intel

软硬件双重驱动
英特尔为 AIGC
产业落地提供动能

目录 Contents

趋势篇

产品与技术篇

趋势篇

AIGC 正成为人工智能赋能千行百业的新动力

人工智能（Artificial Intelligence，AI）在当下科技领域无疑扮演着最重要的角色，它能借助深度学习（Deep Learning，DL）、机器学习（Machine Learning）等方法，通过像人类一样“思考”来实现知识的增长和性能的优化，从而成为金融、医疗、交通物流及智能制造等领域的效率倍增器。ChatGPT、Midjourney 以及 Stable Diffusion 等应用的到来，让人们认识到，以人工智能生成内容（AI-Generated Content，AIGC）为代表的 AI 2.0 正在强劲算力、先进算法和海量数据的加持下，加速完成从感知、理解世界到创造、生成世界的进击，在更多领域中成为内在使能者。

与以往的 AI 应用相比，AIGC 显然更符合人们对人工智能的最初遐想和期望。在自然语言处理（Natural Language Processing，NLP）、机器机视觉（Computer Vision，CV）以及多模态技术等方向上的 AI 技术 / 网络模型的帮助下，通过与人类或设备的交互，AIGC 应用能借助对知识的感知、学习和推理，更准确、更细微以及更深层次地理解和分析输入者的意图，并生成人类可以理解的文本、图像、音频以及视频等内容。例如 ChatGPT 能理解人类输入语句中隐含的幽默和比喻，并以适当的方式予以回应，这是传统的 AI 能力所难以实现的。

这些 AI 技术除了传统的机器学习 / 深度学习方法（如循环神经网络（Recurrent Neural Network，RNN）等），生成式对抗网络（Generative Adversarial Network，GAN）和大语言模型（Large Language Model，LLM）也是推动 AIGC 发展的重要网络模型。

其中，GAN 是通过两个神经网络（分别作为生成器和判别器）相互博弈的方法来进行学习，其以无监督学习的模式让生成器和判别器相互对抗、不断学习优化，最终使生成的内容更符合人类真实体验。

而 LLM 是使用大量数据训练的深度学习模型，其不仅具备更强的学习性能和更优的模型拟合效果，也拥有更强的通用型，能帮助用户完成不同类型的任务。目前流行的生成式预训练 Transformer 模型（Generative Pre-Trained Transformer，GPT）就是一种典型的预训练 LLM 模型。

AIGC 拟真的交互能力和高质量的内容创作能力，让传统的内容生成模式从专业生成内容（Professional Generated Content，PGC）、用户生成内容（User Generated Content，UGC）跨入 AI 生成模式，从而让 AI 能力在更多的行业和领域中成为生产力本身而仅非助力，进而演化出丰富的业务应用场景。

作为新型的内容生产方式，AIGC 对企业的赋能与价值目前已在商业营销、数字化办公、在线客服以及数字化服务等大量领域得到验证和认可。一些研究报告显示，有 33% 的企业在营销场景、31.9% 的企业在在线客服领域以及 27.1% 的企业在数字办公场景迫切希望得到 AIGC 能力的支持[1]。而对于 AIGC 带来的总体潜在经济效益，一些市场研究认为其每年或可高达 6.1 至 7.9 万亿美元[2]。

AIGC 需要全新软硬件驱动力来加速产业落地

AIGC 在更多行业和领域的推广与商用化落地也并非易事。基于 LLM 模型等构建的 AIGC 应用无疑在算力需求和数据规模上会带来更大挑战，尤其是 LLM 模型拥有的巨大参数规模（数百上千亿计）需要承载设备具备更强的算力，更大的内存带宽等，这使得开发者 / 创作者在设备或平台的选择上更具难度。传统上，这些体量庞大的网络模型让 AI 应用背后的模型总是离不开云服务、数据中心这样的部署环境，用户需通过联网接入才能开启开发 / 创作之旅。

在 AIGC 应用更多下沉和落地于各行业、各领域的今天，其发展也需要更多开发者 / 创作者共同参与其中。除了互联网这样的新兴产业，新闻媒体、艺术创作、广告设计以及电商客服等同样也是 AIGC 重要的应用方向。这就需要 AIGC 的载体，以及对应的开发 / 创作方案更为轻便、灵活和高效。例如旅行途中的设计师可能需要将偶然的灵感在 AIGC 应用的帮助下变成作品，进行中的商务会议可能需要 AIGC 应用实时协助生成数据报表或 PPT。

这些变化在带来挑战的同时也带来新的机遇：

一方面，是相关领域的厂商如何借助前沿的硬件 / 软件技术发展，满足 AIGC 应用开发与运行所需的高性能和便捷性。即如

何让 AIGC 开发 / 创作所需的软硬件能力浓缩到更为普及和轻薄的设备中，例如笔记本电脑，使之更贴合使用者的习惯。

另一方面，这一趋势也成为诸多厂商深度参与 AIGC 产业的重要切入点。无论是独立硬件供应商 (IHV)、独立软件开发商 (ISV)，还是原始设计制造商 / 原始设备制造商 (ODM/OEM) 或系统集成商 (SI) 都可以通过在硬件设计、软件优化、工具链、技术经验等方面的协作获得新的市场机遇。

作为行业领头羊的英特尔，不仅在 AI 领域有着全面的软硬件产品体系和技术栈，也在 AI 应用的行业落地和方案优化部署上有着丰富的经验和实战案例。

■ 在硬件方面

除了以英特尔® 至强® 可扩展处理器平台为代表的算力王牌外，英特尔也在推动 AIPC 这一全新产品形态的发展。

- 在硬件配置上，英特尔® AIPC 引入了全新的英特尔® 酷睿™ Ultra 处理器，其不仅采用了 Intel 4 制程，也是首款集成神经网络处理器 (Neural Processing Unit，NPU) 的处理器。同时，AIPC 中英特尔® 锐炫™ 系列独立显卡 (GPU) 的加入，也为 AIGC 所需的算力提供支撑。

- 而针对 CPU、GPU 和 NPU 的异构协作，英特尔也通过优化设计来实现性能加速、能耗降低并提升安全性，实现各种 AIGC 应用的离线稳态运行。

■ 在软件方面

软件能力也是 AIPC 体验的关键，而英特尔在这方面有着独特的优势地位。英特尔通过多个层面的软件能力助力 AIGC 产业的发展。

- 首先是通过软件来充分激发英特尔 ® 架构硬件蕴含的算力，让底层硬件性能更好地支撑上层的 AIGC 应用，例如 OpenVINO™ 工具套件全面支持英特尔异构的硬件平台，开发者可以灵活选用各种英特尔® CPU、GPU 和 NPU 等计算引擎，满足包括客户端、边缘计算的不同业务场景需求；同时提供包括面向低比特混合精度的量化在内的多种软硬件优化策略等。

- 其次是通过广泛地合作，对各类 AI 框架、平台和加速库提供面向英特尔® 架构的支撑和优化。例如 OpenVINO™ 工具套件广泛支持 PyTorch, TensorFlow, PaddlePaddle 等主流 AI 框架，并与 HuggingFace 合作，借助 Optimum Intel 帮助开发者将更多 HuggingFace 的大语言模型进行压缩优化，并在英特尔的平台上进行部署。又如 BigDL-LLM 加速库利用了低比特优化和硬件优化技术，可以极低的时延运行和微调大语言模型，并支持各类标准 PyTorch API (如 HuggingFace Transformers 和 LangChain) 和大模型工具 (如 HuggingFace PEFT、DeepSpeed、vLLM 等)。

这种软硬件协同的驱动力对 AIGC 应用开发和创作带来的助益是显而易见的。以 Stable Diffusion 的使用为例，借助英特尔为其开发的 AI 框架，开发者可在开启 OpenVINO™ 工具套件的情况下，仅通过一行代码的安装，就可以加速 PyTorch 模型运行，并让 Automatic1111 for Stable Diffusion WebUI 在英特尔® 锐炫™ 显卡上流畅运行，从而快速生成高质量图片。

同时，英特尔也与众多合作伙伴一起，通过开源社区、技术创新大会等形式，积极促进 AIGC 生态的发展、联动与创新，并催化更多 AIGC 的应用方案和落地实例。例如，英特尔计划在其推出的“AIPC 加速计划”中与超过 100 家 ISV 合作伙伴开展深度合作，并集合 300 余项 AI 加速功能，在 2025 年前为超过 1 亿台 PC 带来 AI 特性。这些合作和 AI 加速功能显然将进一步提升 AIGC 在各领域的产业落地。

本白皮书接下来将对英特尔在 AIGC 领域提供的部分产品与技术，包括 AIPC 产品、英特尔® 锐炫™ 系列独立显卡、BigDL-LLM 加速库以及 OpenVINO™ 工具套件等进行简要介绍。除文中述及的内容，英特尔在各类 AI 加速引擎、面向 AIGC 的可信执行环境构建等方面也进行了大量的探索。限于篇幅，读者可访问英特尔官网 (intel.cn) 参阅更多信息。

产品与技术篇

AIPC

推动产业和技术发展的最佳方法是，通过不断地迭代试错和优化调整来使其持续前行，AIGC 产业的发展也同样如此。但对于高强度算力的需求，令传统的 AI 方案往往需要高性能处理器、独立显卡甚至是专用的 AI 算力卡提供支持，因此方案中的模型通常会基于云端或数据中心部署。这不仅拔高了用户借助 AI 实现数字化转型的门槛，也与 AIGC 应用落地和赋能更多行业的目标背道而驰。

而英特尔与众多合作伙伴一起携手推动的 AIPC 产品，则通过软硬件架构的革新，将 LLM 模型等引入 PC 产品（个人台式机、笔记本电脑等），从而使 AIGC 应用的开发、创作与使用实现本地化，在开发便利性、数据安全性以及应用范围的扩展性方面获得突破，打开 AIGC 时代更大的想象空间。例如在 AIPC 本地运行的 LLM 模型，无需联网也能实现智能问答。这在网络安全等级较高的环境下，无疑有着重要的意义。

在推动 AIPC 落地的过程中，英特尔为 AIPC 提供了新一代英特尔® 酷睿™ Ultra 处理器（代号“Meteor Lake”），这一处理器通过创新的架构设计，内置了专门用于 AI 计算的 NPU 处理器。同时，英特尔也在 AIPC 中布局了英特尔® 锐炫™ 系列独立显卡，实现“CPU+NPU+GPU”的多轮驱动，为 AIPC 产品的落地与推广构筑了坚实的硬件基础。

英特尔® 酷睿™ Ultra 处理器

基于 Intel 4 制程工艺的英特尔® 酷睿™ Ultra 处理器，借助先进的 Foveros 3D 封装技术，实现了全新的分离式多模块架构设计并集成 NPU 处理器，同时全新的架构设计也实现了更出色的能耗比。新处理器引入的优势特性包括：

图 1 在第六届中国国际进口博览会上展示的英特尔® 酷睿™ Ultra 处理器[3]

- 采用了先进的 Foveros 3D 封装技术，这一封装技术所具备的高密度、高带宽、低功耗互连等特性，把采用多种制程工艺的模块组合成分离式模块架构，包括计算模块、片上系统（System on Chip，SoC）模块、输入 / 输出（Input/Output，I/O）模块和图形模块；
- 计算模块由基于 Intel 4 制程工艺的性能核（P-Core）及能效核（E-Core）组成，新的制程工艺使晶体管密度相比上一代（Intel 7）增加了约一倍，每瓦性能提高了约 20%[4]；
- SoC 模块集成了 NPU 处理器，同时兼容 OpenVINO™ 工具套件等软件提供的 API 接口，能以低功耗的方式持续为 AIGC 提供开发与运行所需的性能；
- 图形模块集成了与英特尔® 锐炫™ 独立显卡类似的图形架构，性能相比传统集成显卡有着显著提升，性能是上一代的 2 倍[5]。而 I/O 模块则集成了雷电技术 4（Thunderbolt™ 4）和 PCIe 5.0 接口，吞吐性能更为卓越。
- NPU 处理器采用了多引擎架构，由两个神经计算引擎组成，可共同处理同一负载也可各自处理不同负载。每个神经计算引擎都包括了推理管道和 SHAVE DSP 两部分。其中推理管道由乘积累加运算阵列、激活功能块和数据转换块组成，其能利用固定运算功能来处理常见的计算任务，实现优秀的能耗比。而 SHAVE DSP 是专门为 AI 设计的超长指令字 / 数字信号处理器，其内置的流式混合架构向量引擎（Streaming Hybrid Architecture Vector Engine，SHAVE）可与推理管道、直接内存访问（Direct Memory Access，DMA）引擎协同，提高 NPU 处理器的性能。

CPU+NPU+GPU 协同，英特尔为 AIGC 打造的“三管齐下”

在为 AIGC 提供助力的层面，英特尔在 AIPC 产品中制定的是 CPU+NPU+GPU 异构协同策略。三种算力芯片中，CPU 响应速度最好，GPU 算力最强，而 NPU 能耗比最优。针对 AIGC 应用开发和运行中不同工作负载的特性，AIPC 产品可以根据需求的不同，将任务灵活地分配到不同的算力芯片中去，以形成总体效能的最优化。

首先，得益于新一代制程工艺，英特尔® 酷睿™ Ultra 处理器的 CPU 部分（计算模块）具有快速响应的能力，适合那些需要快速决策，对时延要求较高的轻量级 AI 任务。而英特尔® 锐炫™ 系列独立显卡（或英特尔® 酷睿™ Ultra 处理器中的图形模块）则具有强劲的并行处理和高吞吐性能，非常擅长处理与媒体、 3D 应用程序和图形渲染有关的任务，并能同时运行处理大量 AI 任务（更多英特尔® 锐炫™ 系列显卡内容，请参阅本白皮书下一章节）。而作为专用低功耗 AI 引擎，英特尔推出的首款 NPU 处理器以其良好的能耗比，更适合用于维持长时间的 AI 运行。

三种算力芯片，CPU+NPU+GPU 的协同作业，可以借助英特尔® Thread Director（由英特尔面向混合架构提供的协调机制，能以纳秒级的间隔监视各核心和线程的运行情况并反馈）来完成。同时，OpenVINO™ 工具套件也为不同的算力芯片提供了 AI 引擎的调度能力。通过这种协作能力，本地化的 AIPC 能在不同的应用场景中提供不同的 AI 能力，并具有更好的响应速度、能耗和安全隐私性。

在“Intel Innovation 2023”现场，英特尔向用户展示了在基于英特尔® 酷睿™ Ultra 处理器的个人电脑上，通过 AIGC 模型生成了一段泰勒 · 斯威夫特曲风的歌曲，这说明 AIPC 已经能在 AIGC 领域为用户带来实实在在的生产力价值。

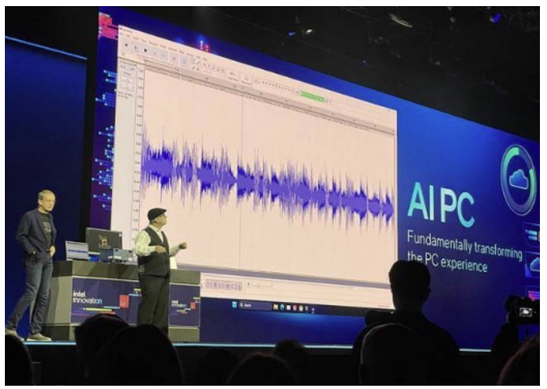

图 2 使用 AIPC 生成泰勒 · 斯威夫特曲风的歌曲[6]

英特尔® 锐炫™ 系列独立显卡

一直以来，GPU 产品都是 AI 应用开展训练和推理的重要算力引擎之一，因此在英特尔面向 AIGC 应用推出的产品体系中，除了有酷睿™ Ultra 处理器这样集成了 NPU 的先锋锐器，也包括了英特尔® 锐炫™ 系列独立显卡这一 GPU 领域的新秀。现在，英特尔 ® 锐炫™ 系列独立显卡产品正广泛地被运用于台式机、笔记本、工作站以及工业自动化检测系统等边缘环境中。除了在游戏体验、视频创作等领域给予使用者超凡的体验外，这一独立显卡产品也在 X^e-HPG 微架构、OpenVINO™ 工具套件等的加持下，在 AIGC 领域展现出强劲的实力。

图 3 英特尔® 锐炫™ 系列独立显卡[7]

作为英特尔在独立显卡领域推出的最新一代产品线，英特尔® 锐炫™ 系列独立显卡通过高度可扩展的图形引擎、光栅化、实时光线追踪、网格着色以及英特尔® X^e 超级采样（X^e Super Sampling，X^eSS）等技术和特性的加入，不仅帮助视频、图形和游戏领域的用户获得最大化的性能表现，也在 AIGC 领域助力开发者获得处理性能上的巨大收益。

全新 X^e-HPG 微架构带来性能大幅提升

英特尔® 锐炫™ 系列独立显卡的核心是英特尔最新推出的 X^e-HPG 微架构。与前代 X^e 微架构（X^e-LP）相比，新架构通过架构的更新，增强的基线、性能强劲的 AI 引擎、支持下一代编解码器标准的增强媒体引擎，以及在整数和浮点计算性能方面的巨大改善，成为英特尔® 锐炫™ 系列独立显卡强有力的技术支撑。

X^e-HPG 微架构由渲染切片（Rendering Slice）和 X^e 内核（X^e-Core）组成了灵活、可扩展的底层架构。其中每个 X^e 内核由 16 个矢量引擎（X^e Vector Engine，XVE）、16 个 X^e 矩阵扩展引擎（X^e Matrix Extension，XMX）、1 个加载 / 存储单元（或数据端口）以及 1 个共享的 L1/SLM 缓存组成，并连接到一组专用的图形

加速基线中，基线中包括一个射线跟踪单元 (Ray-Tracing Unit，RTU)、一个线程排序单元 (Thread-Sorting Unit，TSU) 以及一个采样器 (Sampler)。

如图 4 所示，每 4 个 X^e 内核组成 1 个渲染切片 (Rendering Slice)，渲染切片中还会内置几何处理 (Geometry)、像素后端 (Pixel Backend) 等功能组件。而连接每个渲染切片的 L2 高速缓存所组成的高带宽内存交换矩阵，能帮助 X^e-HPG 微架构灵活地构建多渲染切片配置，实现可扩展性。不同型号的英特尔® 锐炫™ 显卡拥有数量不等的渲染切片，最多可达 8 个 (例如英特尔® 锐炫™ A770 显卡和英特尔® 锐炫™ A770M 显卡)。

X^e 内核中，XVE 矢量引擎是执行指令的基本单元，类似于上一代 X^e-LP 架构中的执行单元 (EU)。它的主要计算单元是 256 位宽的单指令多数据 (Single Instruction Multiple Datastream，SIMD) 浮点单元，即算术逻辑单元 (Arithmetic Logic Units，ALU)，主要负责传统图像处理的计算。为有效提高执行性能和算力，经优化的 ALU 支持浮点运算指令，可与整数运算指令同时运行，并支持包括 DP4a 的快速 INT8 计算。

X^e 内核中新增了 16 个专用的高带宽 (1024 位宽)XMX 矩阵引擎，用于提升 AI 计算中最常见的矩阵乘法和累加计算性能。其使用了新的点积累加收缩 (Dot Product Accumulate Systolic，DPAS) 指令来执行矩阵乘法累加运算，并涵盖包括 BF16、INT8 等最常见的 AI 数据类型，实现了基于硬件的 AI 加速。同时，XMX 矩阵引擎也是 XeSS 技术的算力核心之一，其能在实时渲染过程中使用 AI，从而带来画质的大幅提升。

X^e 内核还内置了一个缓存单元，可根据 AI 或其它工作负载的需要在 L1 缓存和共享本地内存 (Shared Local Memory，SLM) 之间动态分配调整。

为 AIGC 提供强劲和便捷的算力基座

借助 N6 芯片工艺，英特尔® 锐炫™ 系列独立显卡获得了额外的 v-f 改进，可在同等电压下实现高达 1.5 倍的频率增长，而功耗则通过微架构优化和固有动态电容 (Dynamic Capacitance，Cdyn) 降低而降低。同时，其能通过更好的缓存和压缩技术来降低带宽利用率，将能耗重定向到 GPU，从而提高性能。这些举措让显卡的最高主频可达 2.1 GHz。

同时，显卡还拥有最多 32 个 X^e 内核 (内含 512 个 XVE 矢量引擎和 512 个 XMX 矩阵引擎)，与上一代 X^e-LP 架构产品相比，计算能力提高了 5 倍以上[8]。XVE 矢量引擎对 DP4a 提供了良好的支持。DP4a 是针对低精度 AI 计算所做的优化，其将 32 位输入分成 8 位块，然后独立地对这些块进行计算操作。而 XMX 矩阵引擎则通过对乘法累加的深度流水线化，其与 DP4a 一样，每个操作数都被分成 4 个块，这些块被独立地执行相乘和累加这些操作指令。数据块的变小带来了计算速度的大幅增加。

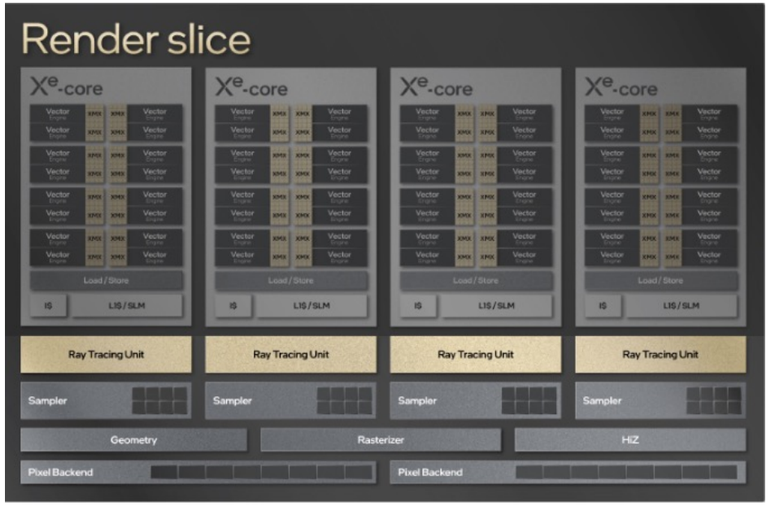

图 4 X^e-HPG 微架构渲染切片架构

因此，英特尔® 锐炫™ 系列独立显卡可为 AIGC 开发者提供：

- FP32 下高达 17.2 TFLOPS 的性能峰值；
- 使用 XMX 矩阵引擎时，FP16 下高达 137.6 TOPS 的性能峰值；
- 使用 XMX 矩阵引擎时，INT8 下高达 275.2 TOPS 的性能峰值。

针对 AIPC (个人台式电脑、笔记本电脑等) 这一类轻便型的 AIGC 开发设备，英特尔® 锐炫™ 系列显卡引入了统一无损压缩 (Unified Lossless Compression) 和对高带宽内存的支持。统一无损压缩能减少 AIGC 任务中数据读 / 写所需的带宽和功耗，这帮助 AIGC 开发者即便在笔记本内存有限的情况下，也可以应对 LLM 模型场景。而对高带宽内存的支持使 AIGC 的推理工作实现了更高的性能。显卡可支持 GDDR6 图形内存，每个引脚最多支持 17.5 GT/s；在 2x16 位配置中，最多支持八个通道 (最多 256 位)，由此，AIGC 任务可获得高达 560 GB/s 的低时延带宽，使笔记本等便携设备在 AIGC 开发中同样游刃有余。

图 5 面向笔记本电脑部署的英特尔® 锐炫™ 系列独立显卡

面向 AIGC 的性能优化还来自于 OpenVINO™ 工具套件、微软 WinML API 以及面向英特尔® 架构优化的深度神经网络计算库 (Compute Library for Deep Neural Networks，CLDNN) 的协同配合。OpenVINO™ 工具套件可以在开发者改动极少代码的前提下，在英特尔® 锐™ 系列显卡上顺畅运行。更多 OpenVINO™ 工具套件的使用和优化细节，请参阅下文 OpenVINO™ 工具套件部分。

CLDNN 库可以帮助英特尔® 锐炫™ 系列独立显卡借助英特尔® 深度学习部署工具包、一系列面向英特尔® 架构优化的深度学习框架以及英特尔® oneAPI 数学内核库 (英特尔® oneMKL) 等来实现深层次的优化，提升其在 AIGC 任务中的性能表现。

以英特尔® Deep Link 技术实现强强联手

英特尔® 锐炫™ 系列独立显卡在 AIGC 领域提供的性能输出，不仅可以来自显卡本身，也能在英特尔® Deep Link 技术的加持下，通过与酷睿™ Ultra 处理器等算力芯片的协同，帮助开发者在 AIPC 等设备上实现对 LLM 模型的有效驱动。

如前文所述，由英特尔推出的 AIPC 实现了“CPU+GPU“的双向算力驱动，借助 CPU 强大的计算性能和锐炫™ GPU 出色的渲染性能，为 AIPC 在 AIGC 领域的运用构筑了坚实的硬件基础。但这也面临了一个新的挑战，即如何让 CPU 与 GPU 等不同的算力硬件被自动调度并参与到 AIGC 工作负载中，这对于深挖算力硬件的潜能，提升 AIPC 等设备的性能尤为重要。

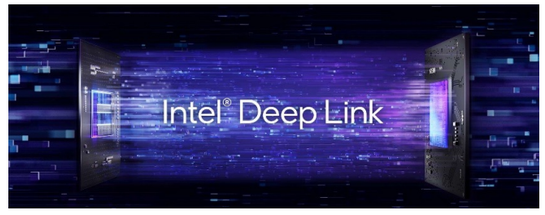

图 6 英特尔® Deep Link 技术

英特尔® Deep Link 技术是英特尔助力开发者应对以上挑战的有效方案。其通过动态功率共享 (Dynamic Power Share)、超级编码和超级计算等技术栈的引入，助力硬件设备实现性能和效率的全新提升。

- **动态功率共享技术**

这一技术能在设备功耗有限的情况下 (典型如笔记本电脑)，在 CPU 和 GPU 之间动态分配功率，更大地释放 CPU 或 GPU 的性能。借助智能算法管理的统一功耗包络，设备在运行 AIGC 等工作负载时，如果 CPU 更需要功率，系统会智能地将功率更多地分配给 CPU，反之对 GPU 也是一样，最终目的是让设备在功率一定时，实现更好的计算性能。一般地，动态功率共享技术能让创作和计算密集型应用程序的性能最高提升 30%[9]。

- **超级编码技术**

超级编码技术是允许用户同时运用两个显卡（例如带有 iGPU 的英特尔® 酷睿™ 处理器与英特尔® 锐炫™ 独立显卡协同工作）的编解码引擎，来大大提升编解码效率。这种协作是通过 OneVPL 的 API 接口来实现的。OneVPL 是一个跨平台的开放性框架，应用程序通过其 API 接口来识别并调用多个处理引擎，充分增强视频处理能力。

- **超级计算技术**

超级计算技术同样也能帮助 AIGC 任务从多个显卡的协同工作中收益，例如带有 iGPU 的英特尔® 酷睿™ 处理器与英特尔® 锐炫™ 独立显卡协同工作。这一技术使用 OpenVINO™ 工具套件提供的机器学习服务（Machine Learning Services，MLS）框架，这一框架能根据当前应用或负载的特征，例如时延敏感度、吞吐量、性能要求、功率消耗等，对算力和负载进行自动化的合理匹配。当 MLS 框架启动时，不同的计算处理任务将被分配到不同的工作线程，并根据对算力的需求分配到不同的设备上。

OpenVINO™ 工具套件

作为英特尔推出的用于加速 AI 推理及部署的软件工具，OpenVINO™ 工具套件能在异构平台上基于不同维度的 AI 方法，包括深度学习、基于注意力的网络以及 LLM 等，帮助 AI 领域的开发者在计算机视觉、自动语音识别、自然语言处理以及推荐系统等场景中，加速相关应用程序和解决方案的开发。OpenVINO™ 工具套件对大量流行 AI 框架（如 PyTorch、TensorFlow、PaddlePaddle 等）都提供了支持，同时也能借助硬件平台（如第四代英特尔® 至强® 可扩展处理器）的特性和内置加速器增加 AI 应用的功能和性能。随着最新版本 OpenVINO™ 工具套件 2023.2（2023 年 11 月发布）的推出，2023 系列发布的 3 个版本均为 AIGC 模型提供了广泛的支持，并持续提升大语言模型的优化和性能，使开发者在借助工具套件提升其作品的产出效率和质量时更为得心应手。

接下来，OpenVINO™工具套件会以更快的频率发布更多的功能和更好的性能，欢迎开发者访问并关注我们的主页，*https://www.intel.com/content/www/us/en/developer/tools/openvino-toolkit/overview.html* 获取最新的产品信息。

OpenVINO™ 工具套件优化思路

OpenVINO™ 工具套件主要包括模型优化器（Model Optimizer）和推理引擎（Inference Engine）等核心组件。其中模型优化器是一种跨平台命令行工具，可将在不同框架下训练后的网络模型转换为中间表示（即 Intermediate Representation，IR）文件，由 bin（经训练的数据文件）和 xml（描述网络拓扑的文件）两种格式文件组成）用于后续 AI 推理环节。这一过程中，模型优化器提供了两类优化方式，首先是针对复杂网络的模型结构压缩技术。OpenVINO™ 工具套件在读取模型后，将对其网络结构进行深度分析，基于预设策略，将一些常见的线性算子进行融合处理，从而压缩网络结构，节省数据在内存中的读写耗时。

另一类优化是面向低比特混合精度的量化与模型重训练策略。量化过程是将基于高精度数据格式的模型（由 IR 文件表示）转为较低精度，并加入校准过程以保证精度不受损失。一般 AI 框架中训练的模型通常为 FP32 精度数据格式，而英特尔® 架构处理器等算力平台目前已对 INT8 等低精度数据格式下的模型推理有了良好的支持，可帮助 AIGC 开发者在损失很小精度的前提下实现更高的推理效率。目前，OpenVINO™ 工具套件支持通过 Post-training 工具，基于精度感知的量化策略，进行混合精度量化。

推理引擎是一种基于 x86 内核指令集的加速引擎，其能通过通用 API 接口使 AIGC 应用在不同硬件平台上运行，并使用硬件指令集来加速 AI 任务的运行（例如其已能支持第四代英特尔® 至强® 可扩展处理器，并通过英特尔® AMX、英特尔® AVX-512 以及采用 VNNI 的英特尔® DL Boost 技术来提升推理性能）。在完成模型压缩等优化后，OpenVINO™ 工具套件可根据 AIGC 所使用的硬件平台，对计算图结构进行进一步优化，通过提升算子计算并行性等方式来提升整个网络结构的吞吐量。

同时，OpenVINO™ 工具套件也提供了针对多任务场景的跨平台异构加速技术。例如可通过异步执行策略，将 AIGC 的推理任务平均分配到不同的酷睿™ Ultra 处理器内核资源上，减少多线程间的任务同步，提供网络吞吐能力；同时能够根据设备负载情况，自动将推理任务以模型子图为单位，在酷睿™ Ultra 处理器和锐炫™ 系列显卡这样的异构算力设备上进行动态部署，全面激活英特尔平台的 iGPU 资源，提升资源使用率。

目前，工具套件所支持的硬件设备既包括英特尔® 至强® 可扩展处理器系列、集成 NPU 的新一代英特尔® 酷睿™ Ultra 处理器等，也包括英特尔® 锐炫™ 系列显卡产品。可以让 AIGC 开发者实现一次开发，即可面向不同平台部署并获得一致的性能表现，提升 AIGC 作品的开发与部署效率。

面向 AIGC 推出的 OpenVINO™ 工具套件新版本

AIGC 应用和模型的爆炸式增长，不仅在更多 AI 应用领域中带来了诸多令人耳目一新的 AIGC 模型，巨大的模型也使得计算复杂性得以成倍增加。而要加速 AIGC 作品的落地，对训练、推理流程和工具的优化和加速也必不可少。

最新版本 OpenVINO™ 工具套件（2023.2）的推出，正是英特尔为应对这一变化而提供的有力支撑。新版本通过数项增强功能的加入，对 AIGC 应用场景开展了针对性的优化，实现了许多关键功能。英特尔希望通过工具套件的协助，AIGC 开发者能在资源受限的环境（例如普通台式机和笔记本电脑）中本地运行计算任务。

OpenVINO™ 工具套件（2023.2）新版本提供的面向 AIGC 的部分增强功能包括：

- **面向 AIGC 的增强：**新版本提升了面向 AIGC 应用和技术方案（例如 LLM）的性能和可访问性，包括 runtime 时的性能以及对内存的优化。对于众多热门的 AIGC 模型，英特尔提供了优化和验证，包括但不限于 T5, Aquila/Aquila2, Vicuna, OPT, Pythia, StableLM, Blenderbot , RedPajama, XGen , LongChat , Jais , orca-mini, Mistral, Falcon, MPT, BLOOM, Dolly/ Dolly 2, LlaMA/LlaMA2, GPT-J, GPT-NeoX, Zephyr, Open-LlaMA, Whisper, Falcon, Stable-Diffusion-XL, Stable-diffusion-1.5, Stable-diffusion-2, DeepFloyd IF, Tiny-SD, Small-SD, LCM, Replit Code, CodeGen/ CodeGen 2, CodeT5, ChatGLM2/ChatGLM3, Yi, 百川 , 通义千问等；其他更多的模型也有社区提供了参考实现，如 LLaVA，DeciDiffusion, LCM LoRA，Würstchen，Distil-Whisper，AudioLDM2，FILM, MusicGen，InstructPix2Pix，Bark，书生等。

- **基于 GPU 平台的 LLM：**新版本使部署在英特尔® GPU 平台上的 LLM 获得显著改进，包括模型覆盖范围扩展至支持动态形状（dynamic shapes），进而帮助 AIGC 开发者提高在英特尔® 架构集成和独立显卡上运行 AIGC 工作负载时的性能。同时新版本也改进了动态形状所涉及的内存复用和权重内存消耗问题。

- **神经网络压缩框架（Neural Network Compression Framework，NNCF）：**新版本中的 NNCF 框架引入了 8 位权重压缩方法，可更轻松地压缩和优化 LLM 模型。同时新版本也加入了 SmoothQuant 功能，可更准确、高效地对 LLM 模型进行训练后的量化。

- **对 PyTorch 框架的增强：**新版本能够自动导入和转换 PyTorch 模型，并可与 OpenVINO™ 工具套件提供的 API 协同使用。AIGC 开发者可借助 PyTorch torch.compile 将 OpenVINO™ 工具套件作为后端，并通过 PyTorch API 来使用。此功能还集成到了 Automatic1111 for Stable Diffusion Web UI 中，提升在英特尔® CPU/GPU 平台上运行 Stable Diffusion 1.5 和 2.1 的性能。

- **提供面向英特尔的 Optimum：**Optimum 是 HuggingFace Transformers 库提供的一个扩展包，得益于 HuggingFace 和英特尔携手对流行的 AIGC 模型进行的持续增强，借助新版本 OpenVINO™ 工具套件来执行推理任务时能获得更快、更高效的运行效果。

- **对英特尔® 酷睿™ Ultra 处理器的支持：**新版本对基于英特尔® 酷睿™ Ultra 处理器开展的 AIGC 任务实现了良好的支持。

基于上述增强功能，OpenVINO™ 工具套件新版本可以面向 AIGC 创作提供多项整体性的优化方案，包括针对大型模型的整体堆栈优化，对 LLM 模型的权重量化以及直接 PyTorch 转换功能等。这些优化方案正在逐步覆盖不同的算力平台和操作系统。

- **大型模型的整体堆栈优化：**针对 AIGC 所使用的大型模型运行时对内存量、内存带宽的高需求，OpenVINO™ 工具套件针对英特尔® XPU 平台都进行了整体堆栈优化以满足推理时权重复制等操作对内存的高要求。这些优化包括对读取和编译模型所需的内存进行调优、对模型的输入 / 输出张量进行调优，以及调整其他内部结构用以缩短模型执行时间。

- **对 LLM 模型的权重量化：**LLM 模型在执行时需要大量的内存带宽，新版本 OpenVINO™ 工具套件在 NNCF 框架内实现了 INT8 LLM 权重量化功能，并用于基于英特尔® CPU 平台

的推理。NNCF 将生成并优化 IR 文件（与 FP16 精度的模型文件相比，文件体积小两倍）。IR 文件将在处理器插件中实现额外的优化，这将提高时延并减少运行时内存消耗。

- **直接 PyTorch 转换功能：**由于目前大多数 LLM 模型都基于 PyTorch 的环境构建，因此新版本提供了直接 PyTorch 转换功能，不仅转换时间大幅降低，对内存的需求也得到了有效压降。

与此同时，新版本 OpenVINO™ 工具套件所具备的转换 API 和权重量化功能也已集成到 Hugging Face optim-intel 扩展中，该扩展允许 AIGC 开发者使用 OpenVINO™ 工具套件作为推理堆栈运行模型，或以方便的方式将模型导出为 OpenVINO™ 格式。

OpenVINO ™ 工具套件加速 Stable Diffusion 实战

作为最为流行的 AIGC 应用，Stable Diffusion 受到大量开发者的青睐，但其配置和使用仍然面临挑战，例如复杂的软件安装和环境配置步骤会带来额外的门槛。同时，AIGC 开发者也希望获得更为有效的硬件加速方法来提升 Stable Diffusion 的运行效率。

来自 OpenVINO™ 工具套件的 Optimum-Intel 能以最少的代码行和依赖项安装，在 Stable Diffusion V2.1 上实现 AI 硬件加速的最快方法。首先通过以下执行参考命令安装 Optimum-Intel：

```
1. pip install -q "optimum-intel[openvino,diffusers]"@git+https://
   github.com/huggingface/optimum-intel.git
```

下载预转换过的 Stable Diffusion 2.1 IR 模型，所使用的基础模型是 stabilityai/stable-diffusion-2-1-base，该模型被转换为 OpenVINO™ 格式并使用 Optimum-Intel 在英特尔® CPU/GPU 平台上开展推理加速。模型权重以 FP16 精度进行存储，这可以有效减少模型的体积。参考代码如下：

```
1. from optimum.intel.openvino import OVStableDiffusionPipeline
2.
3. # download the pre-converted SD v2.1 model from Helena's HF
   repo
4. name = "helenai/stabilityai-stable-diffusion-2-1-base-ov"
5.
6. pipe = OVStableDiffusionPipeline.from_pretrained(name,
   compile=False)
7. pipe.reshape(batch_size=1, height=512, width=512, num_
   images_per_prompt=1)
```

下载预训练和转换为的 IR 格式 Stable Diffusion 模型后，保存预先训练好的模型，选择推理设备并编译，参考代码如下：

```
1. pipe.save_pretrained("./openvino_ir")
2. pipe.to("GPU.1")
3. pipe.compile()
```

此时，AIGC 开发者就可以输入创意提示，测试 Stable Diffusion 模型所生成的图像。参考代码如下：

```
1. # Generate an image.
2. prompt = "red car in snowy forest, epic vista, beautiful landscape,
   4k, 8k"
3. output = pipe(prompt, num_inference_steps=17, output_
   type="pil").images[0]
4. output.save("image.png")
5. output
```

OpenVINO ™ 工具套件优化 LLM 模型实战

以 OpenVINO™ 工具套件对 ChatGLM6b 大模型的优化为例，作为一种非量化优化方案，OpenVINO™ 工具套件与第四代英特尔® 至强® 可扩展平台协同，旨在降低模型对内存的需求，提升推理性能。

在优化开始前，需要先编译 OpenVINO™ 工具套件源码，安装系统依赖并设置环境，创建并启用 Python 虚拟环境，执行参考命令如下：

```
1. $ conda create -n ov_py310 python=3.10 -y
2. $ conda activate ov_py310
```

接下来安装 Python 依赖、使用 GCC 11.3.0 编译 OpenVINO™ 工具套件，克隆 OpenVINO™ 工具套件并升级子模块，执行参考命令如下：

```
1. $ pip install protobuf transformers==4.30.2 cpm_kernels
   torch>=2.0 sentencepiece pandas
2. $ git clone https://github.com/luo-cheng2021/openvino.git -b
   luocheng/chatglm_custom
3. $ cd openvino && git submodule update --init --
   recursive
```

安装 Python 依赖用于编译 Python Wheel，并创建编译目录，执行参考命令如下：

```
1. $ python -m pip install -U pip
2. $ python -m pip install -r ./src/bindings/python/src/compatibility/
   openvino/requirements-dev.txt
3. $ python -m pip install -r ./src/bindings/python/wheel/
   requirements-dev.txt
4. $ mkdir build && cd build
```

使用 CMake 编译 OpenVINO™ 工具套件，执行参考命令如下：

```
1. $ cmake .. -DENABLE_LLMDNN=ON \
2. -DBUILD_PYTHON_TESTS=ON \
3. -DENABLE_CPU_DEBUG_CAPS=OFF \
4. -DENABLE_DEBUG_CAPS=OFF \
5. -DCMAKE_BUILD_TYPE=Release \
6. -DENABLE_INTEL_MYRIAD_COMMON=OFF \
7. -DENABLE_INTEL_GNA=OFF \
8. -DENABLE_OPENCV=OFF \
9. -DENABLE_CPPLINT=ON \
10. -DENABLE_CPPLINT_REPORT=OFF \
11. -DENABLE_NCC_STYLE=OFF \
12. -DENABLE_TESTS=ON \
13. -DENABLE_OV_CORE_UNIT_TESTS=OFF \
14. -DENABLE_INTEL_CPU=ON \
15. -DENABLE_INTEL_GPU=OFF \
16. -DENABLE_AUTO=OFF \
17. -DENABLE_AUTO_BATCH=OFF \
18. -DENABLE_MULTI=OFF \
19. -DENABLE_HETERO=OFF \
20. -DENABLE_INTEL_GNA=OFF \
21. -DENABLE_PROFILING_ITT=ON\
22. -DENABLE_SAMPLES=ON \
23. -DENABLE_PYTHON=ON \
24. -DENABLE_TEMPLATE=OFF \
25. -DENABLE_OV_ONNX_FRONTEND=OFF \
26. -DENABLE_OV_PADDLE_FRONTEND=OFF \
27. -DENABLE_OV_PYTORCH_FRONTEND=OFF \
28. -DENABLE_OV_TF_FRONTEND=OFF \
29. -DENABLE_OPENVINO_DEBUG=OFF \
30. -DENABLE_CPU_DEBUG_CAPS=ON \
31. -DCMAKE_INSTALL_PREFIX=`pwd` /install \
32. -DCMAKE_INSTALL_RPATH="`pwd` /install/runtime/3rdparty/
    tbb/lib:`pwd` /install/runtime/3rdparty/hddl/lib:`pwd` /install/
    runtime/lib/intel64 \
33. -Dgflags_Dir=`pwd` /../thirdparty/gflags/gflags/
    cmake
34. $ make --jobs=$(nproc --all)
35. $ make install
```

此外，需要安装为 OpenVINO™ Runtime 和 openvino-dev 工具完成编译的 Python Wheel，并将 PyTorch 模型转为 OpenVINO™ IR 模型，执行参考命令如下：

```
1. $ pip install ./install/tools/openvino*.whl
2. $ cd ..
3. $ python tools/gpt/gen_chatglm.py /path/to/pytorch/model /path/
   to/ov/IR
```

在完成上述步骤后，开始对大模型进行分析并展开相关的优化。ChatGLM6b 大模型的结构如图 7 所示，其流水线回路主要包含 3 个主要模块，即 Embedding、GLMBlock 层和 lm_logits。模型的流水线中有两类不同的执行图，首次推理时不需要 KV 缓存作为 GLMBlock 层的输入。而从第二次迭代开始，QKV 注意力机制的上一次结果（pastKV）将成为当前一轮模型推理的输入。

随着所生成 tokens 长度不断增加，在流水线推理过程中，模型中输入和输出之间将存在大量的内存副本（内存拷贝开销由模型的参数 hidden_size 以及迭代的次数决定），不仅将占据巨大的内存空间，大量的内存拷贝开销也会使推理执行效率倍受挑战。

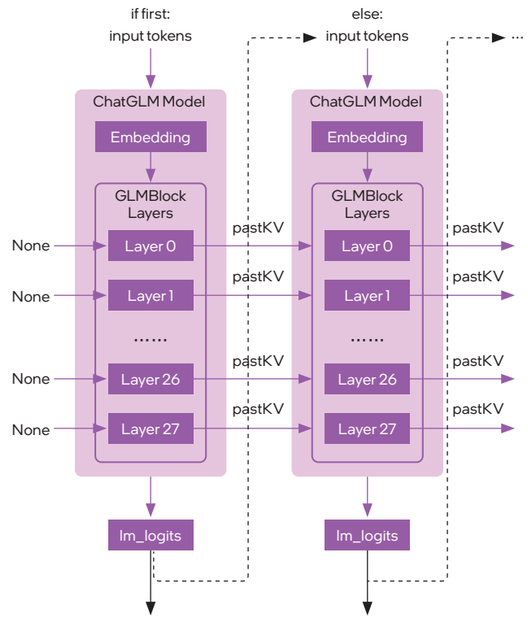

图 7 ChatGLM6b 的模型结构

为应对上述挑战，基于 OpenVINO™ 工具套件的非量化优化方案执行了三个方面的优化动作。

- **优化一：** 利用零拷贝 (Zero-Copy) 视图来传递预分配的 KV 所需的内存副本空间。由于传统的内存拷贝需要耗费大量的处理器资源和内存带宽，因此当内存副本规模大幅增加时，会成为大模型推理效率的瓶颈。而零拷贝技术的引入，能避免数据的多次拷贝，有效实现 KV 缓存加速。

- **优化二：** 使用 OpenVINO™ opset 来重构 ChatGLM 的模型架构，这能够帮助模型中的节点利用英特尔® AMX 指令集内联和多头注意力 (Multi-Head Attention，MHA) 融合来实现推理优化。如图 8 所示，优化方案构建的 OpenVINO™ stateful 模型在 GLMBlock 层重新封装了一个类，并按图中工作流来调用 OpenVINO™ opset，并通过其将图形数据序列化为 IR 模型。

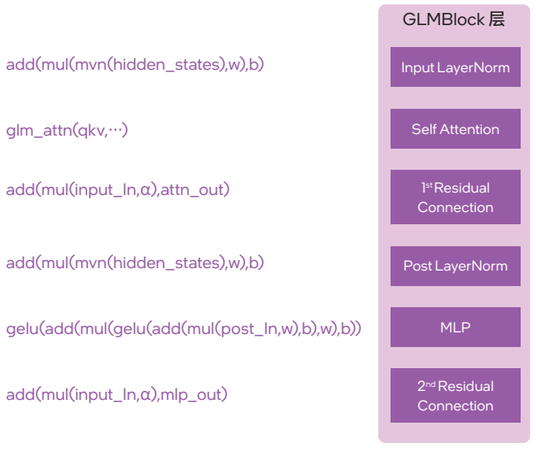

图 8 构建 OpenVINO™ stateful 模型

优化方案一方面构建了全局的上下文结构体，用于在模型内部追加并保存每一轮迭代后的 pastKV 结果，减少相应的内存拷贝开销 另一方面，方案也通过内联优化 (Intrinsic Optimization) 的方式，实现了 Rotary Embedding 和 MHA 融合。第四代英特尔® 至强® 可扩展处理器内置英特尔® AMX 指令集，也能帮助 ChatGLM 大模型提升 BF16 或 INT8 精度数据格式下的模型推理速度。英特尔® AMX 指令集提供的内联指令能快速处理 BF16 或 INT8 精度数据格式的矩阵乘加运算，实现对 ChatGLM 模型中 Attention，Rotary Embedding 等算子的融合，在保证精度的同时提高运算效率，加速推理速度。

- **优化三：** 第三项优化是引入了 OpenVINO™ 工具套件在 HuggingFace 上的 Optimum 接口。Optimum 是 Huggingfac Transformers 库提供的一个扩展包，可用来提升模型在特定硬件基础设施上的训练和推理性能。基于 OpenVINO™ 工具套件提供的 Optimum 接口，开发者能将模型扩展到更多 AIGC 应用中去。相关的优化仅需改动少许代码即可完成，代码修改示例如下 (面向文本分类 (text-classification) 任务):

```
1.  #替换import内容
2.  #from transformers import
    AutoModelForSequenceClassificatdon
3.  from optimum.intel import
    OVModelForSequenceClassificatdon
4.  from transformers impoxt AutoTokenizer, pipeline
5.
6.  #替换模型内容
7.  model_id ="distilbert-base-uncased-finetuned-sst-2-english"
8.  #model = AutoModelForSequenceClassificatdon.from_
    pretrained(model_id)
9.  model = OVModelForSequenceClassificatdon.
    frompretradned(model_id, export=True)
10. tokenizer = AutoTokenizer.from_pretrained(model_id)
11. cls_pipe = pipeline("text-classification", model=model,
    tokenizer=tokenizer)
12. outputs = cls_pipe("He's a dreadful magician. ")
```

在其它任务中，包括 token-classification、question-answering、audio- classification 以及 image- classification 等也同样适用。

同时，开发者还可使用 OpenVINO™ Runtime API 来为 ChatGLM 编译推理创建流水线。在 test_chatglm.py 中，创建继承 transformers.PreTrainedModel 的新类，并通过使用 OpenVINO ™ Runtime Python API 编译模型的推理流水线来更新转发函数。其他成员函数则迁移自 modeling_chatglm.py 的 ChatGLMForConditionalGeneration 中，这可以确保输入准备工作，即 set_random_seed / 分词器 / 连接器以及余下的流水线操作能够与原始模型的源码保持一致。

在模型生成阶段，开发者可借助 OpenVINO™ 工具套件插入用于量化和去量化 (QDQ) 的标签，其可在运行时推理过程中进行量化和去量化。因此在开发中如需使用 INT8 模型，只需设置简单的环境变量 USE_INT8_WEIGHT=1 即可。最后，开发者可按照以下步骤，使用 OpenVINO™ Runtime 流水线测试 ChatGLM，执行参考命令如下:

```
1.  #运行 bf16 模型
2.  $ python3  tools/gpt/test_chatglm.py /path/to/pytorch/model /
    path/to/ov/IR --use=ov
3.  #运行 int8 模型
4.  $ USE_INT8_WEIGHT=1 python test_chatglm.py /path/to/
    pytorch/model /path/to/ov/IR --use=ov
```

BigDL-LLM 加速库

BigDL-LLM 是基于英特尔® XPU(如 CPU、GPU) 平台的开源大模型加速库；它使用低比特优化 (如 FP4/INT4/NF4/FP8/INT8) 及多种英特尔 CPU/GPU 集成的硬件加速技术，以极低的时延运行和微调大语言模型。

BigDL-LLM 支持标准的 PyTorch API (如 HuggingFace Transformers 和 LangChain) 和大模型工具 (如 HuggingFace PEFT、DeepSpeed、vLLM 等)，可助力 AI 开发者和研究者在英特尔平台 (笔记本、工作站、服务器和 GPU) 上高效开发、加速大语言模型算法和应用。

使用 BigDL-LLM 非常简单；只需更改一行代码，您就可以立即观察到显著的加速效果。大量模型 (如 LLaMA / LLaM2、ChatGLM2/ChatGLM3、Mistral、Falcon、MPT、LLaVA、StarCoder、Whisper、百川 / 百川 2、通义千问 / 通义千问 VL、书生、悟道天鹰、MOSS 等) 已在 BgDL-LLM 上得到验证和优化。

便捷的 BigDL-LLM 使用方法

BigDL-LLM 的安装与使用非常简便，推荐使用 Conda 等 Python 环境管理工具来创建 Python 环境并安装必要的库，推荐使用 Python 3.9 来运行 BigDL-LLM。创建 Python 3.9 环境并激活 llm-tutorial 环境，执行参考命令如下 (使用 Conda):

```
1.  conda create -n llm-tutorial python=3.9
2.  conda activate llm-tutorial
```

仅需一行命令即可安装最新版本的 bigdl-llm 以及所有常见 LLM 应用程序开发所需的依赖项。执行参考命令如下:

```
1.  pip install --pre --upgrade bigdl-llm[all]
```

安装并启动 Jupyter 服务。值得注意的是，在服务器上建议使用单个插槽的所有物理内核以获得更好的性能。执行参考命令如下:

```
1.  pip install jupyter
2.
3.  #个人电脑
4.  jupyter notebook
5.
6.  # 以每个插槽有48个核心的服务器为例
7.  export OMP_NUM_THREADS=48
8.  numactl -C 0-47 -m 0 jupyter notebook
```

BigDL-LLM 为用户提供了两种使用方法：便捷命令 (Command Line Interface，CLI) 方法和编程接口 (Application Programming Interface，API) 方法。通过 CLI 方法，用户能方便地完成模型量化并评估量化后的推理效果，由此判断该量化方案是否适用于当前这个模型。这些 CLI 命令包括使用 llm-convert 来对模型的量化精度快速转换用于预览，或者使用 llm-cli/llm-chat 来运行并快速测试量化后的模型。

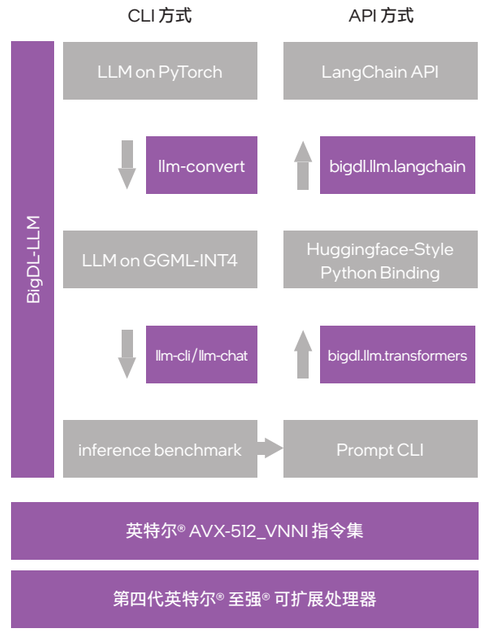

图 9 BigDL-LLM 为大模型提供推理加速

BigDL-LLM 有一个重要优点，即对于基于 HuggingFace Transformer API 的模型，仅需改动一行代码就可进行加速，而且理论上可以支持运行任何的 Transformers 模型。这对于熟悉 Transformer API 的 AIGC 开发者非常友好。作为热门的 Transformers 开源库之一，HuggingFace 上的 Transformers 模型一直是各类大模型的重要组成部分。同时，BigDL-LLM 也提供了方便使用的 LangChain 的集成。因此，AIGC 开发者选择 BigDL-LLM，并借助其提供的面向 HuggingFace 和 LangChain 的 API 编程接口，能够快速地将 LLM 量化方案整合进 HuggingFace 或 LangChain 的项目代码，便捷地完成模型部署。

例如在 HuggingFace 的实战中，开发者仅需更改 import，并在 from_pretrained 参数中设置 load_in_4bit=True 即可，在使用 bigdl.llm.transformers 后 BigDL-LLM 会在加载模型的过程中对模型进行 INT4 的低精度量化，由此实现对基于 HuggingFace Transformers 的模型进行加速。参考代码如下：

```
#基于INT4优化方案导入(load) Hugging Face Transformers 模型
from bigdl.llm.transformers import AutoModelForCausalLM
model = AutoModelForCausalLM.from_pretrained('/path/to/model/', load_in_4bit=True)

#在英特尔架构处理器上运行优化后的模型
from transformers import AutoTokenizer
tokenizer = AutoTokenizer.from_pretrained(model_path)
input_ids = tokenizer.encode(input_str, ...)
output_ids = model.generate(input_ids, ...)
output = tokenizer.batch_decode(output_ids)
```

而在 LangChain 的实战中，BigDL-LLM 同样也通过 API 编程接口 bigdl.llm.langchain 提供了便于使用的 LangChain 集成能力，让开发者能轻松借助 BigDL-LLM 来从头开发或迁移基于 HuggingFace Transformers 优化的 INT4 模型或其它原生 INT4 模型。以下是一个使用 LangChain API 运行 HuggingFace Transformers 模型（已进行 INT4 量化优化）的代码示例：

```
from bigdl.llm.langchain.llms import TransformersLLM
from bigdl.llm.langchain.embeddings import TransformersEmbeddings
from langchain.chains.question_answering import load_qa_chain

embeddings = TransformersEmbeddings.from_model_id(model_id=model_path)
bigdl_llm = TransformersLLM.from_model_id(model_id=model_path, ...)

doc_chain = load_qa_chain(bigdl_llm, ...)
output = doc = chain.run(...)
```

使用 BigDL-LLM 实现语音识别应用实战

Whisper 是一个流行的开源语音识别应用，其底层是基于 Trans- former 的编码器 - 解码器架构。下文将简述如何使用 BigDL-LLM INT4 优化功能对其进行有效优化。

在准备阶段，在设置 BigDL-LLM 环境之后，开发者首先可以安装 bigdl-llm 以及用于音频分析的 librosa 软件包，并下载测试使用的音频文件。在控制台执行参考命令如下：

```
!pip install bigdl-llm[all]
!pip install -U librosa
!wget -O audio_en.mp3 https://datasets-server.huggingface.co/assets/common_voice/--/en/train/5/audio/audio.mp3
!wget -O audio_zh.mp3 https://datasets-server.huggingface.co/assets/common_voice/--/zh-CN/train/2/audio/audio.mp3
```

完成准备工作后，首先加载预训练好的 Whisper 模型。这里需要加载一个经过预训练的 Whisper 模型，例如 whisper-medium。开发者只需在 bigdl-llm 中使用单行 transformers-style API，即可加载具有 INT4 优化功能的 whisper-medium（通过指定 load_in_4bit=True），参考代码如下：

```
from bigdl.llm.transformers import AutoModelForSpeechSeq2Seq

model = AutoModelForSpeechSeq2Seq.from_pretrained(pretrained_model_name_or_path="openai/whisper-medium", load_in_4bit=True)
```

无论是音频预处理还是将模型输出从标记转换为文本的后处理，都需要使用 WhisperProcessor。这里使用官方的 transformers API 加载 WhisperProcessor 即可。参考代码如下：

```
from transformers import WhisperProcessor

processor = WhisperProcessor.from_pretrained(pretrained_model_name_or_path="openai/whisper-medium")
```


加载 WhisperProcessor 后，就可以开始通过模型推理转录音频。以转录英文音频为例，从英语音频文件 audio_en.mp3 开始。在将其输入 WhisperProcessor 之前，需要从原始语音波形中提取序列数据：参考代码如下：

```
import librosa

data_en, sample_rate_en = librosa.load("audio_en.mp3", sr=16000)
```

然后根据序列数据转录音频文件，方法与使用官方的 transformers API 完全相同。其中的 forced_decoder_ids 将为不同语言和任务（转录或翻译）定义上下文 token 。如果设置为 None，Whisper 将自动预测它们。参考代码如下：

```
import torch
import time

# 定义任务类型
forced_decoder_ids = processor.get_decoder_prompt_ids(language="english", task="transcribe")

with torch.inference_mode():
# # 为 Whisper 模型提取输入特征
input_features = processor(data_en, sampling_rate=sample_rate_en, return_tensors="pt").input_features

# 为转录预测 token id
st = time.time()
predicted_ids = model.generate(input_features, forced_decoder_ids=forced_decoder_ids)
end = time.time()

# 将 token id 解码为文本
transcribe_str = processor.batch_decode(predicted_ids, skip_special_tokens=True)

print(f'Inference time: {end-st} s')
print('-'*20, 'English Transcription', '-'*20)
print(transcribe_str)
```

Whisper 可以转录多语言音频，并将其翻译成英文。这里唯一的区别是通过 forced_decoder_ids 来定义特定的上下文 token。现在以转录中文音频并翻译成英文为例，参考代码如下：

```
# 提取序列数据
data_zh,sample_rate_zh=librosa.load("audio_zh.mp3",sr=16000)

# 定义中文转录任务
forced_decoder_ids = processor.get_decoder_prompt_ids(language="chinese", task="transcribe")

with torch.inference_mode():
input_features = processor(data_zh, sampling_rate=sample_rate_zh, return_tensors="pt").input_features
st = time.time()
predicted_ids = model.generate(input_features, forced_decoder_ids=forced_decoder_ids)
end = time.time()
transcribe_str = processor.batch_decode(predicted_ids, skip_special_tokens=True)

print(f'Inference time: {end-st} s')
print('-'*20, 'Chinese Transcription', '-'*20)
print(transcribe_str)

# 定义中文转录以及翻译任务
forced_decoder_ids = processor.get_decoder_prompt_ids(language="chinese", task="translate")

with torch.inference_mode():
input_features = processor(data_zh, sampling_rate=sample_rate_zh, return_tensors="pt").input_features
st = time.time()
predicted_ids = model.generate(input_features, forced_decoder_ids=forced_decoder_ids)
end = time.time()
translate_str = processor.batch_decode(predicted_ids, skip_special_tokens=True)

print(f'Inference time: {end-st} s')
print('-'*20, 'Chinese to English Translation', '-'*20)
print(translate_str)
```

BigDL-LLM 与 LangChain 集成实战

作为一个流行的、使用 LLM 模型驱动应用程序的开源框架，LangChain 能让 AIGC 开发者方便地基于 LLM 模型去构建各类应用程序，包括聊天机器人、文档问答以及语音助手等。在 BigDL-LLM 中，也为开发者提供了面向 LangChain 的集成（包括 LLM wrapper 和 embedding），AIGC 开发者能像使用 LangChain 中其它 LLM wrapper 一样便捷地进行使用。

在准备阶段，在设置环境之后（同上文一致），开发者首先安装 bigdl-llm 与 langchain，在控制台执行参考命令如下：

```
!pip install bigdl-llm[all]
!pip install -U langchain==0.0.248
```

BigDL-LLM 为开发者提供了 TransformersLLM 和 TransformersPipelineLLM，用于实现 LangChain 的 LLM wrapper 的标准接口。这里，开发者可以使用 TransformerLLM.from_model_id，从 huggingface model_id 或路径实例化 TransformerLLM。并可以在 model_kwargs 中以字典形式传入与模型生成相关的参数（如 temperature, max_length）。TransformersPipelineLLM 的实例化方式与 TransformersLLM 类似，也是通过 huggingface model_id 或路径、model_kwargs 以及 pipeline_kwargs 来实现以下以 vicuna-7b-v1.5 模型为例，实例化创建一个 TransformerLLM。参考代码如下：

```
from bigdl.llm.langchain.llms import TransformersLLM

llm = TransformersLLM.from_model_id(
model_id="lmsys/vicuna-7b-v1.5",
model_kwargs={"temperature": 0, "max_length": 1024, "trust_remote_code": True},
)
```

使用 prompt 模板来格式化 prompt，然后调用 LLM 来测试生成的结果，或者在 LLM 上使用 generate 来获取多组结果。参考代码如下：

```
#prompt mode
prompt = "What is AI?"
VICUNA_PROMPT_TEMPLATE = "USER: {prompt}\nASSISTANT:"
result = llm(prompt=VICUNA_PROMPT_TEMPLATE.format(prompt=prompt), max_new_tokens=128)

#llm generate mode
llm_result = llm.generate([VICUNA_PROMPT_TEMPLATE.format(prompt="Tell me a joke"), VICUNA_PROMPT_TEMPLATE.format(prompt="Tell me a poem")]*3)
print("-"*20+"number of generations"+"-"*20)
print(len(llm_result.generations))
print("-"*20+"the first generation"+"-"*20)
print(llm_result.generations[0][0].text)
```

完成上述准备步骤后，开发者可以开始使用 Chains 中的 LLM wrapper 和 embedding。例如首先使用一个基础的 chain LLMChain。这需要创建一个简单的 prompt 模板，并使用 llm 与 prompt 模板来实例化一个 LLMChain。参考代码如下：

```
from langchain import PromptTemplate
from langchain import LLMChain

template ="USER: {question}\nASSISTANT:"
prompt = PromptTemplate(template=template, input_variables=["question"])
llm_chain = LLMChain(prompt=prompt, llm=llm)
```

此时，就可以向 LLM 问一个问题，并在 LLMChain 上调用 run 来获取响应。执行参考命令如下：

```
question = "What is AI?"
result = llm_chain.run(question)
```

如果要构建一个更为复杂的聊天应用程序，那么可以使用更复杂的 chain 和内存缓冲区来记忆聊天记录。参考代码如下：

```
from langchain import PromptTemplate
from langchain.chains import ConversationChain
from langchain.chains.conversation.memory import ConversationBufferMemory

template = "The following is a friendly conversation between a human and an AI.\
\nCurrent conversation:\n{history}\nHuman: {input}\nAI Asistant:"
prompt = PromptTemplate(template=template, input_variables=["history", "input"])
conversation_chain = ConversationChain(
verbose=True,
prompt=prompt,
llm=llm,
memory=ConversationBufferMemory(),
llm_kwargs={"max_new_tokens": 256},
)
query ="Good morning AI!"
result = conversation_chain.run(query)
```

下面的实例将尝试使用 LLM 来理解文本，这是构建问答系统领域的 AIGC 应用的重要步骤。首先需要安装必要的依赖库并加载文档，并拆分输入文件的文本。参考执行命令以及代码如下：


```
!pip install -U faiss-cpu

from langchain.text_splitter import CharacterTextSplitter

input_doc = "文档路径"
text_splitter = CharacterTextSplitter(chunk_size=650, chunk_overlap=0)
texts = text_splitter.split_text(input_doc)
```

完成文档拆分后，需要存储拆分内容，以便日后根据输入查询进行搜索。最常见的方法是嵌入每个分片的内容，然后将嵌入向量存储在向量存储中。BigDL-LLM 提供了 TransformersEmbeddings，允许使用 LLM 从文本输入中获取嵌入"TransformersEmbeddings"的实例化方法与上文的"TransformersLLM"类似。嵌入创建后，可以进一步存储到向量存储中。向量存储负责存储嵌入数据并执行向量搜索（下文使用 Faiss 库为例），参考代码如下：

```
from bigdl.llm.langchain.embeddings import TransformersEmbeddings
from langchain.vectorstores import FAISS

embeddings = TransformersEmbeddings.from_model_id(model_id="lmsys/vicuna-7b-v1.5")
docsearch = FAISS.from_texts(texts, embeddings, metadatas=[{"source": str(i)} for i in range(len(texts))]).as_retriever()
```

最后可以准备 chain 并生成结果，参考代码如下：

```
from langchain.chains.chat_vector_db.prompts import QA_PROMPT
from langchain.chains.question_answering import load_qa_chain

doc_chain = load_qa_chain(
llm, chain_type="stuff", prompt=QA_PROMPT
)

result = doc_chain.run(input_documents=docs, question=query)
```

[1]数据援引自公开媒体报道的TE智库报告《企业AIGC商业落地应用研究报告》：https://baijiahao.baidu.com/s?id=1768733350817647046

[2]数据援引自McKinsey Digital 报告《The economic potential of generative AI: The next productivity frontier》：https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/the-economic-potential-of-generative-ai-the-next-productivity-frontier#introduction

[3]图片援引自公开媒体报道：https://baijiahao.baidu.com/s?id=1781851944676091047

[4, 5]数据援引自公开媒体报道：https://baijiahao.baidu.com/s?id=1777510511062127430

[6]图片援引自公开媒体报道：https://news.sohu.com/a/721947787_120159035

[7]图片援引自英特尔官网：https://www.intel.cn/content/www/cn/zh/products/details/discrete-gpus/arc.html

[8]Xᵉ-LP 架构 Xᵉ–HPG架构不同产品性能比对，详情请参阅英特尔官网：https://intel.com

[9]相关技术细节与性能描述，请访问英特尔官网：https://www.intel.cn/content/www/cn/zh/architecture-and-technology/adaptix/deep-link.html

英特尔技术特性和优势取决于系统配置，并可能需要支持的硬件、软件或服务得以激活。产品性能会基于系统配置有所变化。没有任何产品或组件是绝对安全的。更多信息请从原始设备制造商或零售商处获得，或请见intel.com。

没有任何产品或组件是绝对安全的。

描述的成本降低情景均旨在在特定情况和配置中举例说明特定英特尔产品如何影响未来成本并提供成本节约。情况均不同。英特尔不保证任何成本或成本降低。

预测或模拟结果使用英特尔内部分析或架构模拟或建模，该等结果仅供您参考。系统硬件、软件或配置中的任何差异将可能影响您的实际性能。

特尔并不控制或审计第三方数据。请您审查该内容，咨询其他来源，并确认提及数据是否准确。

本文中提供的所有信息可在不通知的情况下随时发生变更。关于英特尔最新的产品规格和路线图，请联系您的英特尔代表。

本文并未(明示或默示、或通过禁止反言或以其他方式)授予任何知识产权许可。

描述的产品可能包含可能导致产品与公布的技术规格有所偏差的、被称为非重要错误的设计瑕疵或错误。一经要求，我们将提供当前描述的非重要错误。

英特尔未做出任何明示和默示的保证，包括但不限于，关于适销性、适合特定目的及不侵权的默示保证，以及在履约过程、交易过程或贸易惯例中引起的任何保证。