编译和调度，使模型在昇腾 NPU 上可获得显著优于通用后端的训练效率，特别是在千亿参数级大模型训练中验证了良好的扩展性。寒武纪的 MagicMind 则是一个跨框架的推理引擎：它从 TensorFlow / PyTorch 等框架导入模型后，利用 MLIR 等技术在图级别做算子融合和调度，并针对思元 MLU 生成高度优化的代码，在实际业务场景中较直接调用通用算子库有明显性能优势。

从“软件先行”的角度，国产通用框架也在主动反向带动硬件适配。飞桨生态通过 PaddleX / FastDeploy 支持 Kunlun、Ascend、MLU、DCU 等多家国产芯片，成为上层应用对接多源国产算力的统一入口。清华牵头研发的 计图（Jittor） 是国内首个由高校主导并开源的深度学习框架，采用统一计算图与元算子融合、即时编译等技术，在性能上可与国际主流平台竞争。其核心创新在于元算子机制与动态编译技术的深度结合：将神经网络基础算子抽象为三类共 18 个元算子，通过元算子融合构建复杂计算模块，仅需优化少量元算子即可完成新硬件适配；内置编译器可将 Python 代码动态转化为寒武纪 BANG 语言等硬件原生指令，并通过硬件感知的优化编译遍生成高效代码，极大降低了跨芯片适配成本。目前计图已完成对寒武纪 MLU270 的深度移植，并实现与摩尔线程基于 GPU 的的兼容适配。基于计图的 JittorLLMs 等项目已经完成对数个主流国产 AI 芯片厂商的适配，形成了“一个国产框架、适配多家国产硬件”的自主可控生态。

总体来看，框架适配层通过“国际主流框架 + 国产插件”与“国产自研框架 + 多硬件适配”两条路径并行推进，使开发者既可以在熟悉的 PyTorch/TensorFlow 语境下“低迁移成本上国产芯片”，也可以选择完全国产的软件栈构建更高程度自主可控的 AI 系统。随着这两条路线不断成熟，开发者迁移与开发门槛不断降低，实现了生态开发者从“吸引过来”到“留下来”的转变。

2.4 管理监控层：系统稳定的“运维保障屏障”

管理监控层是 AI 芯片软件生态中负责系统运行维护和资源管控的模块，充当整个平台稳定性的“保障屏障”。随着 AI 训练集群规模日益扩大、任务愈发繁杂，如何监控硬件状态并调度资源变得至关重要。

成熟的算力生态往往配套完善的监控和调度系统：例如，NVIDIA 提供了 DCGM (数据中心 GPU 管理) 工具套件用于监控集群中 GPU 的利用率、温度、功耗等指标，并可以通过 Prometheus 等开源平台导出这些指标以实现集中监控[6]。在国产生态中，类似的运维能力也在构建中。如沐曦公司为其“曦云”系列 GPU 开发了 mx-smi 监控工具，可实时查询每块 GPU 的版本、温度、利用率、功耗等信息，并支持启停 GPU、固件升级等操作，功能上类似于 NVIDIA 的 nvidia-smi[7]。

又如，在集群管理方面，一些开源或商用调度系统被引入 AI 领域：Kubernetes 的设备插件机制已被广泛用于 GPU 资源管理，使得 GPU 成为容器可调度的资源[8]；Slurm 等传统 HPC 调度器也支持 GPU 作业调度，为多用户共享的算力中心提供批处理和队列管理。总体而言，管理监控层提供的就是这样的“看守”和“指挥”功能：看守指的是监控预警，及时捕捉硬件故障或性能异常；指挥即通过智能调度，让算力资源发挥最大效益并保障不同任务的服务质量。以下我们分别从监控工具和调度工具两个方面解析其作用。

2.4.1 监控工具：资源分配的“实时感知载体”

监控工具充当 AI 芯片运行状态的“实时感知载体”，帮助运维人员和上层调度实时了解系统健康和性能状况。对于大型 AI 训练集群，监控包括硬件层面的温度、电压、功耗、利用率，以及软件层面的算力利用率、显存占用、网络带宽等指标。例如前文提到的 NVIDIA DCGM 可收集 GPU 核心使用率、显存占用、温度功耗等数据，通过 DCGM-Exporter 将这些指标以 Prometheus 格式提供，从而与行业通用的监控系统对接[8]。NVIDIA Nsight Systems 则进一步地提供了系统级的性能分析功能，能够捕获 GPU 内核执行、内存传输、CPU 与 GPU 的交互等详细信息，帮助开发者识别性能瓶颈并优化应用。类似地，国产 AI 集群一般会将 GPU 或 NPU 的关键指标对接到现有运维平台。例如华为云的 ModelArts 平台支持用户接入 Prometheus 来获取 Ascend 昇腾集群的监控指标[9]。对于没有现成方案的新芯片，厂商通常提供专用工具：例如前述沐曦 mx-smi 通过命令行即可显示单卡的实时状态，包括利用率、温度、内存占用等[7]；寒武纪的 cambricon-smi 也提供类似功能以监控思元加速卡。更进一步的，像 HAMi + 沐曦 MetaX 的统一调度方案中，还开发了图形化界面，将异构指标贯通到 WebUI，直观展示整个集群的资源分配和使用情况，实现全链路的监控可视化。这种将监控与调度融合的思路，使得运维人员可以在一个平台上既看到硬件状态又能调整资源调度策略，从而更及时地处理故障和优化集群负载。

需要强调的是，监控工具的价值不仅在于“看”，更在于未雨绸缪的告警和事后分析。好的监控系统会针对 GPU 过热、显存泄漏、算力长时间闲置等情况发出告警，避免问题扩大影响业务连续性。同时，通过历史监控数据的分析，运维团队

可以发现资源瓶颈（例如某类任务总是显存打满但计算单元闲置）并指导上层优化算法或调整硬件部署。当前国产生态在监控领域仍有提升空间，一些先进功能（如预测性维护、智能诊断）有待引入。但可以看到各大厂商已经开始注重这一环节，以保障 AI 算力基础设施的稳定可靠运行——这是从“可用”走向“大规模好用”的必经之路。

2.4.2 调度工具：资源分配的“智能管控载体”

调度工具的核心任务是在多用户、多任务共享 AI 芯片集群的场景下，合理分配硬件资源，优化资源利用率并保障任务的 QoS（服务质量）。简单来说，它就是算力资源的“大脑和管家”，决定了哪个任务在何时、何地（哪块芯片）运行，以及运行多少实例。传统上，在 HPC 领域广泛使用的 Slurm 调度器已具备对 GPU 资源的管理能力，可以设置每个任务的 GPU 数量、队列优先级等[10]。在云计算和容器化趋势下，Kubernetes + GPU 设备插件成为新的主流方案，容器编排系统通过将 GPU 抽象为一种可调度资源，结合节点标签、NUMA 亲和等策略，实现灵活的算力调度[8]。无论哪种技术框架，智能调度的目标是一致的：提升集群吞吐，避免资源碎片，防止任务互相干扰。为此，现代 AI 调度工具引入了一些创新机制。例如，上述 HAMi v2.7 调度方案针对沐曦 MetaX GPU 提供了细粒度的 GPU 切分共享（sGPU）功能，允许多个容器共享同一物理 GPU 并精确限制每个任务使用的显存和计算比例，由此显著提高了资源利用率[11]。该方案还支持拓扑感知调度，在单机多卡的情况下，调度器会优先将多 GPU 任务分配到具有最高通信带宽的 GPU 组合上（例如共享同一 PCIe Switch 或直连 MetaXLink 的卡组），以提升分布式训练性能[4]。同时引入了三档 QoS 服务等级（BestEffort 尽力而为、FixedShare 固定份额、BurstShare 突发份额），按任务重要性划分资源保障程度，满足不同业务场景需求[12]。这些技术组合，使得大规模 AI 训练集群在确保重点任务性能的同时，实现了对碎片算力的充分挖掘[11]。

对国产 AI 芯片生态而言，调度工具还肩负一个特殊使命：降低生态切换成本。很多 AI 用户习惯于在 NVIDIA GPU 上使用现成的调度方案（如 K8s+NVIDIA 调度器），为了让他们平滑迁移到国产硬件，国产平台的调度系统需要兼容主流框架和接口。例如支持 Kubernetes 的 DevicePlugin 标准、支持 PyTorch 的分布式策

略等，让上层应用无需大改动即可运行在昇腾或寒武纪集群上。这方面已有积极进展：华为的昇腾云底座通过深度适配 K8s 和云原生调度，提供了类似 CUDA 环境的使用体验；一些第三方开源项目（如 Volcano 调度器、Fluid 框架）也在支持国产 AI 硬件。随着调度工具的智能化和标准化，未来 AI 算力池将能像水电一样被灵活调用。用户提交任务无需关心具体使用哪种芯片，由调度系统自动选择最佳的硬件和策略来运行。这将标志着国产 AI 芯片从“可用”迈向“好用”的关键一跃：只有当底层算力被高效且智能地调度起来，开发者才会真正感受到一个繁荣生态所带来的便利与价值。在这条道路上，国产 AI 芯片的软件生态正在稳步前进，逐步构筑起从基础支撑、核心工具、框架适配到管理监控的全栈能力，最终实现从“可用”到“好用”的跨越[5]。

湖南大学

三、国产 AI 芯片软件生态资源现状

3.1 国产 AI 芯片分类及代表性厂商

结合技术路线与核心应用场景，国产 AI 芯片当前已呈现出多元化的技术路径，大体可以按照主要承载的计算负载分为三类：一类是聚焦神经网络等智能算法的专用加速芯片，一类是同时服务于科学计算与智能计算的通用计算型芯片，一类是兼顾图形渲染与 AI 计算的图形计算型芯片。

其中，专用 AI 加速芯片以 NPU（神经网络处理器）及面向云端训练/推理的数据专用架构（DSA）芯片为代表。此类芯片普遍摒弃传统图形流水线与大规模双精度单元，围绕矩阵运算、张量计算等神经网络核心算子进行深度硬件优化，从而在 AI 核心任务上实现更高的算力密度与能效比。代表厂商包括华为昇腾与寒武纪：前者依托“鲲鹏+昇腾”体系打造覆盖“云、边、端”的全场景 AI 基础设施，后者则通过思元 MLU 系列芯片构建云端推训一体及边缘推理产品矩阵，在大模型训练、推理及行业化落地方面积累了较强的技术与市场基础。同时，以燧原科技等为代表的云端 AI 训练/推理加速芯片厂商，同样属于该类，其采用针对 AI 工作负载深度定制的专用架构，在大型模型训练和高并发推理场景下追求极致的性价比与能效表现。

面向“HPC+AI”融合场景的通用计算型芯片，则主要服务于科学计算、工程仿真与 AI 工作负载并存的数据中心和高性能计算平台。海光信息推出的 DCU 是这一技术路径的代表，其通过 DTK 软件平台全面对接 AMD ROCm/HIP 生态，在编程模型与库接口层面与 CUDA 保持高度相似，使原有 GPU/HPC 应用可以在较低迁移成本下平滑迁移至国产平台。该路线在保留高双精度与混合精度浮点能力的同时，为深度学习训练和推理提供可观算力与带宽，尤其适合需要同时运行传统数值仿真、MPI/Fortran 等高性能计算任务以及 AI 任务的科研机构与行业用户，满足“科学计算+智能计算”一体化的算力需求。

在通用 GPU 厂商中，摩尔线程与壁仞科技是两条代表性技术路径。摩尔线程官方定位为以“全功能 GPU”为核心的国产 GPU 厂商，基于自研 MUSA 架构及配套软件栈，其单芯片产品集成 AI 计算加速、图形渲染、物理仿真、科学计算和超

高清视频编解码等多类功能，可覆盖大模型训练/推理、可视化渲染、视频云和交互式图形等多元应用场景，并通过面向 CUDA 生态的迁移工具和主流深度学习框架适配，在一定程度上降低了存量开发者和软件资产向国产平台迁移的门槛。壁仞科技则更聚焦于高端通用 GPGPU 与大规模数据中心算力平台，其 BR 系列通用 GPU 及配套板卡和整机系统主要面向云数据中心、运营商和智算中心等场景，用于支撑大模型训练、AI 推理及高性能科学计算等通用计算负载，在算力密度和能效比上对标国际主流水平，成为“云端大算力”方向国产通用 GPU 的重要代表。

需要指出的是，部分厂商在不同产品线之间同时布局多条技术路径，例如沐曦旗下一方面通过曦思 N 系列切入 AI 专用加速芯片赛道，另一方面以具备较强高精度算力的曦云 C 系列服务“通用计算+AI”融合负载，同时还通过曦彩 G 系列覆盖图形渲染与可视化场景，因而应按芯片产品线而非企业主体进行分类。

3.2 国产 AI 芯片软件生态各环节资源汇总与完善度对比

本节在第二章“四层架构”（基础支撑层、核心工具层、框架适配层、管理监控层）的基础上，对国产 AI 芯片的软件栈资源进行梳理和评价。

表 3.1 国产 AI 芯片的软件栈资源

厂商	基础支撑层(工具)	核心工具层(工具)	框架适配层(工具)	管理监控层(工具)	官方开发者平台	开源仓库(Gitee/GitHub)
英伟达(行业标杆)	CUDA Driver / Runtime	NVCC 编译器, cuBLAS, cuDNN, NCCL 通信库	PyTorch (原生支持), TensorFlow (原生支持)	nvidia-smi, Nsight	https://developer.nvidia.com/	https://github.com/NVIDIA
华为昇腾	CANN Driver/Runtime	ATC 编译器，AOL 算子库，HCCL 通信库	MindSpore 后端，torch_npu 插件	npu-smi, MindStudio, Atlas Scheduler	https://ascend.developer.huaweicloud.com/home	https://gitee.com/ascend
摩尔线程	MUSA Driver/Runti	MUSA 编译器，	Torch-MUSA 插件	mthreads-smi, MUSA	https://developer.mthreads.com/	https://github.com/Moore

厂商	基础支撑层(工具)	核心工具层(工具)	框架适配层(工具)	管理监控层(工具)	官方开发者平台	开源仓库(Gitee/GitHub)
	me	muBLAS, muDNN		Dashboard		Threads/
寒武纪	NeuWare Driver/Runtime	MagicMind 编译器，CNNL 算子库，CNCL 通信库	torch_mlu 插件，TensorFlow 后端	cnmon, MLU Diagnostics	https://developer.cambricon.com/	https://gitee.com/cambricon
沐曦	MXMACA Driver/Runtime	MXMACA 编译器，MXNN 算子库，MCCL 通信库	PyTorch-MXMACA 插件	mx-smi	https://developer.metax-tech.com/	https://gitee.com/metax-ics
海光	DCU Driver/Runtime (兼容 HIP)	HIP 编译器，FFT 数学库	PyTorch 插件，TensorFlow 插件	hy-smi	https://developer.sourcefind.cn	https://gitee.com/anolis/hygon-devkit
壁仞	BIRENSUPA Driver/Runtime	BRCC 编译器，suDNN 算子库，SCCL 通信库	PyTorch 插件	暂未公开	https://developer.birentech.com/	https://gitee.com/BirenTechnology
燧原	云燧 Driver/Runtime	TopsCompiler 编译器，ENCCL 通信库	PyTorch 插件，TensorFlow 插件	暂未公开	https://www.enflame-tech.com/developer	https://github.com/EnflameTechnology
天数智芯	DayChip Driver/Runtime	DayCompile 编译器，DayOP 算子库	PyTorch-DayChip 插件，TensorFlow-DayChip 插件	ixsmi, ixManager	https://www.iluvatar.com/	https://ai.gitee.com/topics/iluvatar

（注：各工具详情及下载请访问对应厂商的“官方开发者平台”或“开源仓库”；标注 “暂未公开” 的工具或仓库，可通过厂商商务渠道咨询获取）

3.2.1 CUDA 生态对标及“兼容”的层次解析

在表中，我们以 CUDA 作为对标标杆，但如果不区分软硬件栈中的不同层次，就很难理解各家国产厂商口中的“支持 CUDA 生态”“兼容 CUDA 编程”的真实含义。同样一句宣传语，对于只写 Python 脚本的算法工程师、需要维护 C++/CUDA 内核的框架与 HPC 开发者、以及芯片厂商内部做驱动与 Runtime 的工程师而言，所对应的技术内涵和迁移成本完全不同。业界讨论“国产 GPU 是否要延续 CUDA/Stream 语义”这一问题，本质上也需要放在这样的分层框架下才能说清楚。

首先需要明确，“兼容 CUDA”绝不是指在硬件架构或驱动层面与 NVIDIA 做二进制级、一致性兼容。NVIDIA 的 GPU 指令集和 CUDA Driver/Runtime（针对 Ampere、Hopper 等架构）是封闭的、受专利保护的体系。国产 GPU（例如摩尔线程的 MUSA 架构）在硬件微架构、指令集以及驱动实现上均为完全自研，不可能在这一层做到“直接替换 NVIDIA 驱动即可使用 CUDA 程序”这样的兼容。当前国产厂商宣传语下所说的“兼容 CUDA”，主要是指在所谓“核心工具层”，即数学库、深度学习算子库、通信库等位置，提供与 CUDA 关键库在函数名、参数形式和语义行为上相同或高度相似的 API，从而实现源代码级的平滑迁移。

从开发者视角来看，整个 AI 软硬件栈可以粗略划分为三个层次。层次一，AI 框架应用层（面向算法工程师）。这包括绝大多数 AI 算法工程师、数据科学家，其主要工作聚焦于模型结构设计、训练策略、数据处理与推理部署等。他们的日常工作形态，一般是在 PyTorch、TensorFlow 等主流深度学习框架中编写 Python 训练和推理脚本，通过 tensor.to("cuda")、model.cuda() 或设置 device="cuda:0" 等方式选择计算设备，依赖框架内置或第三方提供的高层 API，很少直接接触 cuBLAS、cuDNN、NCCL 等底层库。对于这部分开发者而言，无论底层芯片采取“API 兼容 CUDA”（如摩尔线程，通过 Torch-MUSA 等插件提供 tensor.to("musa")）还是“自有 API 体系”（如华为昇腾，通过 torch_npu 提供 tensor.to("npu")），厂商都会通过框架插件把底层复杂性屏蔽掉，他们通常只需要改动设备字符串、安装相应 wheel 包与驱动即可完成迁移。因此，在这一

层上，“是否 CUDA API 兼容”并不是决定性因素，迁移体验好坏更多取决于框架插件是否成熟、算子覆盖是否完整以及性能与稳定性是否达标。

层次二，核心工具层（面向 HPC 与框架开发者）。这一层面主要是高性能计算工程师、深度学习框架与中间件开发者，以及需要手写 C++/CUDA 自定义算子的资深算法工程师。他们直接与 cuBLAS、cuDNN、NCCL 等库打交道，编写 .cu 内核并使用 NVCC 进行编译，向上为 PyTorch、TensorFlow 等框架提供高性能算子与通信组件，并通过多 Stream 与事件机制实现通信与计算的异步重叠。在这一层，“兼容 CUDA”与否会带来数量级不同的迁移工作量。如果走“API 兼容”路径（以摩尔线程 MUSA 为代表），厂商会提供 muBLAS、muDNN 等库，其函数命名、参数列表、调用语义与 cuBLAS、cuDNN 尽量保持一致，对于一个已经大量调用 cuBLAS/cuDNN 的 C++ 项目，往往只需在构建脚本中将链接库从 -lcublas -lcudnn 换成 -lmublas -lmudnn，并使用 MUSA 对应的编译工具链重新编译，即可在国产 GPU 上跑起来，只在个别未覆盖或语义略有差异的接口处做小范围修改。相反，如果走“API 不兼容、构建自有生态”的路径（以华为昇腾 CANN 为代表），则需要将原有调用 cuBLAS、cuDNN、NCCL 的代码整体改写为调用 AOL、HCCL 等新接口，自定义内核也要按 CANN 的规范重写、改用 ATC 等工具完成编译与部署。对这类用户来说，迁移的含义已经从“适配新平台”变成了“重写一套底层实现”。因此，核心工具层才是“CUDA 兼容性”价值的关键所在：它直接决定了既有 CUDA 代码资产能否在国产平台上以较低成本继续发挥作用。

层次三，基础支撑层（驱动与 Runtime 层）。这一层面主要面向芯片厂商内部的硬件、驱动与 Runtime 工程师，以及与之对接的操作系统与虚拟化平台开发者。其工作关注 GPU/NPU 指令集设计、任务调度与内存管理机制、设备驱动程序以及底层 Runtime 库实现等。这一层与具体硬件架构深度绑定，NVIDIA 自身的 CUDA Driver/Runtime 与 Ampere、Hopper 等架构强耦合，且为封闭实现，国产厂商不可能在这一层做“兼容 CUDA”的迁移，只能在理念上对标，比如提供类似 nvidia-smi 的监控工具和类似 CUDA Runtime 的编程模型，但底层实现必然是一整套完全不同的体系（如 MUSA Driver、CANN Driver 等）。对普通开发者而言，这一层几乎没有所谓“迁移路径”，更多是厂商内部工程能力和生态建设的问题。

综合来看，“兼容 CUDA”本质上是一个面向“核心工具层”的工程承诺，它的直接受益者是那些积累了大量 C++/CUDA 代码、需要维护高性能算子和通信组件的框架团队与 HPC 团队。对于停留在框架应用层的大多数算法工程师来说，更需要关注的是：厂商是否提供成熟的 PyTorch 等深度学习框架插件，是否支持自身常用的模型与算子，以及实际性能、稳定性和调试体验是否达标；对于深度依赖 CUDA 自定义算子和 HPC 代码的团队，则需要重点考察国产厂商在核心工具层的 API 兼容程度、文档和样例的完备性以及工程支持能力。放在上述分层框架下来看，业内常说的“延续 CUDA/Stream 语义”，其实更多是指在核心工具层与框架后端继续采用“流/队列 + 异步核函数 + 事件同步”这一类编程抽象，而不是在指令集或驱动层做二进制级兼容。当前国产方案的总体趋势是：在不牺牲自研指令集与硬件架构创新空间的前提下，尽可能在 API 设计与工具链形态上对齐 CUDA 生态的使用习惯，从而在 C/C++ 开发体验和运维调优方式上降低开发者的迁移门槛。

3.2.2 国产 AI 芯片软件生态完善度评价

在前文资源梳理的基础上，本小节结合上表，从基础支撑、核心工具、框架适配和管理监控四个层次，对国内主流 AI 芯片的软件生态进行分层评价，力图在与 NVIDIA CUDA 体系对比的视角下，勾勒出各厂商当前的可用程度与主要短板。

3.2.2.1 华为昇腾

在基础支撑层，CANN Driver/Runtime 提供了较为稳定的驱动、Runtime 与固件管理，主流 Linux 发行版与容器镜像均已支持，日常使用中驱动安装和升级流程相对规范，开发者可以类比 CUDA Toolkit 的体验来理解这一层。

在核心工具层，ATC 编译器、ACL/AOL 算子库以及 HCCL 通信库覆盖了常见 CNN、Transformer 等模型的主流算子和分布式通信路径，官方和第三方实践表明，大模型训练经过针对性调优后可以达到较高的硬件利用率，但新模型结构和长尾算子仍需依赖厂商支持或自行扩展。

在框架适配层，华为一方面通过 MindSpore 提供“原生后端”，另一方面通过 torch_npu 等插件适配 PyTorch 等主流框架，典型训练 / 推理脚本一般只需替换设备类型与后端初始化即可迁移，但与 CUDA 相比，三方库支持度、社区教程数量仍然偏少。

在管理监控层，npu-smi、MindStudio 等工具基本覆盖了设备监控、Profiling 和瓶颈分析需求，支持对算子耗时、带宽利用率等进行定位，整体功能上可对标 nvidia-smi + Nsight 组合，但生态中尚缺少更多由社区驱动的调优脚本与可视化工具。

整体来看，昇腾路线更适合对长期生态建设、自主可控要求较高、能够接受一定迁移投入的大型项目或公共算力平台，在“工具链齐全”方面已具备较强可用性，与 CUDA 相比主要差异集中在开发者积累与第三方生态厚度上，仍处于持续扩展阶段。

3.2.2.2 摩尔线程

摩尔线程的 MUSA 软件栈在设计上明显以“兼容 CUDA 开发体验”为目标，整体定位是通用 GPU 平台，既覆盖图形渲染，也支持 AI 训练与推理。

在基础支撑层，MUSA Driver/Runtime 已经形成较稳定的驱动与运行时发布节奏，支持主流 x86 Linux 发行版，并提供含 muDNN 等库的官方容器镜像，基础安装与运行环境配置相对可控。

在核心工具层，MUSA Toolkit 内集成了编译器、muBLAS、muDNN 等对标 cuBLAS/cuDNN 的库，torch_musa 中提供了与 CUDAExtension 接口保持一致的 MUSAExtension，使现有 C++/CUDA 扩展在较小改动下即可迁移到 MUSA 平台，但在某些高阶算子、长序列 Transformer 等场景的优化经验仍在积累。

在框架适配层，PyTorch 通过 torch_musa 后端可以完成主流训练和推理流程，已有基于 MUSA 的大模型推理教程，日常开发者主要通过“更换后端 + 安装插件”的方式完成迁移；相比 CUDA，第三方框架和工具链（如部分推理引擎、AutoML 工具）的适配覆盖仍不完整。

在管理监控层，mthreads-smi、MUSA Dashboard 等工具提供了基础的设备状态查询和资源监控能力，但在细粒度性能分析、自动化调优脚本和集群级调度工具方面，仍然处于建设阶段，整体调优体验与 NVIDIA Nsight、DCGM 等工具体系相比还有差距。

总体而言，摩尔线程的优势在于延续 C++/CUDA 开发习惯，适合存量 CUDA 代码资产较多、希望在图形 + AI 复合场景中平滑引入国产 GPU 的用户；在大规模训练与精细化运维场景，其软件栈仍处于稳步演进过程中。

3.2.2.3 寒武纪

寒武纪 NeuWare 软件栈在推理场景上的成熟度较高，训练侧生态相对有限，更适合作为行业推理平台而非通用大模型训练底座。

在基础支撑层，NeuWare Driver/Runtime 负责 MLU 设备的驱动与资源管理，配套的 cnmon 工具可以查看卡状态、温度和利用率，整体功能上可类比 CUDA Driver + nvidia-smi。

在核心工具层，MagicMind 推理引擎以及 CNNL、CNCL 等库构成了主要能力边界：前者提供基于 MLIR 的图编译与算子优化，面向多框架推理部署；后者覆盖绝大部分常见算子和分布式通信需求。推理模型（特别是 CV 方向）的性能调优工具和最佳实践文档相对丰富，而训练相关库与工具则相对薄弱。

在框架适配层，寒武纪提供了 torch_mlu/CATCH 等 PyTorch 扩展，以及 TensorFlow 等后端适配，主流推理工作流可以通过官方容器和示例工程较快跑通，但在前沿模型和多框架协同方面与 CUDA 有明显差距。

在管理监控层，除基本状态监控外，MagicMind 也提供了 Profiling 和性能分析工具，用于定位算子层面的瓶颈，不过在集群级监控与资源调度软件上仍相对依赖上层通用平台。

因此，寒武纪当前生态更适合算子形态相对稳定、批量推理需求明确的业务场景；对于需要高频迭代模型结构的大模型训练任务，仍需要在迁移与调优层面投入更多工程建设。

3.2.2.4 沐曦

沐曦基于 MXMACA 异构计算平台构建软件栈，目标是同时覆盖通用计算与 AI 训练 / 推理，目前整体处于快速演进阶段。

在基础支撑层，MXMACA Driver/Runtime 为曦思 N、曦云 C 等产品提供统一的驱动和运行时抽象，支持虚拟化、多实例划分等数据中心场景，基本满足通用 GPU 的部署需求。

在核心工具层，MXMACA 编译器与 MXNN 算子库、MCCL 通信库构成主要能力，官方资料显示已覆盖主流训练与推理算子，并在数据中心场景支持端到端数据处理流程；但与 CUDA 相比，文档完备度和长尾算子支持仍不够充分，高级调优工具也在持续完善中。

在框架适配层，沐曦提供了与 PyTorch 对接的插件和示例工程，且已完成部分大模型推理框架（如 Xorbits Inference）在曦云 C 系列上的适配，能在单卡完成中大型模型推理，但整体生态规模与 CUDA 仍有明显差距。

在管理监控层，当前更多依赖通用容器平台和自研运维工具，设备监控与资源管理能力基本可用，但在大规模集群调度与一体化运维平台上仍处于建设期。

整体来看，沐曦的软件栈在“通用 GPU + 大模型推理 / 训练”方向展现出一定潜力，对普通开发者而言，随着文档、示例与工具链的持续丰富，上手门槛有望进一步降低。

3.2.2.5 海光

海光 DCU 从整体软件路线看，主要基于 AMD ROCm/HIP 软件栈构建，开发模式与 CUDA 有一定相似性，但在库与工具层更依托 ROCm 生态，整体定位于服务“HPC + AI” 融合负载的数据中心。

在基础支撑层，DCU Driver/Runtime 采用 GPU 类通用加速器架构，支持 CPU－DCU 异构部署，已有研究和实际应用表明，其在并行度较高的科学计算任务中可以在一定程度上替代传统 GPU 工作负载。

在核心工具层，DCU 通过 HIP 兼容接口以及配套数学库等组件复用 ROCm 上层生态，使既有 HIP/ROCm 应用在较小改动下即可迁移，但面向大模型训练的专用算子库和高阶性能调优工具仍相对有限。

在框架适配层，社区和厂商已完成对 PyTorch、Paddle 等主流深度学习框架的适配，并在新版本中增加了 DCU 侧的 Profiling 等功能，能够支撑常见训练与推理任务；与 CUDA 平台相比，目前公开的最佳实践、调优案例和大规模分布式训练经验仍在逐步积累。

在管理监控层，海光提供了类似 nvidia-smi 的 hy-smi 工具，用于查看 DCU 的利用率、显存占用和温度等运行状态，基本满足单机运维监控需求；在集群

侧，则依托 HAMi 这一异构设备虚拟化中间件，将 DCU 纳入 Kubernetes 集群统一纳管与调度，实现设备复用和拓扑感知调度，该层在国产生态中相对成熟，但整体效果仍受制于云原生与容器平台的持续演进。

总体而言，海光 DCU + HAMi 方案适合已有 HPC 基础、希望逐步引入 AI 工作负载的机构，在“统一调度异构设备、兼顾 HPC 与 AI”方面具有一定优势，面向大模型场景的专用工具链和实战经验仍在不断充实之中。

3.2.2.6 壁仞

壁仞 BR 系列 GPGPU 在硬件算力指标上对标高端数据中心 GPU，但其软件栈公开信息相对有限，整体更偏向“深度合作 + 厂商支持”模式。

在基础支撑层，BIRENSUPA Driver/Runtime 提供了设备驱动与资源管理能力，能满足数据中心部署需求，但具体支持的 OS、虚拟化形态等细节暂未形成完全公开的技术文档。

在核心工具层，官方介绍中提及 BRCC 编译器、suDNN 算子库和 SCCL 通信库等组件，用于服务大模型训练和高性能计算，但 API 细节和算子覆盖度的公开资料较少，开发者往往需要依赖厂商提供的 SDK 与示例进行适配。

在框架适配层，壁仞提供了面向 PyTorch 的插件和若干行业应用示例，可以支撑基础训练与推理流程，但与 CUDA 平台相比，公开的大型模型适配案例和第三方框架支持仍然有限。

在管理监控层，目前公开信息显示主要依赖厂商内部工具和合作方平台，nvidia-smi / Nsight 级别的通用监控与分析工具尚未在社区中形成统一认知。

整体来看，壁仞方案在大算力场景中具备发展潜力，更契合具备较强工程能力、并愿意与厂商建立紧密合作关系的用户；对于希望主要依托公开资料和社区力量完成迁移的团队，其生态成熟度仍需结合具体项目进行评估。

3.2.2.7 燧原

燧原的软件栈围绕云端推理场景做了较深的纵向优化，在训练和通用计算侧则相对克制。

在基础支撑层，云燧 Driver/Runtime 提供了面向数据中心的驱动与资源管理能力，可在主流服务器平台部署，但公开资料更聚焦于具体解决方案而非底层接口细节。

在核心工具层，TopsCompiler 与 ENCCL 构成推理优化的主干：前者负责模型图优化和部署，后者负责多卡协同与通信加速，在高并发推理场景下能提供较高的吞吐与能效；与 CUDA TensorRT 体系相比，算子类型和应用场景更集中。

在框架适配层，官方已完成对 PyTorch、TensorFlow 等的推理路径适配，开发者通常通过导出 ONNX / 中间格式，再由 TopsCompiler 完成部署；完整的训练支持则相对有限。

在管理监控层，局部公开了运维工具和监控接口，但整体更多依赖于合作云平台或自建监控系统，尚未形成类似 CUDA+NVIDIA 生态那样的统一工具体系。

整体而言，燧原更适合已有成熟模型、重点关注云端推理成本与吞吐的场景，在“训练 + 通用计算”方向的软件生态仍处于循序拓展阶段。

3.2.2.8 天数智芯

天数智芯围绕 DeepSpark 软件生态构建通用 GPU 平台，重点发力 AI 推理与行业应用落地。

在基础支撑层，DayChip/CoreX Driver/Runtime 为 GPU 设备提供驱动与运行时支持，配套的安装包与系统镜像主要面向数据中心和边缘服务器场景，基本满足常规部署需求。

在核心工具层，DayCompile 编译器与 DayOP 算子库构成基础能力：前者面向自家 GPU 做模型图编译与算子生成，后者提供常用深度学习算子的实现与优化；在此基础上，进一步叠加 IxRT 推理引擎和基于 TVM 的 IGIE 图优化框架，用于完成端到端的图优化和推理加速。这一层在推理场景的功能密度较高，但通用训练工具链和面向复杂大模型的专用优化能力仍相对薄弱。

在框架适配层，DeepSparkInference 等项目提供了覆盖 CV、NLP、语音等领域的模型示例，支持在 IGIE/IxRT 上运行，并配套评测结果与部署脚本，降低了在天数 GPU 上进行推理部署的上手门槛；不过与 CUDA+TensorRT 生态相比，模型数量、更新频率以及社区维护规模仍有限。

在管理监控层，ixsmi 提供类似 nvidia-smi 的基础监控能力，支持查看设备利用率、显存占用和温度等信息，DeepSpark 平台则在此基础上集成部分评测与管理功能，但针对大规模集群场景的生产级监控与调度能力仍有待进一步验证和完善。

整体上，天数智芯现有软件栈在围绕推理与应用场景构建紧凑一体化方案方面具有一定优势，适用于对成本敏感、希望在国产 GPU 上快速落地多领域推理任务的用户；在训练和高性能科学计算等方向的支持能力仍处于有序补齐阶段。

3.3 国产 AI 芯片软件社区活跃度对比数据表

本节旨在从“官方软件栈资料公开度”“开发者社区讨论活跃度”和“开源代码仓库活跃度”三个维度，对典型国产 AI 芯片生态与 NVIDIA CUDA 进行横向对比，为读者在选型时提供直观参考。这三个维度一方面反映厂商在文档、工具链和开源社区上的长期投入与贡献度，另一方面也体现开发者参与度和生态自发活力，从而帮助用户在性能指标之外，综合判断后续使用过程中的学习成本、问题排查效率以及生态可持续性。

表 3.2　国产 AI 芯片软件社区活跃度对比数据表

平台/生态	官方软件栈资料公开度（定性等级）	开发者社区讨论活跃度（近一年概况）	开源代码仓库活跃度（官方账号总体情况）
NVIDIA CUDA（行业标杆）	A+：CUDA Toolkit、cuDNN、NCCL、TensorRT / Triton 等组件文档极其完备，官方提供成体系的编程手册、最佳实践指南、GTC 课程与 MOOC，第三方教材与博客数量也极为可观。	A+：NVIDIA Developer Forums 与 Technical Blog 形成高黏性社区，仅技术博客 2022 年就发布 550+ 篇文章、年访问量 200 万级；论坛覆盖 CUDA、HPC、Omniverse 等数十个板块，长期维持“十万级”话题与回复规模。	A+：GitHub 组织 NVIDIA 下约 600 个仓库、近 2 万关注者，cutlass、tensorrt-llm、triton-inference-server 等核心项目 star 数合计达数万级，绝大部分仓库保持周级甚至日级更新。
华为昇腾 + MindSpore	A：以昇腾 CANN + MindSpore + 华为云 ModelArts 为核心，形成软硬一体的软件栈。官网公开安装指引、API 文档、算子开发手册以及大规模 Ascend 全栈培训与认证课程，并提供针对不同行业场景的示例与案例教程。	A：昇思 MindSpore 社区与华为云开发者社区下的 Ascend/MindSpore 板块较为活跃，官方活动中半年征集技术干货已达百篇量级，论坛与问答区每年新增技术帖和问题解答在“数百篇”左右，形成相对稳定的问答	A：GitHub 组织 mindspore-ai 目前约有 40+ 个公开仓库，核心仓库 mindspore 的 star 约 4.6k，配套有 docs、models、book、hub 等多类功能型仓库；整体 star 总量在 5k+ 级别，主仓库保持月度乃至周级

平台/生态	官方软件栈资料公开度 （定性等级）	开发者社区讨论活跃度 （近一年概况）	开源代码仓库活跃度 （官方账号总体情况）
		闭环。	提交，体现出较高的持续维护投入。
摩尔线程 MUSA	A-：提供 MUSA SDK、编程指南、算子开发文档以及面向高校/企业的系列课程，基础工具链和入门示例较为齐全，但在高级调优、生态整合（如与主流框架的深度融合）文档仍在逐步完善。	B+：目前以官网开发者中心与培训课程为主，尚未形成类似“大型技术论坛”的集中讨论区，开发者交流更多依赖课程评论区、微信群/公众号等渠道，公开可见的话题量在“百贴级”。	A-：GitHub 组织 MooreThreads 公开仓库约 30+，其中 Moore-AnimateAnyone 等代表性项目 star 总量接近 3.5k，torch_musa 在过去一年持续有功能更新与 bug 修复，代码维护频率较高。
寒武纪 Cambricon	B+：提供 Cambricon Neuware、BANG / BANGPy 等软件栈文档与课程资料，覆盖基础安装、算子编程和性能调优，多数资料需在开发者平台注册后获取，公开程度略低于 CUDA / 昇腾。	B：官方开发者社区分为课程区、活动区等板块，公开统计显示课程区已有数百主题帖与数百条回复，整体讨论规模在“百贴级”量级，活跃度中等偏上，但跨领域技术分享相对有限。	B+：GitHub 组织 Cambricon 目前有约 20+ 仓库，以 mlu-ops 等算子库为核心，大部分仓库 star 数在几十到百级，总体 star 规模为“数百级”，更新节奏以版本发布为主。
沐曦 MetaX	B+：提供面向 MXC/G 系列 GPU 的编程模型与工具链文档，官方站点给出了驱动安装、运行时接口说明以及典型推理/训练案例，但高阶优化与第三方框架适配文档仍在补齐过程中。	B：沐曦开发者社区设有多块 GPU 产品与软件栈讨论区，部分话题浏览量达到千级，整体发帖与回复量在“百贴级”上下，属于快速起步、尚在扩张的生态。	B：官方开源主要分布在 GitHub 组织 MetaX-MACA（目前公开仓库 33 个）以及少量 Gitee 镜像下，以编译器插件、GPU 算子库、模型示例和性能评测为主；其中 mcFaiss、mcTVM 等核心仓库 star 多为数百到数千级，整体 star 总量达到“数千级”量级，体现出在算子优化与推理引擎方向较为集中的开源投入。
海光 DCU + HAMi	B+：海光 DCU 适配主流深度学习框架（如 PyTorch、Paddle 等）的文档由官方与社区共同维护，其中 HAMi 项目为国产算力统一软件栈提供了较系统的说明与使用示例。	B：线下面向 DCU / HAMi 的开发者沙龙与培训活动较多，线上集中论坛相对较弱，更多讨论散布在开源社区 issue、邮件列表和技术博客中，整体讨论规模为“几十到百贴级”。	A-：核心开源项目为 GitHub 上的 Project-HAMi/HAMi（CNCF Sandbox，Linux Foundation LFX Insights 统计）。截至 2025 年，HAMi 仓库约 2.6k GitHub stars、416 forks，过去一年中提交或

平台/生态	官方软件栈资料公开度 （定性等级）	开发者社区讨论活跃度 （近一年概况）	开源代码仓库活跃度 （官方账号总体情况）
			issue/PR 活动频繁，最近一个季度活跃贡献者约 86 人。在国产统一算力栈方向的开源项目中，活跃度处于第一梯队。
壁仞科技 Biren	B：提供 BIRENSUPA 等软件平台的开发文档与工具说明，在模型分析、性能评估等方面配套若干技术文章与培训材料，但对外公开的细粒度 API 文档相对有限。	B-：目前主要通过官网技术专栏、合作平台博客和线下开发者活动进行技术传播，缺乏大型开放 BBS，公开可见的互动讨论量相对其他头部厂商偏少。	B：壁仞相关的开源仓库资料目前主要以与高校合作项目的形式存在。典型代表是挂在 GitHub 组织 BRTResearch 名下的 AIChip_Paper_List 仓库，由上海交大 ACA Lab 与 Biren Research 联合维护，主要整理 AI/ML/DL 加速器与芯片架构相关论文，当前 star 约 600 个，在 AI 芯片文献整理方向具有一定影响力。整体来看，尚未形成以“官方企业账号 + 完整软件栈”为中心的大规模开源版图，开源活跃度处于中等水平。
燧原科技 Enflame	B+：燧原软件栈围绕训练卡/推理卡提供编程手册、算子库说明和案例教程，官方网站公开了较完整的 SDK 文档和 FAQ，对主流训练/推理框架给出了适配说明。	B：官方多次举办“AI 芯片开发者论坛”等活动，并在合作社区发布技术文章与实践案例，线上集中论坛规模不大，但叠加线下会议形成“活动驱动型”社区形态。	B+：GitHub 组织 EnflameTechnology 目前有 20+ 仓库、数十名关注者，在编译工具链、算子库和示例工程上保持持续提交，部分仓库 star 数达到几十级，整体活跃度中等偏上。
天数智芯 Iluvatar + DeepSpark	B+：Iluvatar CoreX 提供推理引擎 IxRT 与编译工具链文档，基础安装与接口说明较为完整；DeepSpark 平台从应用模型与场景角度补充大量部署示例与指导。	A-：DeepSpark 社区定位为软硬件协同的开放平台，公开资料显示已支持数百个模型与上百家合作伙伴，技术博客与教程更新频率较高，综合来看讨论与使用反馈具有较高活跃度。	A-：GitHub 组织 Deep-Spark 下有约 20 个仓库，核心仓库 DeepSpark、DeepSparkHub、iluvatar-corex-ixrt 等分别拥有数十 star，并保持年度版本迭代；结合 Gitee 镜像，整体 star 规模达到“数百级”，在国产 GPU 中开源力度相对靠

平台/生态	官方软件栈资料公开度 （定性等级）	开发者社区讨论活跃度 （近一年概况）	开源代码仓库活跃度 （官方账号总体情况）
			前。

从官方软件栈资料看，CUDA 与昇腾处于绝对第一梯队：前者依托多年全球开发者生态，在 API 说明、最佳实践与教学资源上几乎“想得到的都能找到”；后者凭借完整的 CANN + MindSpore + 云服务，在国产方案中形成最接近 CUDA 的文档与培训体系。摩尔线程、寒武纪、沐曦、燧原、天数等厂商的软件栈文档基本覆盖了安装与常规开发路径，但在高阶性能调优、跨平台协同等方面仍有补齐空间；海光 DCU / HAMi 与壁仞则更多依赖社区文档与技术专栏，资料分散度略高、系统性略弱。对普通开发者而言，这一维度直接影响“上手难度”和排错效率。

社区活跃度在很大程度上体现用户参与度与生态自驱力。NVIDIA 依托 Developer Forums 与高产的 Technical Blog，长期维持十万级主题与回复，几乎任何问题都能在社区中找到类似案例；昇腾 / MindSpore 依托华为云开发者社区和昇思论坛，技术干货征集与问答活动频繁，形成相对稳定的用户群体。相比之下，多数国产 GPU 厂商仍处于“社区建设期”：官方论坛或讨论区的规模多为几十到数百主题，部分讨论被分散在微信群、公众号和合作社区中，信息查找成本相对更高。表中给出的等级与示意性的发帖量，意在帮助读者快速理解“能否找到同路人”和“遇到问题时有没有人回答”。

开源仓库活跃度主要反映厂商及其生态伙伴对公共代码库的持续投入。NVIDIA 的开源版图最大，不仅仓库数、star 数远超其他厂商，而且在编译器、算子库、推理引擎等关键组件上均有高频提交；昇腾 / MindSpore 与 HAMi 则代表了国产阵营中开源投入最系统的两条路线，前者以框架为中心，后者以统一软件栈为核心，都在 star、fork 与贡献者规模上达到“千级”量级。摩尔线程、寒武纪、沐曦、燧原、天数与壁仞等厂商的公开仓库数多在几十个上下，整体 star 规模从数百到数千不等，其中部分项目（如 torch_musa、DeepSparkHub 等）已经具备较强的生态外溢效应。对普通用户而言，这一维度意味着是否容易找到官方或社区维护的示例工程、适配脚本和调试工具。

综合以上三个维度，可以将“官方资料完善度”视为厂商自身投入的窗口，将“社区讨论活跃度”和“开源仓库活跃度”视为用户参与度与生态健康度的外在体现。在实际选型时，性能指标固然重要，但若目标项目高度依赖社区经验迁移、第三方工具以及自定义算子开发，则应优先考虑在三个维度上整体评分更高的平台；反之，对于高度定制、以厂商直连技术支持为主的场景，也可以在保证基本资料完备的前提下，适当权衡社区规模与开源体量。

3.4 本章小结

从开发者视角看，当前国产 AI 芯片在框架适配层已经基本能够支撑主流深度学习工作流，算法工程师在绝大多数场景下可以通过“改设备类型 + 安装插件”的方式完成迁移；在核心工具层和管理监控层，各家厂商在 API 兼容度、性能调优工具和集群运维能力上仍存在明显差距，但整体正沿着“对标 CUDA/HIP 开发体验、保留自研架构自主性”的方向持续演进。

从更宏观的生态视角看，国产 AI 芯片软件生态也已逐步摆脱早期“各自为战”的状态，初步形成了多技术路径并行、厂商间差异化发展的格局。生态建设大致呈现两大主流方向：一方面，以华为昇腾为代表的“全栈生态”路径，依托较全面的工具覆盖和相对成熟的端到端方案形成合力；另一方面，以摩尔线程、海光为代表的“兼容生态”路径，通过在接口和工具上对标 CUDA/HIP，显著降低用户的迁移成本。同时，应用场景的专业化特征愈发明显，例如寒武纪、燧原科技等厂商聚焦特定细分领域，以能效比或性价比为切入点，满足差异化的市场需求。

在此基础上，行业协同也开始出现雏形。一方面，中国电子技术标准化研究院牵头制定生态兼容性标准，推动不同厂商之间的接口互通；另一方面，华为通过开放 CANN 核心模块、摩尔线程通过兼容 CUDA 的技术路线，为跨厂商技术迁移提供了基础条件；同时，地方政府主导的智算中心在实际部署中采用多厂商芯片混合架构，通过统一调度实现算力的协同输出。

总体而言，需要保持一种既乐观又克制的判断：国产生态已经从“基础可用”逐步走向“特定场景可用”，但在工具链完备性、整体生态成熟度以及开发者基础规模等方面，与国际主流生态相比仍存在明显差距。因此，企业在技术选型时应立

足于自身业务需求：明确训练与推理负载、公有云与私有化部署等具体场景，评估现有技术栈的迁移成本与可行性，并审慎考察厂商在相关行业的实际落地案例。选择与自身需求相匹配的方案，而非简单追逐“国产”或“对标某一家厂商”，才是平衡技术投入与业务产出的关键。

湖南大学

四、结语

随着大模型的需求井喷，国产 AI 芯片迎来了蓬勃发展，也进入了由“硬拼算力”到“生态好用”转型的关键时期，当下逐步形成了以华为昇腾为代表的“全栈生态”与以摩尔线程、海光为代表的“兼容生态”两大主流路径，同时寒武纪、燧原科技等厂商在特定推理场景的深耕，彰显了“场景为王”的差异化竞争趋势 。然而，我们必须清醒地认识到，与国际主流生态相比，国产软件生态在工具链丰富性、成熟度及开发者基础方面仍有差距。因此，回归业务本质、评估技术栈兼容性、考察行业落地案例，已成为务实选型的关键。

展望未来，唯有坚持“标准化、开源化、协同化”的发展路径，才能真正构建起自主、开放且富有活力的技术体系 。这需要产学研各方持续投入，推动技术标准化、工具开源化与产业协同化。我们期待，在各界的共同努力下，国产 AI 芯片软件生态能够完成从“好用”到“卓越”的最终跨越，为我国在全球科技竞争中赢得战略主动权提供坚实支撑。

湖南大学

附录一 AI 芯片硬件基础：理解软件生态所“指挥”的对象

AI 芯片与传统 CPU 的核心区别在于：AI 芯片采用大规模并行计算架构（数千个计算单元），而非 CPU 的顺序处理模式。这一硬件特性决定了软件生态需支持高并发任务调度与内存优化。例如，单颗 GPU 常按 32 线程为一组“warp”并行执行指令（单指令多线程，即 SIMT 模式）[13]；开发者需要将大任务拆分为许多小任务，以充分利用数千个并行线程。又如，GPU 采用高带宽的片上高速缓存和显存（如 HBM 高带宽显存），其数据传输速度可达数 TB/s 量级，远超 CPU 主存几十 GB/s 的带宽。但要充分释放这些硬件潜能，软硬件协同的生态至关重要。

计算核心：执行计算的“工人”。AI 芯片的计算核心负责执行矩阵乘法等并行算术操作。不同芯片架构采用了不同的计算单元组织方式，软件优化策略也需随之调整。主流 GPU 采用单指令多线程（SIMT）架构，一个“warp”包含 32 个同时执行同一指令的线程[13]。例如 NVIDIA Ampere 和 AMD RDNA 架构都通过这种并行模型，让上万线程同步运行以加速深度学习等任务。国内 GPU 厂商也沿用类似设计：如摩尔线程（Moore Threads）GPU 采用统一的 MUSA 架构实现图形与计算融合，其软件栈提供 C/C++ 并行编程环境，支持数千内核的并行加速[15]。大量简单“工人”并行工作的模式要求上层软件具备高并发任务拆分和调度能力，确保每个计算核心都被充分利用[5]。

存储层次：数据流动的“高速公路”。AI 芯片通常具有分级存储体系，从寄存器、片上缓存、共享内存到高带宽显存，形成数据流动的“高速公路”。相比 CPU 主要依赖几十 GB/s 带宽的 DRAM 内存，AI 芯片的显存（如 HBM）提供了数量级提升的带宽。例如，NVIDIA Hopper 架构 GPU 配备 80GB HBM3 显存。如此巨大的带宽差异要求软件充分利用片上高速缓存和显存，减少数据在各层级之间的不必要搬运。国内 AI 芯片在这方面逐步缩小差距：寒武纪思元系列芯片通过 Cambricon NeuWare 软件平台提供了高效的张量算子库（CNNL）和通信库（CNCL），配合多级存储优化，在实际训练任务中实现业界领先的计算和通信效率[2]。软件生态需针对硬件的存储特性进行内存优化和算子融合，避免“内存墙”导致计算单元空转[1]。

互连结构：多芯片协同的“通信网络”。大型 AI 模型的训练往往需要多卡甚至多节点协同，芯片间互连架构决定了数据同步效率。当前 GPU 集群采用高速互连总线提供远高于 PCIe 的带宽，可在多 GPU 间建立直连通信。NVIDIA 900 GB/s 带宽的 NVLINK[11]这类高速“通信网络”让多卡间参数同步和模型拆分更高效，也催生了诸如 NCCL 等分布式通信库用于协调多 GPU 训练。国内厂商在互连技术上也积极布局：例如沐曦（MetaX）开发了自研的 MetaXLink 芯片互连，并支持拓扑感知的调度策略，在单机多卡场景下优先分配高速互联域内的 GPU 组合，显著提升多卡训练吞吐[4]。而华为宣称其 CloudMatrix 384 集群搭载的昇腾 910C NPU 单 Die 具有 196GB/s 的“灵衢”互联带宽，双 Die 聚合可达 392GB/s。总体而言，互连带宽和拓扑对软件分布式训练框架提出了要求：通信算法需要针对互连结构优化，尽可能减少跨节点的数据交换瓶颈。

指令集：软件控制硬件的“语言”。硬件只能执行机器指令，因此软件生态中的编译器和运行时最终必须将高层代码翻译为特定架构的指令集。这相当于软件与芯片交流的“语言”。以 NVIDIA 平台为例，开发者在 PyTorch 等框架中编写的模型，底层需经过 CUDA 编译工具链转换为 GPU ISA（如 PTX 中间表示再编译为实际 SASS 指令）才能在硬件上运行。同样地，国产 AI 芯片也有各自的指令体系：比如华为昇腾提供了 CANN 编译器和算子库，将模型算子翻译为昇腾 NPU 的底层指令；寒武纪 NeuWare 平台的代码生成器会将优化后的计算图编译为思元芯片可执行的二进制。业界观点指出，为增强自主可控，华为已将其硬件使能层 CANN 完全开源，提供底层算子和调度能力，让开发者能够更自主地优化指令级性能。因此，指令集支持和工具链完善程度极大影响了软件生态的易用性——成熟生态通常提供丰富的编译优化选项，使模型无需修改即可映射到芯片指令并高效运行。

综上，AI 芯片的硬件特性在并行度、存储体系、互连架构、数据精度等多个方面对软件生态提出了独特要求。首先，高并行计算单元意味着软件栈需要良好的并行性支持和任务调度机制，确保每个核心都有事可做。其次，复杂的存储层次要求编译器和运行时策略（如内存复用、预取）贴合芯片的缓存和显存特点，降低访存延迟。再次，高速互连使得分布式训练框架必须考虑通信开销、采用集合通信算法和拓扑感知调度以充分利用带宽。最后，混合精度与专用算力单元（如张量核心、

低精度计算）也需要软件提供相应支持，以便开发者方便地利用更低数值精度来换取性能提升。以主流加速器为例，NVIDIA H100（Hopper）单卡配备 80 GB HBM3，显存带宽超过 3 TB/s，并提供大容量 L2 以缓解访存瓶颈；这类带宽与层级缓存设计决定了编译器/运行时必须以内存复用、计算-通信重叠与预取/重排为优化主线，以避免被访存拖垮总体吞吐。总之，硬件的设计理念从底层约束和指引着软件生态的演进方向：只有软硬件协同优化，才能让 AI 芯片从“能用”进一步走向“好用”。正如 Ascend 昇腾架构采用分层解耦设计，使开发者可按需调用从模型到算子、再到底层内核的各级资源，其开源 Runtime 还支持对硬件资源的细粒度控制，以帮助开发者充分挖掘芯片性能[5]。这一软硬结合的模式，也是在后摩尔时代突破算力瓶颈、释放 AI 芯片潜能的关键。

湖南大学

参考文献

[1] Chen T, Moreau T, Jiang Z, et al. TVM: an automated end-to-end optimizing compiler for deep learning//Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18). Carlsbad, CA, USA: USENIX Association, 2018: 578-594.
[2] 宗建树, 杨洋, 余庚宗. 寒武纪(688256)深度报告：云程发轫，厚积薄发[EB/OL]. 长江证券研究所, 2024-09-01[2025-11-12]. https://pdf.dfcfw.com/pdf/H3_AP202409021639684152_1.pdf.
[3] 一起读年报中没说的隐坑 本期研究：寒武纪[EB/OL]. 东方财富网, 2025-09-01[2025-11-16]. 可获得于: https://emcreative.eastmoney.com/app_fortune/article/index.html?artCode=20250901165620486295280&postId=1593120168.
[4] 沐曦 MetaX 团队联合 HAMi 推出统一调度方案：实现 sGPU 共享、三档 QoS、拓扑智能调度与 WebUI 全面适配[EB/OL]. InfoQ, 2025-07-25[2025-11-12]. https://www.infoq.cn/article/CJSEIXxKzO9QYRoHShrT.
[5] 华为算力“公共事业”：“超节点+全栈开源”如何撬动 AI 未来？[EB/OL]. InfoQ, 2025-09-28[2025-11-12]. https://www.infoq.cn/article/gjh7qy46itf76nfiu9tx.
[6] NVIDIA Data Center GPU Manager (DCGM)[EB/OL]. NVIDIA Developer, [2025-11-12]. https://developer.nvidia.com/dcgm.
[7] 曦云系列通用计算 GPU mx-smi 使用手册[EB/OL]. MetaX Developer Center, [2025-11-12]. https://developer.metax-tech.com/api/client/document/preview/334/C500_mxsmiManual_CN.html.
[8] 配置及管理 ack-nvidia-device-plugin 组件[EB/OL]. 阿里云文档, 2025-09-16[2025-11-12]. https://help.aliyun.com/zh/ack/ack-managed-and-ack-dedicated/user-guide/gpu-device-plugin-related-operations.
[9] 使用 Prometheus 查看 Lite Cluster 监控指标[EB/OL]. 华为云文档, [2025-11-12]. https://support.huaweicloud.com/usermanual-cluster-modelarts/umn-cluster-modelarts-0026.html.
[10] Nebius team. Slurm Workload Manager: The go-to scheduler for HPC and AI workloads[EB/OL]. Nebius AI Cloud, 2025-08-01[2025-11-16]. Available at: https://nebius.com/blog/posts/slurm-workload-manager
[11] ANDERSCH M, PALMER G, KRASHINSKY R, et al. NVIDIA Hopper Architecture In-Depth[EB/OL]. NVIDIA Developer Blog, 2022-03-22[2025-11-12]. https://developer.nvidia.com/blog/nvidia-hopper-architecture-in-depth/.
[12] Introduction to the npu-smi Command for Versions 1.0.0-1.0.10[EB/OL]. Huawei Support, 2024-12-27[2025-11-12]. https://support.huawei.com/enterprise/en/doc/EDOC1100079295/7a356c41/introduction-to-the-npu-smi-command-for-versions-100-1010.
[13] LIN Y, GROVER V. Using CUDA Warp-Level Primitives[EB/OL]. NVIDIA Developer Blog, 2018-01-15[2025-11-12]. https://developer.nvidia.com/blog/using-cuda-warp-level-primitives/.