intel + 量子位

最in大模型 专栏

- 最in fashion的大模型话题
- 最inside intel的创新解决方案
- 最in practice的落地实战指南

目录

CONTENTS

如何在英特尔® 平台上实现高效的大语言模型训练后量化

提升 SmoothQuant 量化方法的效力

作者：英特尔公司 陆崟彤、何欣、郭恒、程文华、王畅、王梦妮、沈海豪

本文介绍了可提升大语言模型的训练后量化表现的增强型 SmoothQuant 技术，说明了这项技术的用法，并证明了其在准确率方面的优势。此方法已整合至英特尔® Neural Compressor（https://www.intel.cn/content/www/cn/zh/developer/tools/oneapi/neural-compressor.html）中。英特尔® Neural Compressor 是一个包含量化、剪枝（稀疏性）、蒸馏（知识提炼）和神经架构搜索等多种常用模型压缩技术的开源 Python 库。目前，诸如 TensorFlow、英特尔® Extension for TensorFlow（https://www.intel.cn/content/www/cn/zh/developer/tools/oneapi/optimization-for-tensorflow.html）、PyTorch、英特尔® Extension for PyTorch（https://www.intel.cn/content/www/cn/zh/developer/tools/oneapi/optimization-for-pytorch.html）、ONNX Runtime 和 MXNet 等主流框架，都能与之兼容。

英特尔® Neural Compressor 已经支持多款英特尔® 架构的硬件，比如英特尔® 至强® 可扩展处理器（https://www.intel.cn/content/www/cn/zh/products/details/processors/xeon/scalable.html）、英特尔® 至强® CPU Max 系列（https://www.intel.cn/content/www/cn/zh/products/details/processors/xeon/max-series.html）、英特尔® 数据中心 GPU Flex 系列（https://www.intel.cn/content/www/cn/zh/products/details/discrete-gpus/data-center-gpu/flex-series.html）和英特尔® 数据中心 GPU Max 系列（https://www.intel.com/content/www/us/en/products/details/discrete-gpus/data-center-gpu/max-series.html）。本文涉及的实验基于第四代英特尔® 至强® 可扩展处理器（https://www.intel.cn/content/www/cn/zh/events/accelerate-with-xeon.html）进行。

大语言模型

大语言模型 (Large Language Model, LLM) 需基于海量数据集进行训练，可能拥有数十亿权重参数。其先进的网络结构和庞大的参数量，使它们能够很好地应对自然语言本身的复杂性。完成训练后的大语言模型，可针对各种下游的自然语言处理 (NLP) 和自然语言生成 (NLG) 任务进行调优，让其更适合对话式聊天机器人（如 ChatGPT）、机器翻译、文本分类、欺诈检测和情感分析等任务场景。

大语言模型部署面临的挑战

大语言模型在执行自然语言处理和自然语言生成任务方面表现出色，但其训练和部署颇为复杂，主要面临以下挑战：

- AI 与内存墙（https://medium.com/riselab/ai-and-memory-wall-2cb4265cb0b8）瓶颈问题：算力每两年提高 3.1 倍，内存带宽却只提高 1.4 倍；
- 网络带宽挑战：训练大语言模型需要采用分布式系统，这对网络带宽提出了较高要求；
- 系统资源有限：训练后的模型往往会部署在算力和内存资源均有限的系统上。

因此，采用训练后量化的方法来为大语言模型瘦身，对于实现低时延推理至关重要。

大语言模型的量化

量化是一种常见的压缩操作，可以减少模型占用的内存空间，提高推理性能。采用量化方法可以降低大语言模型部署的难度。具体来说，量化是将浮点矩阵转换为整数矩阵：

$$X_{int8} = round(X_{fp32}/S) + Z$$

其中 X_fp32、S 和 Z 分别为输入矩阵、比例因子和整数零点。

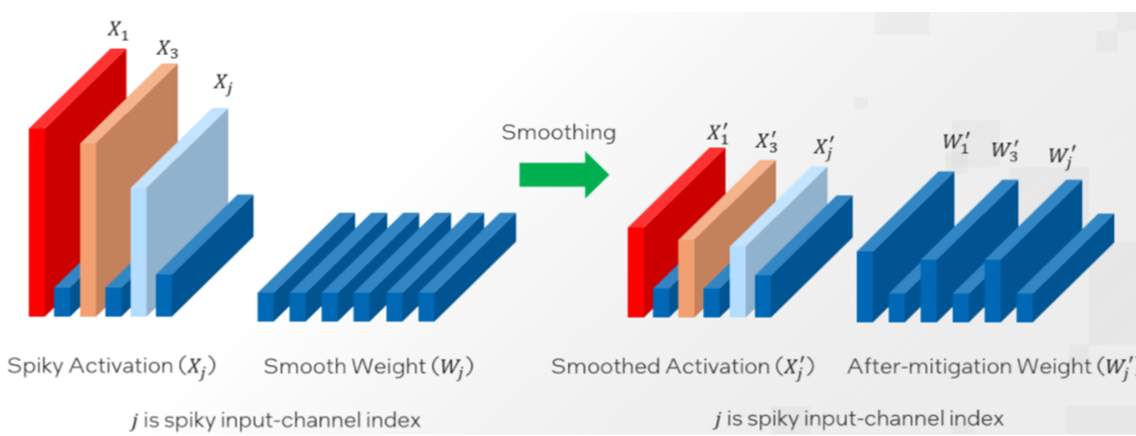
有关每通道 (per-channel) 量化策略虽然可能会减少量化损失，但不能用于激活值量化的原因，请参看 SmoothQuant 相关文档（https://github.com/intel/neural-compressor/blob/master/docs/source/smooth_quant.md）。不过，激活值量化误差损失却是导致模型量化准确率下降的重要因素。为此，人们提出了很多方法来降低激活值量化损失，例如：SPIQ（https://arxiv.org/abs/2203.14642）、Outlier Suppression （https://arxiv.org/abs/2209.13325）和 SmoothQuant（https://arxiv.org/abs/2211.10438）。这三种方法思路相似，即把激活值量化的难度转移到权重量化上，只是三者在转移难度的多少上有所不同。

增强型 SmoothQuant

SmoothQuant 引入了一个超参数 α 作为平滑因子来计算每个通道的量化比例因子，并平衡激活值和权重的量化难度。

$$s_j = max(|X_j|)^{\alpha}/max(|W_j|)^{1-\alpha}$$

其中 j 是输入通道索引。



对于 OPT 和 BLOOM 等大多数模型来说，α=0.5 是一个能够较好实现权重和激活值量化难度分割的平衡值。模型的激活异常值越大，就越需要使用更大的 α 值来将更多的量化难度转移到权重上。

原始的 SmoothQuant 旨在通过针对整个模型使用一个固定值 α 来分割权重和激活值的量化难度。然而，由于激活异常值的分布不仅在不同模型之间存在差异，而且在同一模型的不同层之间也不尽相同，因此，本文推荐使用英特尔® Neural Compressor 的自动调优能力，逐层获取最佳 α 值。

相关方法包括以下五个主要步骤（伪代码如下所示）：

1. 通过特殊的回调函数 register_forward_hook 捕获 (hook) 模型各层的输入和输出值。
2. 根据用户定义的 α 范围和步长生成一个 α 值列表。
3. 根据给定的 α 值重新计算平滑因子并调整参数（权重值和激活值）。

4. 对权重执行每通道量化与反量化（quantization_dequantization），对输入值执行每张量（per-tensor）量化与反量化，以预测与给定 α 值对应的每层输出值。
5. 计算相对实际输出值的均方损失，将调整后的参数恢复回来，并保存每层的最佳 α 值。

Algorithm: α auto-tuning

Data: neural-network model of layers [L_1, L_2,...,L_n]

Result: Layer-wise optimal α values {L_1: α_1, L_2: α_2,..., L_n: α_n}

```
Hook input & output values of all layers
Generate user-defined α values [α1, α2, ..., αn]
for Li in [L1, L2,...,Ln]:
    for αj in [α1, α2, ..., αn]:
        recalculate smoothing factor s(αj, Wi(Li), Xi(Li))
        adjust Wi& Xi -> W_i = X
        Xi = Xi diag(s)^-1
        Wi = diag(s) Wi    # Adjust parameters
        Xqi = per_tensor_quant_dequant(Xi)
        Wqi = per_channel_quant_dequant(Wi)
        Yqi = Wqi * Xqi  # Predict output
        loss = MSE(Yqi, Yi)
        record loss and recover parameters
```

本文提出的方法支持用多个标准（如最小值、最大值和平均值）来确定 Transformer 块的输入层归一化（LayerNorm）操作的 α 值。实验发现，将 α 范围设为 [0.3, 0.7]，步长设为 0.05，对大多数模型来说都能达到很好的平衡。

这一方法有两个显著特点：一是全自动化，二是比原始方法支持的融合模式多。

下图提供了在 BLOOM-1b7 模型上执行 SmoothQuant α 值自动调优的样例代码：

```
from neural_compressor.adaptor.torch_utils.smooth_quant import TorchSmoothQuant
model_name = 'bigscience/bloom-1b7'
model = transformers.AutoModelForCausalLM.from_pretrained(model_name, torchscript=True)
alpha = 'auto'
sq = TorchSmoothQuant(model, dataloader)
sq.transform(alpha)
```

启用增强型 SmoothQuant 的样例代码

用户只需传递一个模型名称（model_name）和一个数据加载器。值得注意的是，模型分析主要依靠的是 Torch JIT。用户可以在加载 Hugging Face 模型（https://huggingface.co/models）时将 torchscript 设置为 True，或将 return_dict 设置为 False。更多信息请参阅英特尔® Neural Compressor 文档（https://github.com/intel/neural-compressor/blob/master/docs/source/smooth_quant.md）。

结果

本文提出的增强型 SmoothQuant 的主要优势在于提高了准确率。

经过对多种主流大语言模型的评估，具备自动调优能力的 INT8 SmoothQuant 最后一个词元（last-token）的预测准确率要高于原始 INT8 SmoothQuant 和 FP32 基线方法。详见下图：

Model\Last token accuracy	FP32	INT8 (w/o SmoothQuant)	INT8 (w/ SmoothQuant)	INT8 (w/ SmoothQuant auto tuning)
bigscience/bloom-560m	65.20%	63.44%	66.48% (alpha=0.5)	64.76% (alpha: 95.9% over 0.6, 4.1% in [0.4, 0.6])
bigscience/bloom-1b7	71.43%	67.78%	72.56% (alpha=0.5)	72.58% (alpha: 55.1% over 0.6, 30.6% in [0.4, 0.6], 14.3% under 0.4)
bigscience/bloom-3b	73.97%	69.99%	74.02% (alpha=0.5)	74.16% (alpha: 100% over 0.6)
bigscience/bloom-7b1	77.44%	75.46%	77.02%(alpha=0.5)	77.45% (alpha: 91.8% over 0.6, 4.9% in [0.4, 0.6], 3.3% under 0.4)
bigscience/bloom-176b	84.17%	82.13%	83.52% (alpha=0.6)	-
facebook/opt-125m	63.89%	63.48%	63.44% (alpha=0.5)	64.14% (alpha: 59.4% over 0.6, 8.1% in [0.4, 0.6], 32.4% under 0.4)
facebook/opt-1.3b	75.41%	73.59%	70.94% (alpha=0.5)	74.80% (alpha: 69.9% over 0.6, 24.7% in [0.4, 0.6], 5.5% under 0.4)
facebook/opt-2.7b	77.79%	78.57%	78.60%(alpha=0.5)	78.25% (alpha: 73.2% over 0.6, 21.6% in [0.4, 0.6], 5.2% under 0.4)
facebook/opt-6.7b	81.26%	76.65%	81.58%(alpha=0.5)	81.39% (alpha: 68.0% over 0.6, 26.8% in [0.4, 0.6], 5.2% under 0.4)
EleutherAI/gpt-j-6B	79.17%	78.82%	78.84%(alpha=0.6)	79.29% (alpha: 96.4% over 0.6, 3.6% in [0.4, 0.6])

FP32 基线方法、INT8（启用和不启用 SmoothQuant）以及 INT8（启用本文提出的增强型 SmoothQuant）的准确率对比

从上图可以看出，在 OPT-1.3b 和 BLOOM-1b7 模型上，本文提出的增强型 SmoothQuant 的准确率比默认的 SmoothQuant 分别高 5.4% 和 1.6%。量化后的模型也缩小到 FP32 模型的四分之一，大大减少了内存占用空间，从而有效地提升大模型在英特尔® 平台上的推理性能。

更全面的结果请见 GitHub 存储库（https://github.com/intel/neural-compressor/blob/master/docs/source/smooth_quant.md）。同时，也欢迎您创建拉取请求或就 GitHub 问题（https://github.com/intel/neural-compressor/issues）发表评论。期待听到您的反馈意见和建议。

作者：

英特尔公司人工智能资深架构师沈海豪、英特尔公司人工智能资深软件工程师程文华、英特尔公司人工智能软件工程师陆崟彤、何欣、郭恒、王畅、王梦妮，他们都在从事模型量化及压缩的研究与优化工作。

用基于英特尔® SGX 的可信执行环境有效应对大语言模型隐私和安全挑战

作者：英特尔公司 俞巍、李志强、李青青、龚奇源

可信执行环境是什么？大语言模型为什么需要它？

OpenAI 的 GPT 系列大语言模型（Large Language Mode，以下缩写为 LLM）的兴起与应用，也带来了诸如数据泄露、数据滥用、模型被攻击和知识产权被窃取等一系列隐私和安全风险或挑战。

可信执行环境（Trusted Execution Environment，以下缩写为 TEE）是一项基于软硬件组合创建安全执行环境，能够更好地确保计算和数据处理机密性和完整性。其关键机制为：

- **安全隔离：**通过硬件加密和内存隔离等硬件隔离技术，将敏感数据和关键代码与其他应用及操作系统相隔离，从而确保它们即使在系统其他部分被攻击或受到恶意软件影响时也能够得到更好的保护。
- **安全验证：**在启动过程中进行身份验证和完整性检查，确保只有经过授权的代码和数据可以在其中运行，以此防止恶意软件或未经授权的访问。
- **安全执行环境：**提供包含加密算法、安全协议和密钥管理等防护功能的执行环境，用于处理敏感数据和执行关键算法，以增强数据在执行过程中的保密性和完整性。

TEE 与 LLM 可在多行业、多场景融合，TEE 可用于为 LLM 提供颇具商业落地价值的隐私和数据保护创新解决方案。

2. LLM 与 TEE 的融合需求

LLM在许多行业的不同场景都有着广泛应用，例如金融行业的风险评估和交易分析，医疗保健领域的医学图像识别、病历纪录和疾病预测，以及法律和合规行业的法律咨询、合同审查和文书处理等。这些行业或场景中涉及到的数据多为重要敏感的交易数据或个人数据，必须得到有效保护。

将 TEE 与 LLM 融合，有助于在这类场景中更好地保障数据在 LLM 模型训练和推理过程中的保密性。训练阶段，TEE 中的数据处理都处于加密状态；推理阶段，TEE 则可保护用户输入和模型结果的隐私。同时，其硬件隔离和安全验证机制可以更有效地防止未经授权的访问和攻击，增强模型运行时的安全性。

3. TEE 与 LLM 融合的挑战：资源和性能限制

- **资源限制：**TEE 的计算资源和存储空间通常都非常有限，LLM 庞大的模型参数和计算需求可能会超出一般 TEE 的能力范围。
- **性能下降：**I/O 数据的加密和安全计算操作会引入额外的计算开销，导致模型训练和推理性能有一定程度下降。基于算法的解决方案可减少模型规模和计算需求，以适应 TEE 的资源限制，但 CPU 仍会成为制约 LLM 训练的算力瓶颈。

4. 基于英特尔® 平台的解决方案：加速 TEE 与 LLM 融合应用

4.1 基于英特尔® SGX/TDX[1] 的 TEE 解决方案

英特尔自第三代英特尔® 至强® 可扩展处理器开始内置英特尔® 软件防护扩展（英特尔® SGX）技术，其安全飞地的容量最多可达单颗 CPU 512GB，双路共计 1TB 容量，可满足目前千亿大模型的执行空间需求。此外，该技术提供支持的机密计算可实现应用层、虚拟机 (VM)、容器和功能层的数据隔离。无论是在云端、边缘还是本地环境，都能确保计算与数据始终在私密性和安全性上获得更全面的保护，以免暴露给云服务提供商、未经授权的管理员和操作系统，甚至是特权应用。另一方面，英特尔® Trust Domain Extension（英特尔® TDX）可将客户机操作系统和虚拟机应用与云端主机、系统管理程序和平台的其他虚拟机隔离开来。它的信任边界较英特尔® SGX 应用层的隔离边界更大，使受其保护的机密虚拟机比基于英特尔® SGX 的安全飞地的应用更易于进行大规模部署和管理，在部署 LLM 这类复杂应用时，TDX 在易用性上更具优势。此外，今年推出的全新第四代英特尔® 至强® 可扩展处理器内置英特尔® AMX，可大幅提升矩阵运算性能，而英特尔® SGX/TDX 也可为英特尔® AMX、英特尔® DL Boost 等计算指令提供支持，进而为 TEE 中的大模型赋予快速落地+优化性能的双重优势。

图 1. SGX/TDX 的可信边界

构建完善的 TEE 生态系统对推动 LLM 的应用和发展至关重要。开发者需要能够简化集成和使用过程的面向 TEE 的开发者工具和框架。为此，英特尔在 SDK 的基础上，推出了开源的 lib OS 项目 Gramine 来帮助开发者更好地使用基于英特尔® SGX 的 TEE，助推 LLM 与 TEE 的融合。

- **4.1.1 大语言模型推理**

使用私有数据进行个性化训练的大模型不仅包含私有数据信息，其查询本身也具有隐私性，尤其是基于边端的非安全环境部署。基于英特尔® SGX/TDX 的 TEE 可为大模型提供更安全的运行环境，在数据上传云端前，查询可先通过客户端对传输内容加密，云端只需在英特尔® SGX/TDX 中解密查询问题，然后输入大模型的推理服务中，并将所得结果在云端的 TEE 中加密后传输回本地客户端。在整个工作流程中，客户端以外的数据和运行态程序均处于密态环境当中，效率远远高于其他基于纯密码学的解决方案。目前像 LLAMA 7B、ChatGLM 6B 等模型都可以在该 TEE 方案上满足实时可交互性能的运行。图 2 展示了使用 LLM 部署知识问答的参考设计。基于英特尔® SGX/TDX 的 TEE 为实际部署 LLM 中的自有知识产权保护提供了一套完整的方案，优化整个模型在查询、传输和推理过程中的安全保护。

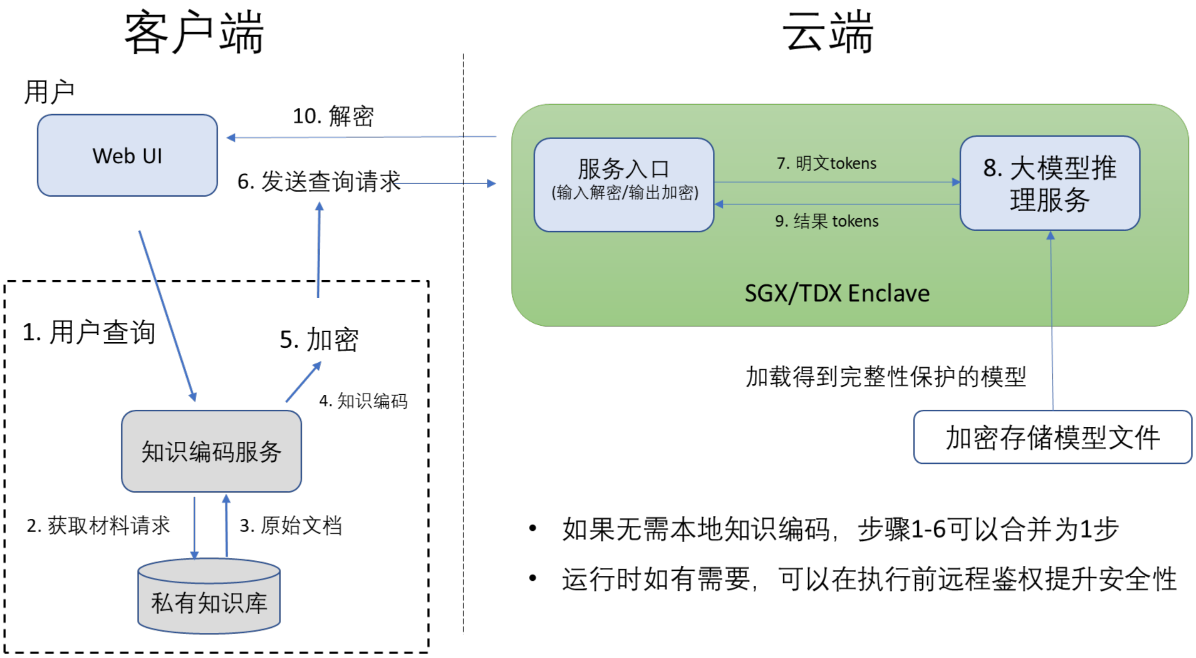


图 2. 基于 TEE 的大语言模型私密问答

- **4.1.2 联邦学习**

借助基于 TEE 的联邦学习解决方案[2]（见图 3），就可在多机构之间实现基于 NLP 的深度学习，例如使用 BERT 的命名体识别。在金融和医疗等行业提升准确性，实现多机构数据互通，同时更好避免数据泄露。

此方案中每个参与方包含一个 Avalon[3] 管理模块和 Gramine 工作负载，均运行在英特尔® SGX 的安全飞地中，在管理模块彼此间的远程认证完成执行后，即可启动联邦学习过程，参与方在本地使用各自的数据集进行训练，然后将梯度上传至聚合方，聚合方进行聚合后将平均梯度下发至各参与方，以继续进行下一轮训练。对比图 4 所示的 BERT + CRF 模型[4]，此方案可以在强化隐私保护的同时，让性能损失维持在 50% 以下[2]。

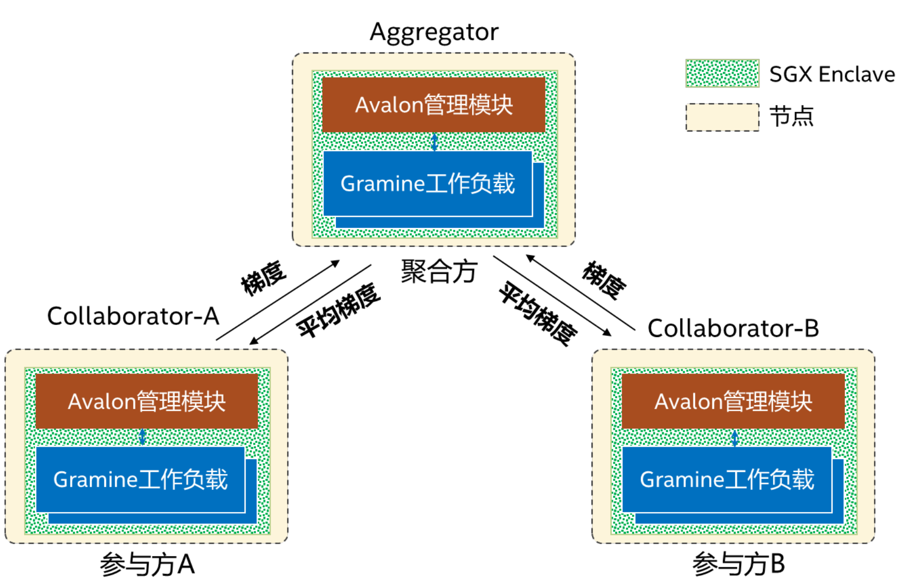


图 3. 基于 TEE 的联邦学习

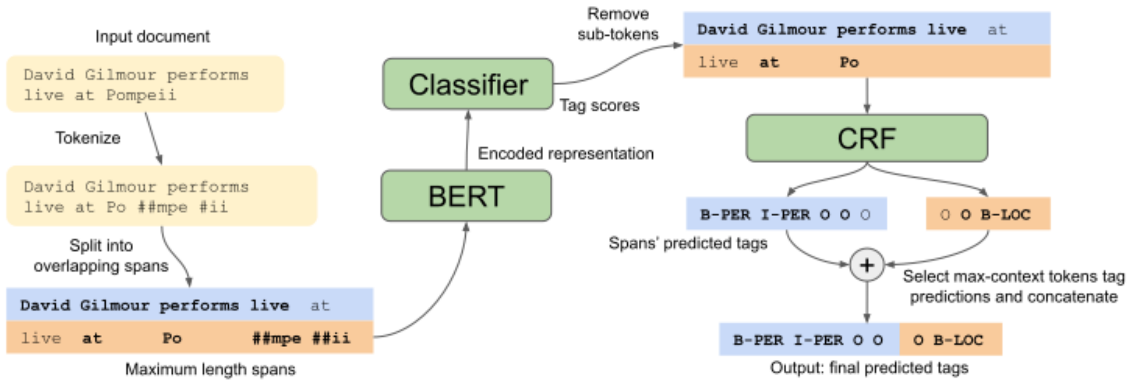


图 4. BERT + CRF 模型[4]

4.2 BigDL：端到端大模型和 TEE 融合的方案

据行业用户反馈，LLM 在端到端应用中的痛点包括：

- **软件栈复杂，难以确保端到端应用的安全。**LLM 的训练和推理常依赖较多的软件栈、服务和硬件。为保护用户数据和模型产权，需确保每个环节的安全性（不同硬件、运行环境、网络和存储等）。
- **计算量大，且对性能敏感。**LLM 的计算量非常大，需引入足够多的性能优化。但是，不同模型、平台和软件栈需要使用不同的优化方案，要在特定平台上实现更理想的性能，需要长时间的性能调优。

为解决这些痛点，由英特尔主导的开源项目 BigDL，近期就推出了针对 LLM 的隐私保护方案，其两大主要功能为：

- **提供端到端的安全保护：**在不修改代码的情况下，为单机和分布式的 LLM 应用提供端到端的安全保护功能。具体包括，基于英特尔® SGX/TDX 的 TEE、远程证明、统一的密钥管理接口和透明的加解密 API 等。
- **实现一站式性能优化：**BigDL Nano 提供的针对 LLM 的一站式性能优化方案，可让现有 LLM 应用在几乎不用修改代码的情况下受益于英特尔® AMX、英特尔® AVX-512 和英特尔® Extension for PyTorch。同时，用户还可利用 BigDL Nano 提供的 LLM API，快速构建应用。
-

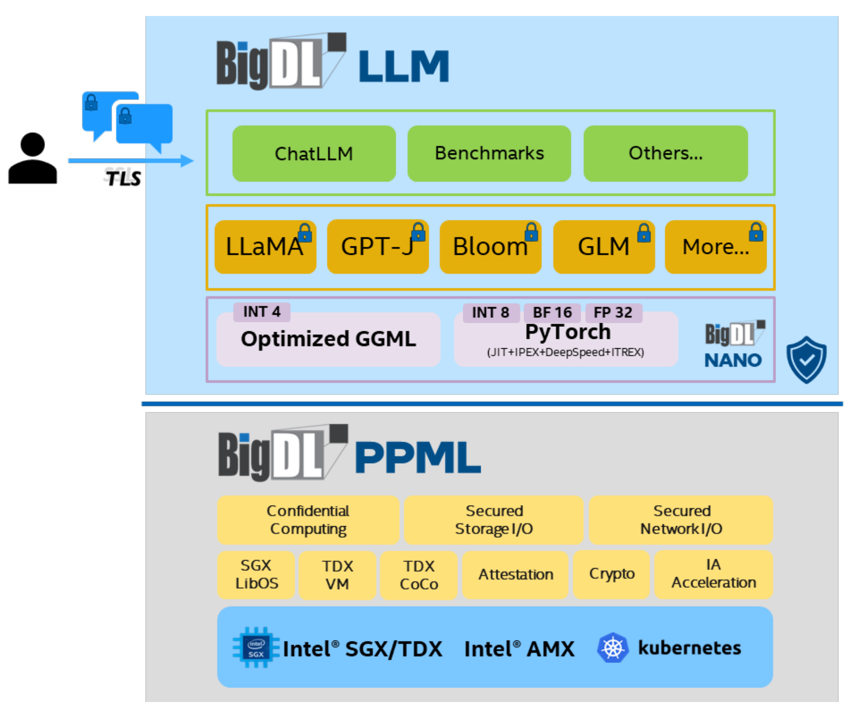


图 5. BigDL 端到端安全的大模型方案

如图 6 所示，在应用了 PPML（Privacy Preserving Machine Learning，隐私保护的机器学习）提供的安全技术后，由于更强的安全和隐私保护会带来额外开销，因此端到端应用性能会略有下降；但应用了 BigDL Nano 提供的优化功能后，端到端的性能得到了显著改善*，总体性能甚至高于没有任何保护的明文性能。

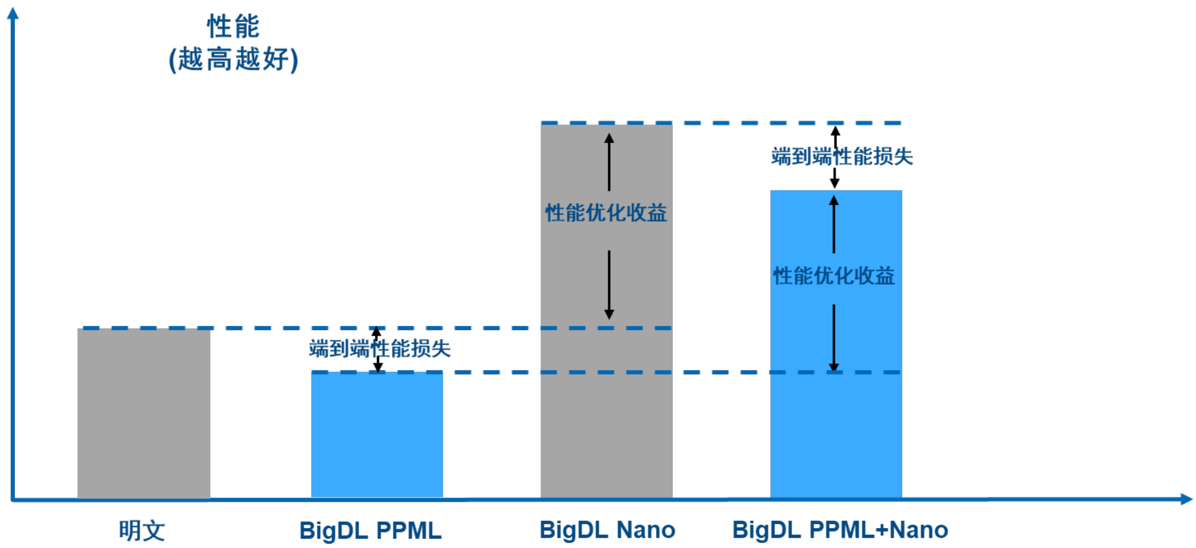


图 6. BigDL PPML + Nano 端到端性能损失情况

目前，该方案已经开源，并开始陆续交付给行业客户进行测试和集成[5]。

5. 未来趋势

TEE 提供了隐私保护和数据安全防护功能的创新解决方案，将在 LLM 实际落地过程中扮演重要角色。通过将二者融合，可更好地保障数据在训练和推理过程中的保密性，增强对未经授权访问和模型结果篡改的防御。然而，在 TEE 中保护用户隐私的同时，需要平衡性能需求，随着大模型对于计算需求的大幅提升，算力可能会执行在异构硬件上，TEE 和异构硬件的结合将成为未来发展趋势。随着 CPU 性能的提升以及内置 AI 加速技术的升级和更新，在方便部署的场景下，CPU 会是大模型推理和 TEE 结合的首选，在训练的场景下，基于 CPU 的 TEE 结合异构硬件的加密支持，则会是大模型训练甚至大模型联邦训练的技术方向。英特尔将一如既往地以软硬结合的产品技术组合促进开发者参与，推动 LLM 与 TEE 的融合。

作者简介：

英特尔公司 AI 架构师俞巍，英特尔公司平台安全资深架构师李志强，英特尔公司安全软件研发工程师李青青，英特尔公司软件架构师龚奇源，都在从事 AI 和 SGX/TDX 相关工作。

[1] SGX/TDX: https://www.intel.cn/content/www/cn/zh/customer-spotlight/cases/innovation-data-protection-with-security-engines.html

[2] Wei Yu. et al. 2022, TEE based Cross-silo Trustworthy Federated Learning Infrastructure, FL-IJCAI'22

[3] https://github.com/hyperledger-archives/avalon

[4] Souza, F., Nogueira, R. and Lotufo, R. 2020, Portuguese Named Entity Recognition using Bert-CRF, arXiv.org（请参见 https://arxiv.org/pdf/1909.10649.pdf ）

[5] https://github.com/intel-analytics/BigDL/tree/main/python/llm

*性能优化效果与具体平台、模型和环境有关

只改一行代码
在第四代至强® 可扩展平台上
高效微调优化 ChatGLM-6B

作者：英特尔公司夏磊

大语言模型的应用与微调优化必要性

ChatGPT 的横空出世开启了大语言模型（LLM）的普及元年，BERT、GPT-4、ChatGLM 等模型的非凡能力则展现出类似通用人工智能（AI）的巨大潜力，也因此得到了多行业、多领域的广泛关注。为加速这些大模型与特定领域的深度融合，以及更好地适应特定任务，基于任务特性对这些模型进行定制化微调至关重要。然而，它们庞大的参数使得用传统方式对大模型进行调优面临诸多挑战，不仅要求相关人员熟练掌握微调技巧，还需要付出巨大的训练成本。近年来，出现了参数高效微调（Parameter-Efficient Fine-Tuning, PEFT）和提示微调（Prompt-tuning）技术。这些技术因其成本更低、应用方式更简单便捷，正在逐渐取代大模型传统调优方法。

本文结合目前在中文应用场景中具有出色表现的开源预训练大模型 ChatGLM-6B，介绍如何通过对其开源 Prompt-tuning 代码进行极少量的修改，并结合第四代英特尔® 至强® 可扩展处理器[i]的全新内置 AI 加速引擎——英特尔® 高级矩阵扩展（Intel® Advanced Matrix Extension，简称英特尔® AMX）及配套的软件工具，来实现高效、低成本的大模型微调。

基于英特尔® 架构硬件的微调优化方案

本文通过以下三个方面实现了基于第四代英特尔® 至强® 可扩展处理器的 ChatGLM 高效微调优化：

1. **借助英特尔® AMX，大幅提升模型微调计算速度** AMX 是内置于第四代英特尔® 至强® 可扩展处理器中的矩阵乘法加速器，能够更快速地处理 BFloat16（BF16）或 INT8 数据类型的矩阵乘加运算，从而显著提升模型训练和推理的性能。

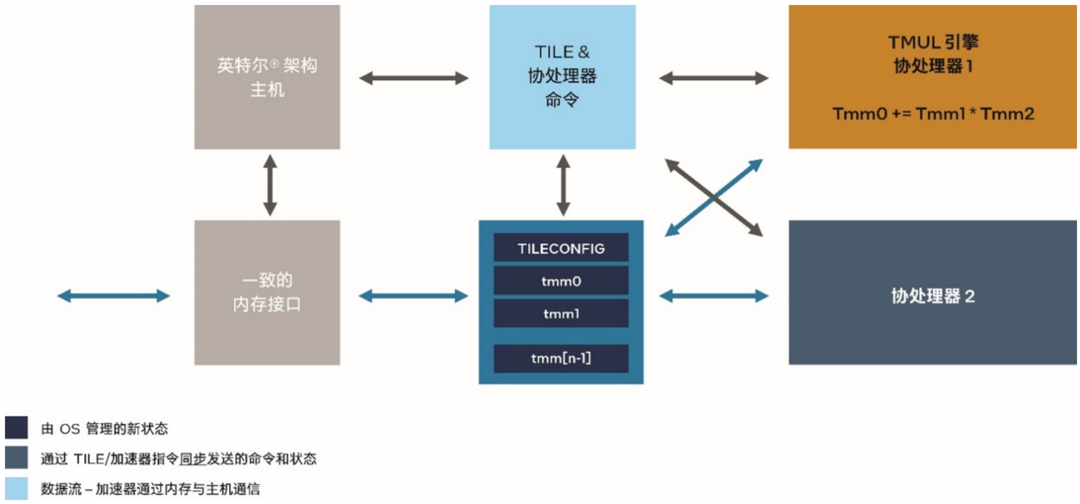


图 1. 英特尔® AMX 技术架构

目前，现行的 PyTorch 框架中，已经可以通过具备 BF16 自动混合精度功能自动实现对 AMX 加速器的利用。

就 ChatGLM-6B 而言，其开源微调代码的 autocast_smart_context_manager() 函数，也已具备对 CPU 自动混合精度的支持。因此，只需在启动微调时加入 CPU 自动混合精度的使能参数即可直接利用英特尔® AMX 带来的优势。

```
    def autocast_smart_context_manager(self, cache_enabled: Optional[bool] = True):
        """
        A helper wrapper that creates an appropriate context manager for `autocast` while feeding it the desired
        arguments, depending on the situation.
        """
        #print(self.use_cpu_amp)
        if self.use_cuda_amp or self.use_cpu_amp:
            if is_torch_greater_or_equal_than_1_10:
                ctx_manager = (
                    torch.cpu.amp.autocast(cache_enabled=cache_enabled, dtype=self.amp_dtype)
    if self.use_cpu_amp
                    else torch.cuda.amp.autocast(cache_enabled=cache_enabled, dtype=self.amp_dtype)
                )
            else:
                ctx_manager = torch.cuda.amp.autocast()
        else:
            ctx_manager = contextlib.nullcontext() if sys.version_info >= (3, 7) else contextlib.suppress()

        return ctx_manager
```

图 2. 通过 trainer.py 中的 autocast_smart_context_manager() 函数，
在 ChatGLM-6B 开源 prompt-tuning 目录下实现对 CPU 和 GPU 的自动混合精度支持

具体方法是在启动微调的 train.sh 脚本时做如下修改：

```
python3 main.py \
 --do_train \
 ...
 --half_precision_backend cpu_amp \
 --bf16
```

2. 结合英特尔® MPI 库充分利用处理器架构特点和多核配置，发挥 CPU 的整体效率

第四代英特尔® 至强® 可扩展处理器最多可拥有 60 个内核。这些内核通过 4 个集群（cluster）的方式进行内部组织。理论上，当多个处理器内核并行处理一个计算任务并需要共享或交换数据时，同一个集群内的内核之间的通信时延较低。因此，在使用 PyTorch 框架进行模型微调时，我们可以将同一个集群上的内核资源分配给同一个 PyTorch 实例，从而为单个实例提供更理想的计算效率。此外，通过利用 PyTorch 的分布式数据并行（Distributed Data Parallel, DDP）功能，还可将两个 CPU 上的 8 个集群的内核资源汇集在一起，充分发挥整体效率。

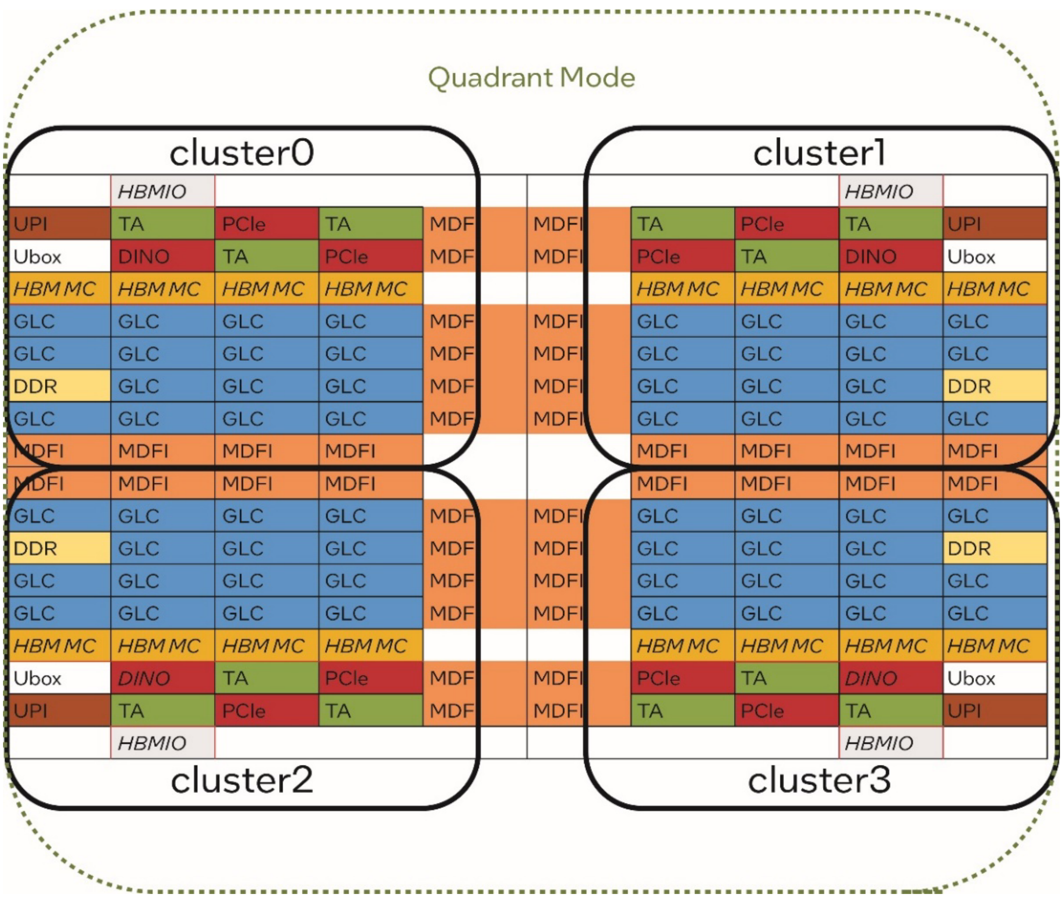


图 3. 第四代英特尔® 至强® 可扩展处理器的内部集群（cluster）架构

为实现从应用程序代码到数据通信的整体简化，PyTorch 框架支持多种分布式数据并行后端（backend），其中 MPI 后端方式能够很好地满足我们的优化需求。

Backend	gloo		mpi		nccl	
Device	CPU	GPU	CPU	GPU	CPU	GPU
send	✓	✗	✓	?	✗	✓
recv	✓	✗	✓	?	✗	✓
broadcast	✓	✓	✓	?	✗	✓
all_reduce	✓	✓	✓	?	✗	✓
reduce	✓	✗	✓	?	✗	✓
all_gather	✓	✗	✓	?	✗	✓
gather	✓	✗	✓	?	✗	✓
scatter	✓	✗	✓	?	✗	✓
reduce_scatter	✗	✗	✗	✗	✗	✓
all_to_all	✗	✗	✓	?	✗	✓
barrier	✓	✗	✓	?	✗	✓

图 4. PyTorch 支持的多种分布式数据并行的后端（来源：PyTorch 官网[ii]）

但是，通过 pip 或 conda 来安装的预编译 PyTorch 二进制包中并未将 MPI 的后端作为缺省功能编译。因此，我们需要安装 MPI 协议工具库并通过手工编译来获得对 MPI 后端的支持。

英特尔® MPI 库[iii] 是一个实现 MPICH 规范的多结构消息传递库，使用该库可创建、维护和测试能够在英特尔® 处理器上实现更优性能的先进和复杂的应用。它采用 OFI 来处理所有通信，能够提供更高的吞吐量、更低的时延和更简单的程序设计。

以下是基于英特尔® MPI 库的 PyTorch 编译步骤:

#下载英特尔® MPI 库并安装

```
wget https://registrationcenter-download.intel.com/akdlm/IRC_NAS/718d6f8f-2546-4b36-b97b-bc58d5482ebf/l_mpi_oneapi_p_2021.9.0.43482_offline.sh
bash l_mpi_oneapi_p_2021.9.0.43482_offline.sh
source your_installation_path/vars.sh
```

安装 PyTorch 编译依赖包

```
pip install numpy pyyaml mkl mkl-include setuptools cmake cffi typing
pip install mkl_include
```

#下载 PyTorch 源码并完成编译、安装

```
git clone --recursive  https://github.com/pytorch/pytorch.git
cd pytorch
python setup.py install
```

在获得了支持 MPI 后端的 PyTorch 后，只需按如下方法在 ChatGLM Prompt-tuning 目录下的 main.py 修改一行代码：

将 dist.init_process_group(backend='gloo', world_size=1, rank=0) 改为 **dist.init_process_group(backend='mpi')**

```
if training_args.do_train:
    checkpoint = None
    if training_args.resume_from_checkpoint is not None:
        checkpoint = training_args.resume_from_checkpoint
    # elif last_checkpoint is not None:
    #exit()
    model.gradient_checkpointing_enable()
    model.enable_input_require_grads()
    os.environ['MASTER_ADDR'] = 'localhost'
    os.environ['MASTER_PORT'] = '12355'
    dist.init_process_group(backend='gloo', world_size= 1, rank=0)
    train_result = trainer.train(resume_from_checkpoint=checkpoint)
    # trainer.save_model()  # Saves the tokenizer too for easy upload
```

图 5. 修改前的 main.py

```python
if training_args.do_train:
    checkpoint = None
    if training_args.resume_from_checkpoint is not None:
        checkpoint = training_args.resume_from_checkpoint
    # elif last_checkpoint is not None:
    #     checkpoint = last_checkpoint
    model.gradient_checkpointing_enable()
    model.enable_input_require_grads()
    world_size = int(os.environ['PMI_SIZE'])
    world_rank = int(os.environ['PMI_RANK'])
    dist.init_process_group(backend='mpi', world_size= world_size, rank= world_rank)
    train_result = trainer.train(resume_from_checkpoint=checkpoint)
    # trainer.save_model()  # Saves the tokenizer too for easy upload
```

图 6. 修改后的 main.py

3. **利用至强®** CPU Max 系列**集成的 HBM 满足大模型微调所需的大内存带宽**

基于 Transformer 的大模型，由于参数、训练数据和模型规模的复杂程度较高，因此内存复杂度通常是 $O(n^2)$。这意味着这些大模型需要足够大的内存带宽支持才能获得更好的运行性能。英特尔® 至强® CPU Max 系列[iv]，配备 64 GB 的 HBM2e 高带宽内存，为在 CPU 上高效运行大模型提供了高达~1TB/s 的内存带宽支持。

该 CPU 集成的 HBM，能够在 3 种模式下灵活配置：

- HBM-Only 模式——支持内存容量需求不超过 64 GB 的工作负载，具备每核 1 至 2 GB 的内存扩展能力，无需更改代码和另购 DDR，即可启动系统。
- HBM Flat 模式——可为需要大内存容量的应用提供灵活性，通过 HBM 和 DRAM 提供一个平面内存区域（flat memory region），适用于每核内存需求 >2 GB 的工作负载。可能需要更改代码。
- HBM 高速缓存模式——为内存容量 >64 GB 或每核内存需求 >2GB 的工作负载提供更优性能。无需更改代码，HBM 将用作 DDR 的高速缓存。

针对 ChatGLM-6B 微调，试验结果显示：与其他两种模式相比，**HBM 高速缓存模式**在性能和使用方便性方面均更胜一筹。

在英特尔® 至强® CPU Max 系列产品上，结合之前的两项优化，我们可以通过以下命令行启动 ChatGLM-6B 微调：

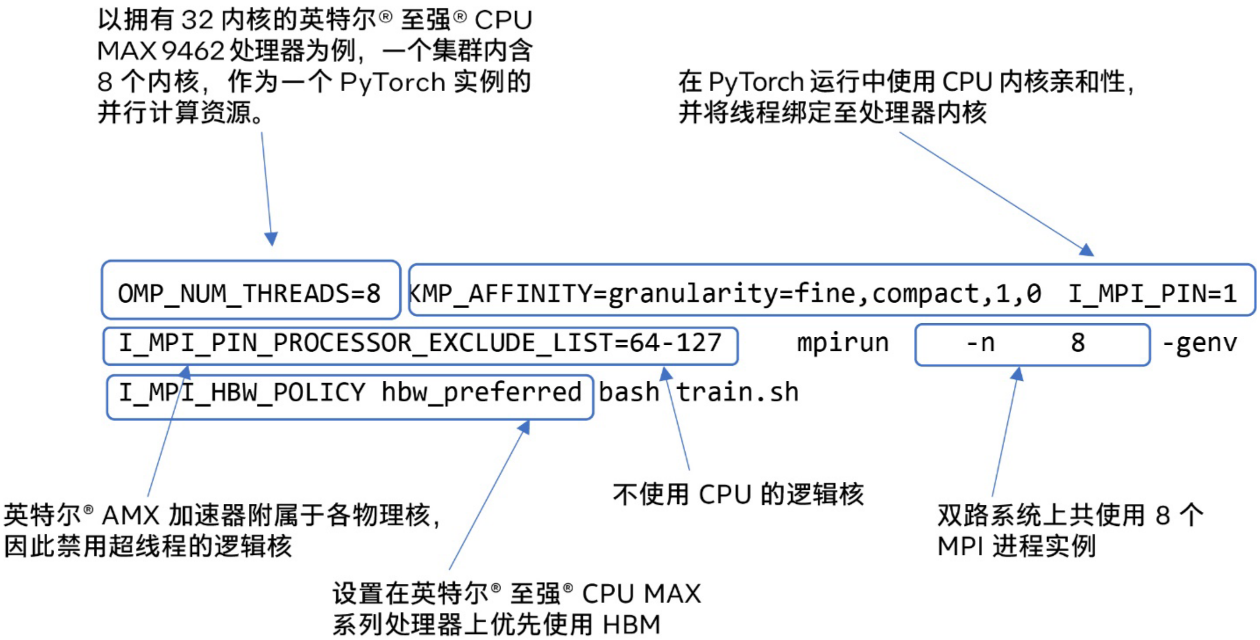


图 7. 在拥有 32 个物理核的英特尔® 至强® CPU Max 9462 双路服务器上启动微调

优化结果

通过以上简单软、硬件综合优化，无须采用昂贵的 GPU 硬件，即可实现对 ChatGLM-6B 模型的高性能微调。

注： 以上代码修改需要配合 python 工具包 accelerate 0.18.0 和 transformers 4.28.0。

作者简介：

夏磊，英特尔（中国）有限公司人工智能首席工程师，拥有近 20 年的人工智能从业经验，在软件算法、自动控制和工程管理等领域积累了丰富经验。

---

[i] https://www.intel.cn/content/www/cn/zh/products/docs/processors/xeon-accelerated/4th-gen-xeon-scalable-processors-product-brief.html
[ii] https://pytorch.org/docs/stable/distributed.html
[iii] https://www.intel.cn/content/www/cn/zh/developer/tools/oneapi/mpi-library.html
[iv] https://www.intel.cn/content/dam/www/central-libraries/cn/zh/documents/2023-03/23-cmf54-xeon-cpu-max-series-product-brief.pdf
[v] https://www.intel.cn/content/dam/www/central-libraries/cn/zh/documents/2023-03/23-cmf54-xeon-cpu-max-series-product-brief.pdf

创建 OpenVINO™ Stateful 模型与 Runtime 流水线，赋能 ChatGLM

作者：赵桢、罗成、李亭骞、邹文艺

引言

自大语言模型 (LLM) 成为热点话题以来，涌现了一大批中文大语言模型并在优化平台中得到了积极部署。ChatGLM 正是广受好评的主流中文大语言模型之一。然而，由于 ChatGLM 模型尚未成为 Transformer 生态的原生模型，因此，官方 optimum 扩展库对其仍缺乏支持。本文提供了一种使用 OpenVINO™ opset 重构该模型架构的便捷方法。该方案包含专为 ChatGLM 定制的优化节点，且这些节点都利用英特尔® 高级矩阵扩展（Intel® Advanced Matrix Extensions，缩写为英特尔® AMX）内联和 MHA（Multi-Head Attention，多头注意力）融合实现了高度优化。

*请注意，本文仅介绍了通过为 ChatGLM 创建 OpenVINO™ stateful 模型实现优化的解决方案。本方案受平台限制，必须使用内置了英特尔® AMX 的第四代英特尔® 至强® 可扩展处理器[i]（代号 Sapphire Rapids）。笔者不承诺对该解决方案进行任何维护。

ChatGLM 模型简介

笔者在查看 ChatGLM 原始模型的源码时，发现 ChatGLM 与 Optimum *ModelForCasualML* 并不兼容，而是定义了新的类 *ChatGLMForConditionalGeneration*。该模型的流水线回路包含 3 个主要模块（Embedding、GLMBlock 层和 lm_logits），结构如下：

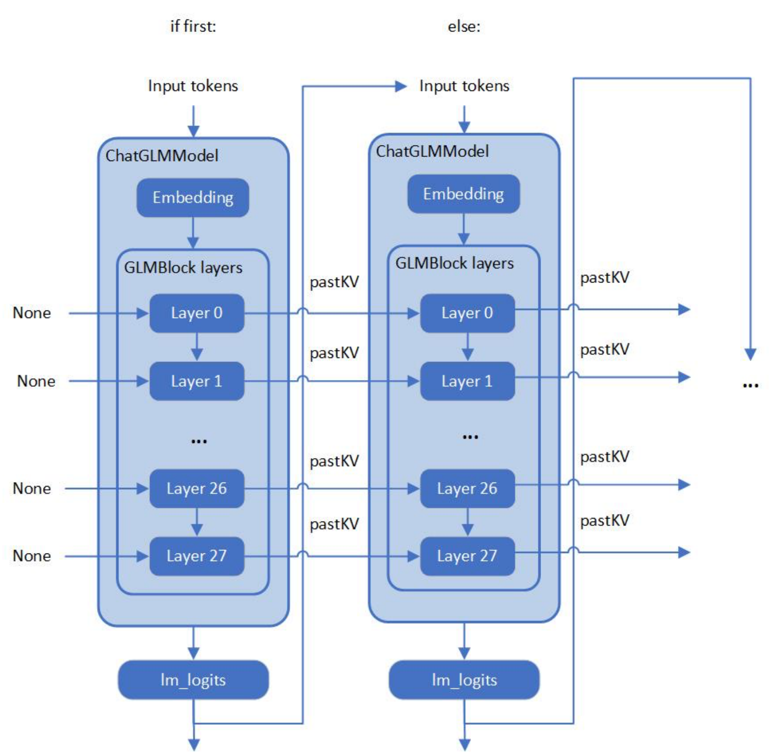


图 1. ChatGLM 的模型结构

如上图所示，整个流水线实际要求模型有两个不同的执行图，使用输入提示符进行首次推理时不需要 KV 缓存作为 *GLMBlock* 层的输入。从第二次迭代开始，QKV 注意力机制的上一次结果将成为当前一轮模型推理的输入。随着生成符的长度不断增加，在流水线推理过程中，模型输入和输出之间将存留大量的大型内存副本。以 ChatGLM6b 默认模型配置[ii] 为示例，输入和输出阵列之间的内存副本类似于以下伪代码，其内存拷贝的开销由模型的参数 hidden_size 以及迭代的次数决定：

```
while(eos_token_id || max_seq_len){
  memcpy(model_inp, model_outp, num_layer*2*sizeof(model_outp)* hidden_size)
  model_outp.push_back(gen_token)
}
```

1. 因此，本文要解决的两大关键问题是：

- 如何优化模型推理流水线来消除模型输入和输出之间的内存副本
- 如何通过重新设计执行图来优化 *GLMBlock* 模块

构建 OpenVINO™ stateful 模型实现显著优化

首先，需要分析 *GLMBlock* 层的结构，尝试封装一个类并按以下工作流来调用 OpenVINO™ opset。接着，将图形数据序列化为 IR 模型 (.xml, .bin)。

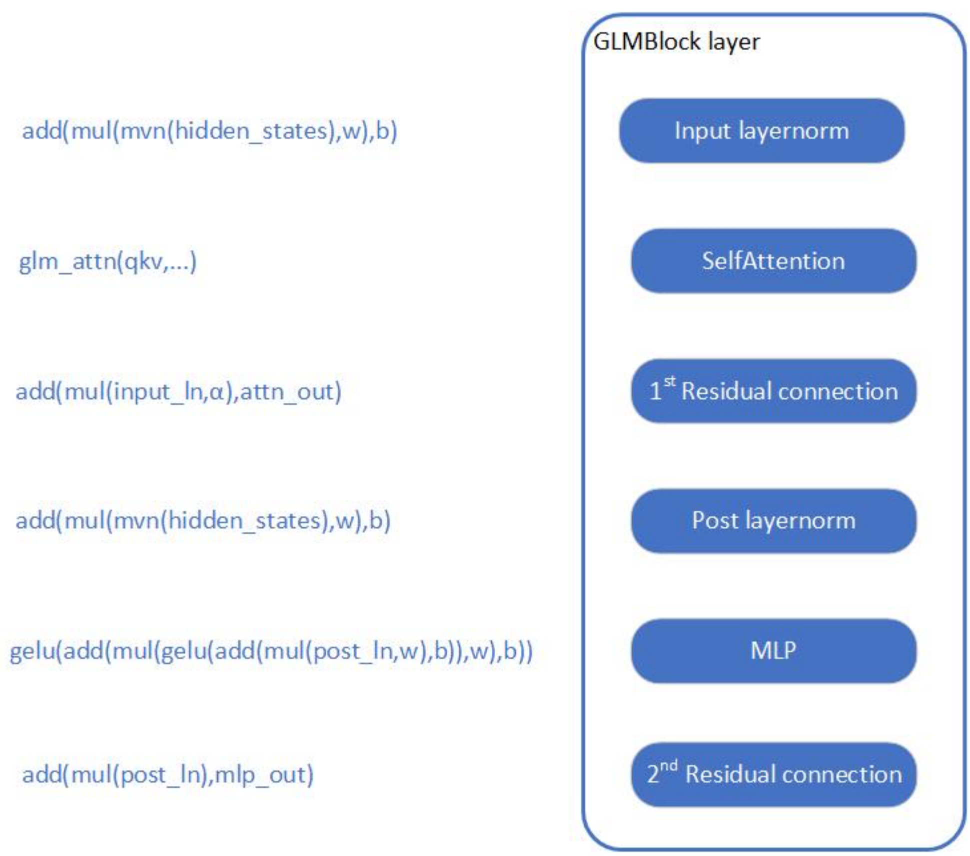


图 2. 为 ChatGLM 构建 OpenVINO™ stateful 模型

关于如何构建 OpenVINO™ stateful 模型，可参阅以下文档：

https://docs.openvino.ai/2022.3/openvino_docs_OV_UG_network_state_intro.html

OpenVINO™ 还提供了模型创建样本，以展示如何通过 opset 构建模型。

https://github.com/openvinotoolkit/openvino/blob/master/samples/cpp/model_creation_sample/main.cpp

ChatGLM 的自定义注意力机制是本文所关注和优化的部分。主要思路是：构建全局上下文结构体，用于在模型内部追加并保存每一轮迭代后的 pastKV 的结果，这样减少了 pastKV 作为模型输入输出的拷贝开销，同时使用内联优化以实现 Rotary Embedding 和多头注意力机制 (Multi-Head Attentions)。

英特尔® AMX 是内置在第四代英特尔® 至强® 可扩展处理器中的矩阵乘法加速器，能够更快速地处理 bf16 或 int8 数据类型的矩阵乘加运算，通过加速张量处理，显著提高推理和训练性能。借助英特尔® AMX 内联指令（用于加速计算的单指令多操作），实现了对 ChatGLM 模型中 Attention，Rotary Embedding 等算子的高度优化，并且使用 bf16 指令进行乘加操作，在保证浮点指数位精度的同时提高运算效率。

与此同时，本方案还使用 int8 精度来压缩全连接层的权重，在实时计算中将使用 bf16 进行计算。因此，无需通过训练后量化 (PTQ) 或量化感知训练（QAT）对模型进行低精度处理。模型压缩方法可以降低模型存储空间，减少内存带宽的负载，因为计算仍然使用浮点，不会造成溢出，不会对模型精度造成损失。

为 ChatGLM 创建 OpenVINO™ stateful 模型

请依照下方示例配置软硬件环境，并按照以下步骤优化 ChatGLM：

硬件要求

第四代英特尔® 至强® 可扩展处理器（代号 Sapphire Rapids）或其后续的、仍内置英特尔® AMX 的产品

软件验证环境

Ubuntu 22.04.1 LTS

面向 OpenVINO™ Runtime Python API 的 Python 3.10.11

用于构建 OpenVINO™ Runtime 的 GCC 11.3.0

cmake 3.26.4

构建 OpenVINO™ 源码

- 安装系统依赖并设置环境
- 创建并启用 Python 虚拟环境

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

- 安装 Python 依赖

```
$ pip install protobuf transformers==4.30.2 cpm_kernels torch>=2.0 sentencepiece pandas
```

- 使用 GCC 11.3.0 编译 OpenVINO™
- 克隆 OpenVINO™ 并升级子模块

```
$ git clone https://github.com/luo-cheng2021/openvino.git -b luocheng/chatglm_custom
$ cd openvino && git submodule update --init --recursive
```

- 安装 Python 环境依赖，以构建 Python Wheel

```
$ python -m pip install -U pip
```

```
$ python -m pip install -r ./src/bindings/python/src/compatibility/openvino/requirements-dev.txt
$ python -m pip install -r ./src/bindings/python/wheel/requirements-dev.txt
```

- 创建编译目录

```
$ mkdir build && cd build
```

- 使用 CMake 编译 OpenVINO™

```
$ cmake .. -DENABLE_LLMDNN=ON \
  -DBUILD_PYTHON_TESTS=ON \
  -DENABLE_CPU_DEBUG_CAPS=OFF \
  -DENABLE_DEBUG_CAPS=OFF  \
  -DCMAKE_BUILD_TYPE=Release \
  -DENABLE_INTEL_MYRIAD_COMMON=OFF \
  -DENABLE_INTEL_GNA=OFF \
  -DENABLE_OPENCV=OFF \
  -DENABLE_CPPLINT=ON \
  -DENABLE_CPPLINT_REPORT=OFF \
  -DENABLE_NCC_STYLE=OFF \
  -DENABLE_TESTS=ON \
  -DENABLE_OV_CORE_UNIT_TESTS=OFF \
  -DENABLE_INTEL_CPU=ON \
  -DENABLE_INTEL_GPU=OFF \
  -DENABLE_AUTO=OFF \
  -DENABLE_AUTO_BATCH=OFF \
  -DENABLE_MULTI=OFF \
  -DENABLE_HETERO=OFF \
  -DENABLE_INTEL_GNA=OFF \
  -DENABLE_PROFILING_ITT=ON\
  -DENABLE_SAMPLES=ON \
  -DENABLE_PYTHON=ON \
  -DENABLE_TEMPLATE=OFF  \
  -DENABLE_OV_ONNX_FRONTEND=OFF \
  -DENABLE_OV_PADDLE_FRONTEND=OFF \
  -DENABLE_OV_PYTORCH_FRONTEND=OFF \
  -DENABLE_OV_TF_FRONTEND=OFF \
  -DENABLE_OPENVINO_DEBUG=OFF \
  -DENABLE_CPU_DEBUG_CAPS=ON \
  -DCMAKE_INSTALL_PREFIX=`pwd`/install \
```

```
 -
DCMAKE_INSTALL_RPATH=`pwd`/install/runtime/3rdparty/tbb/lib:`pwd`/install/runtime/3rdparty/hddl/lib:`pwd`/install/runtime/li
b/intel64 \
 -Dgflags_Dir=`pwd`/../thirdparty/gflags/gflags/cmake
$ make --jobs=$(nproc --all)
$ make install
```

- 安装针对 OpenVINO™ Runtime 和 openvino-dev 工具构建好的 Python Wheel

```
$ pip install ./install/tools/openvino*.whl
```

- 检查系统 GCC 版本和 Conda Runtime GCC 版本。如下所示，如果系统 GCC 版本高于 Conda GCC 版本，请升级 Conda GCC 至相同版本，以满足 OpenVINO™ Runtime 的需求。（可选）

```
##check system (OpenVINO compiling env) gcc version
$ gcc --version
gcc (Ubuntu 11.3.0-1ubuntu1~22.04.1) 11.3.0
```

```
##check conda python (runtime env for OpenVINO later) gcc version
$ python
Python 3.10.11 (main, May 16 2023, 00:28:57) [GCC 11.2.0] on linux
```

```
##If sys gcc ver > conda gcc ver, upgrade conda gcc ver -> sys gcc ver
$ conda install -c conda-forge gcc=11.3.0
```

- 将 PyTorch 模型转为 OpenVINO™ IR

```
$ cd ..
$ python tools/gpt/gen_chatglm.py /path/to/pytorch/model /path/to/ov/IR
```

使用 OpenVINO™ Runtime API 为 ChatGLM 构建推理流水线

本文提供了使用 Transformer 和 OpenVINO™ Runtime API 构建推理流水线的样本。首先，在 *test_chatglm.py* 中，创建一个由 *transformers.PreTrainedModel* 衍生的新类。然后，通过使用 OpenVINO™ Runtime Python API 构建模型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*，如此一来，即可确保输入准备工作、*set_random_seed*、分词器/连接器 (tokenizer/detokenizer) 以及余下的流水线操作能够与原始模型的源码保持一致。

如需启用 int8 权重压缩，只需设置简单的环境变量 USE_INT8_WEIGHT=1。这是因为在模型生成阶段，已使用 int8 对全连接层的权重进行了压缩，因此模型可在之后的运行过程中直接使用 int8 权重进行推理，从而免除了通过框架或量化工具压缩模型的步骤。

请按照以下步骤使用 OpenVINO™ Runtime 流水线测试 ChatGLM：

- 运行 bf16 模型

```
$ python3 tools/gpt/test_chatglm.py /path/to/pytorch/model /path/to/ov/IR --use=ov
```

- 运行 int8 模型

```
$ USE_INT8_WEIGHT=1 python test_chatglm.py /path/to/pytorch/model /path/to/ov/IR --use=ov
```

权重压缩：降低内存带宽使用率，提升推理速度

本文采用了 Vtune 对模型权重数值精度分别为 bf16 和 int8 的内存带宽使用率（图 3 和图 4）以及 CPI 率进行了性能对比分析（表 1）。结果发现：当模型权重数值精度压缩至 int8 时，可同时降低内存带宽使用率和 CPI 率。

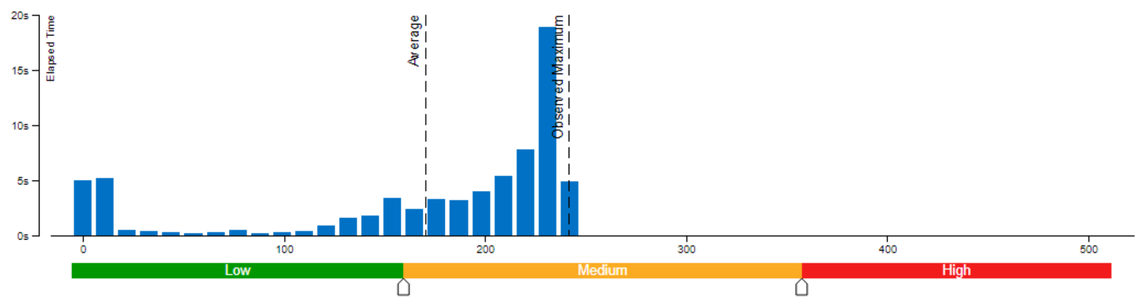


图 3. 模型权重数值精度为 bf16 时的内存带宽使用率

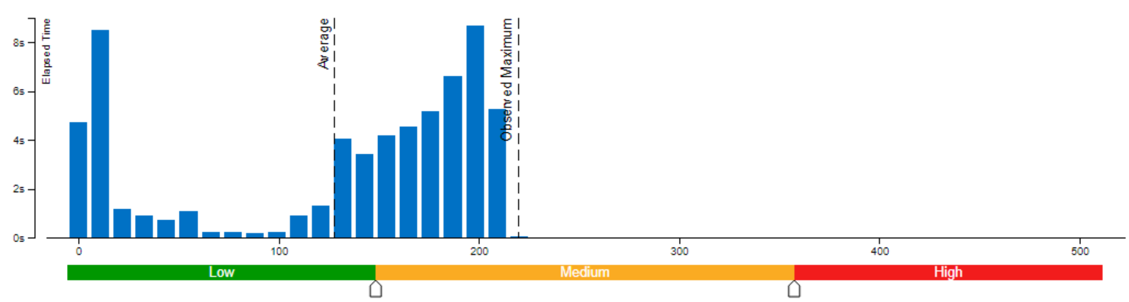


图 4. 模型权重数值精度为 int8 时的内存带宽使用率

模型权重数值精度	bf16	int8
CPI 率	10.766	1.175

表 1.采用不同模型权重数值精度时的 CPI 率

每条指令消耗的时钟周期 (Clockticks per Instruction Retired, CPI) 事件率，也称为“平均指令周期数 (Cycles per Instruction)”，是基于硬件事件抽样收集的基础性能指标之一，在抽样模式下也称为“性能监控计数器 (PMC) 分析”。该比率计算方式为：用处于非停机状态的处理器时钟周期数 (Clockticks) 除以已消耗指令数。每个处理器用于计算时钟周期数和已消耗指令数的确切事件可能并不相同，但 VTune Profiler 可辨别和使用正确的数量。

CPI < 1 时，通常为采用指令密集型代码的应用，而 CPI > 1 则可能是停滞时钟周期密集型应用，也可能是内存密集型应用。由此，我们可以得出结论，类似 chatGLM 等语言模型对内存带宽的要求非常高，性能往往受到内存操作或带宽的限制。很多场景下，消除内存操作的负载，性能会因此获得大幅收益。在优化此类模型时，如何在不影响精度的同时对模型进行压缩或轻量化处理是一项不可或缺的技巧。除此之外，在异构平台和框架上进行部署，还涉及到减少内存/设备存储之间的数据搬运等优化思路。因此，在压缩模型的同时，还需要考虑对原始 pytorch 模型推理 forward/generates 等函数流水线的优化，而 OpenVINO™ 在优化模型自身的同时，还将流水线的优化思路体现在修改模型结构中（将 KV cache 保存在模型内部），通过优化 Optimum-intel 等框架的流水线，减少内存拷贝和数据搬运。

结论

笔者根据上述方法重新设计执行图并优化了 GLMBlock，消除了 ChatGLM 模型输入和输出之间的内存副本，且模型运行高效。随着 OpenVINO™ 的不断升级，本方案的优化工作也将得到推广并集成至正式发布的版本中。这将有助于扩展更多的大语言模型用例。敬请参考 OpenVINO™ 官方版本[iii] 和 Optimum-intel OpenVINO™ 后端[iv]，获取有关大语言模型的官方高效支持。

作者简介:

英特尔® OpenVINO™ 开发工具客户支持工程师赵桢和邹文艺，英特尔® OpenVINO™ 开发工具 AI 框架工程师罗成和李亭骞，都在从事 AI 软件工具开发与优化工作。

---

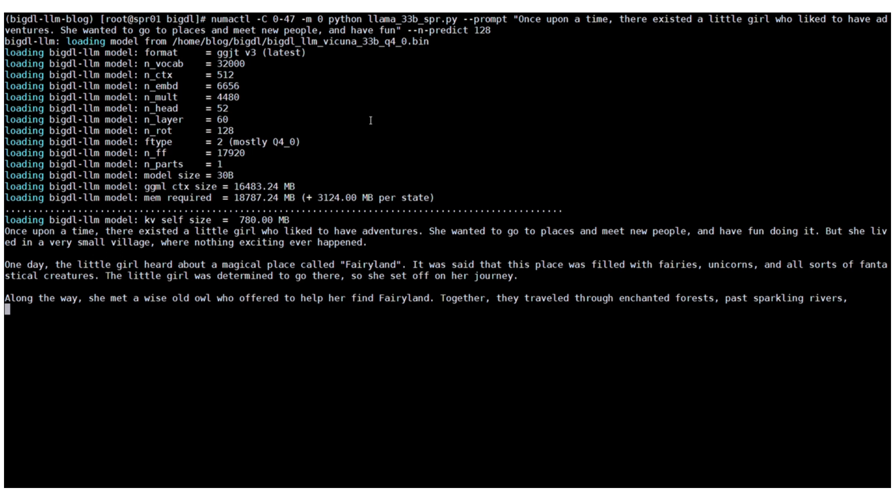
[i] https://www.intel.cn/content/www/cn/zh/events/accelerate-with-xeon.html
[ii] https://huggingface.co/THUDM/chatglm-6b/blob/main/config.json
[iii] https://www.intel.cn/content/www/cn/zh/developer/tools/openvino-toolkit/overview.html
[iv] https://huggingface.co/docs/optimum/main/en/intel/index

BigDL-LLM：在英特尔® 平台上加速大语言模型的便捷新利器

作者：英特尔公司黄晟盛、黄凯、戴金权

我们正迈入一个由大语言模型 (Large Language Model, LLM) 驱动的 AI 新时代，LLM 在诸如客户服务、虚拟助理、内容创作、编程辅助等各类应用中正发挥着越来越重要的作用。然而，随着 LLM 规模不断扩大，运行大模型所需的资源消耗也越来越大，导致其运行也越来越慢，这给 AI 应用开发者带来了相当大的挑战。为此，英特尔最近推出了一个名为 BigDL-LLM[i] 的大模型开源库，可助力 AI 开发者和研究者在英特尔® 平台上加速优化大语言模型，提升大语言模型在英特尔® 平台上的使用体验。

下面就展示了使用 BigDL-LLM 加速过的 330 亿参数的大语言模型 Vicuna-33b-v1.3[ii] 在一台搭载英特尔® 至强® 铂金 8468 处理器的服务器上运行的实时效果。



视频：在一台搭载英特尔® 至强® 铂金 8468 处理器的服务器上运行 330 亿参数大语言模型的实际速度（实时录屏）

BigDL-LLM：英特尔® 平台上的开源大语言模型加速库

BigDL-LLM 是一个针对大语言模型的优化加速库，是开源 BigDL 的一部分，通过 Apache 2.0 许可证发布。它提供了各种低精度优化（例如 INT4/INT5/INT8），并可利用多种英特尔® CPU 集成的硬件加速技术（AVX/VNNI/AMX 等）和最新的软件优化，来赋能大语言模型在英特尔® 平台上实现更高效的优化和更为快速的运行。

BigDL-LLM 的一大重要特性是：对基于 Hugging Face Transformers API 的模型，只需改动一行代码即可对模型进行加速，理论上可以支持运行**任何** Transformers 模型，这对熟悉 Transformers API 的开发者非常友好。除了 Transformers API，很多人也会使用 LangChain 来开发大语言模型应用。为此，BigDL-LLM 也提供便于使用的 LangChain 的集成[iii]，从而让开发者能够轻松使用 BigDL-

LLM 来开发新应用或迁移现有的、基于 Transformers API 或 LangChain API 的应用。此外，对于一般的 PyTorch 大语言模型（没有使用 Transformer 或 LangChain API 的模型），也可使用 BigDL-LLM `optimize_model` API 一键加速来提升性能。详情请参阅 GitHub README[iv] 以及官方文档[v]。

BigDL-LLM 还提供了大量常用开源 LLM 的加速样例（e.g. 使用 Transformers API 的 样例[vi] 和使用 LangChain API 的样例[vii]，以及教程（包括配套 jupyter notebooks）[viii]，方便开发者快速上手尝试。

安装和使用：简便的安装过程和易用的 API 接口

安装 BigDL-LLM 非常简便，只需执行如下所示的这一行命令即可。

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

使用 BigDL-LLM 对大模型进行加速也是非常容易的（这里仅用 Transformers 风格 API 进行举例）。使用 BigDL-LLM Transformer 风格 API 对模型加速，只需要改动模型加载部分，后续使用过程与原生 Transformers 完全一致。而用 BigDL-LLM API 加载模型的方式与 Transformers API 也几乎一致——用户只需要更改 import，在 from_pretrained 参数中设置 load_in_4bit=True 即可。BigDL-LLM 会在加载模型的过程中对模型进行 4-bit 低精度量化，并在后续推理过程中利用各种软硬件加速技术优化其执行。

```
#Load Hugging Face Transformers model with INT4 optimizations
from bigdl.llm. transformers import AutoModelForCausalLM
model = AutoModelForCausalLM.from_pretrained('/path/to/model/', load_in_4bit=True)
```

示例：快速实现一个基于大语言模型的语音助手应用

下文将以 LLM 常见应用场景"语音助手"为例，展示采用 BigDL-LLM 快速实现 LLM 应用的案例。通常情况下，语音助手应用的工作流程分为以下两个部分：

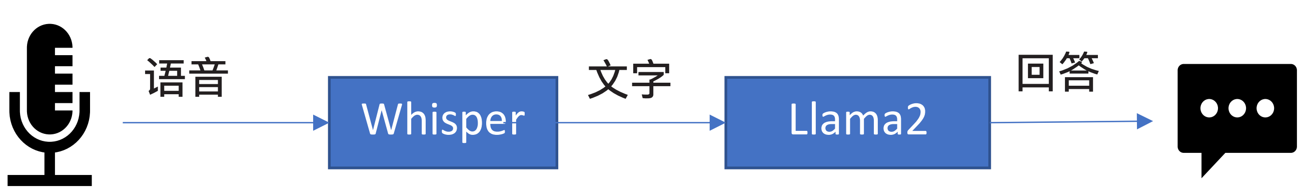


图 1. 语音助手工作流程示意

1、语音识别——使用语音识别模型（本示例采用了 Whisper 模型[ix]）将用户的语音转换为文本；
2、文本生成——将 1 中输出的文本作为提示语(prompt)，使用一个大语言模型（本示例采用了 Llama2[x]）生成回复。

以下是本文使用 BigDL-LLM 和 LangChain[xi] 来搭建语音助手应用的过程：

在**语音识别**阶段：第一步，加载预处理器 processor 和语音识别模型 recog_model。本示例中使用的识别模型 Whisper 是一个 Transformers 模型。只需使用 BigDL-LLM 中的 AutoModelForSpeechSeq2Seq 并设置参数 load_in_4bit=True，就能够以 INT4 精度加载并加速这一模型，从而显著缩短模型推理用时。

```
processor = WhisperProcessor .from_pretrained(recog_model_path)
recog_model = AutoModelForSpeechSeq2Seq .from_pretrained(recog_model_path, load_in_4bit=True)
```

第二步，进行语音识别。首先使用处理器从输入语音中提取输入特征，然后使用识别模型预测 token，并再次使用处理器将 token 解码为自然语言文本。

```
input_features = processor(frame_data,
                           sampling_rate=audio.sample_rate,
                           return_tensor="pt").input_features
predicted_ids = recogn_model.generate(input_features, forced_decoder_ids=forced_decoder_ids)
text = processor.batch_decode(predicted_ids, skip_special_tokens=True)[0]
```

在**文本生成**阶段，首先使用 BigDL-LLM 的 **TransformersLLM API** 创建一个 LangChain 语言模型（TransformersLLM 是在 BigDL-LLM 中定义的语言链 LLM 集成）。您可以使用这个 API 加载任何一个 Hugging Face Transformers 模型。

```
llm = TransformersLLM . from_model_id(
        model_id=llm_model_path,
        model_kwargs={"temperature": 0,
                      "max_length": args.max_length,
                      "trust_remote_code": True},
)
```

然后，创建一个正常的对话链 **LLMChain**，并将已经创建的 llm 设置为输入参数。

```
# The following code is complete the same as the use-case
voiceassistant_chain = LLMChain(
    llm=llm,
    prompt=prompt,
    verbose=True,
    memory=ConversationBufferWindowMemory(k=2),
)
```

这个链条将会记录所有的对话历史，并将对话历史适当地格式化为大语言模型的提示语，用于生成回复。这时候只需要将识别模型生成的文本作为 "human_input" 输入即可，代码如下：

```
response_text = voiceassistant_chain .predict(human_input=text,
                                              stop="\n\n")
```

最后，将语音识别和文本生成步骤放入循环中，即可在多轮对话中与该"语音助手"交谈。您可访问此链接[xii]，查看完整的示例代码，并使用自己的电脑进行尝试。快用 BigDL-LLM 来快速搭建自己的语音助手吧！

作者简介：

英特尔公司 AI 资深架构师黄晟盛，英特尔公司 AI 框架工程师黄凯，英特尔院士、大数据技术全球 CTO、BigDL 项目创始人戴权，都在从事大数据和 AI 相关工作。

---

[i] https://github.com/intel-analytics/BigDL/tree/main/python/llm
[ii] Vicuna 模型是社区基于 LLaMA 模型微调而得的。https://huggingface.co/lmsys/vicuna-33b-v1.3
[iii] https://github.com/intel-analytics/BigDL/blob/main/python/llm/README.md#langchain-api
[iv] https://github.com/intel-analytics/BigDL/blob/main/python/llm/README.md
[v] https://bigdl.readthedocs.io/en/latest/doc/LLM/index.html
[vi] https://github.com/intel-analytics/BigDL/tree/main/python/llm/example/transformers/transformers_int4
[vii] https://github.com/intel-analytics/BigDL/tree/main/python/llm/example/langchain
[viii] https://github.com/intel-analytics/bigdl-llm-tutorial
[ix] https://github.com/openai/whisper
[x] https://huggingface.co/meta-llama/Llama-2-7b-chat-hf
[xi] https://github.com/langchain-ai/langchain
[xii] https://github.com/intel-analytics/BigDL/blob/main/python/llm/example/langchain/transformers_int4/voiceassistant.py

英特尔® Extension for Transformer 一石双鸟：让 LLM CPU 推理加速达 40x+ 攻克聊天场景应用难题！

作者：英特尔公司沈海豪、罗屿、孟恒宇、董波、林俊

英特尔® Extension for Transformer 是什么？

英特尔® Extension for Transformers 是英特尔推出的一个创新工具包，可基于英特尔® 架构平台，尤其是第四代英特尔® 至强® 可扩展处理器（代号 Sapphire Rapids，SPR）显著加速基于 Transformer 的大语言模型（Large Language Model, LLM）。其主要特性包括：

- 通过扩展 Hugging Face transformers API 和利用英特尔® Neural Compressor，为用户提供无缝的模型压缩体验；
- 提供采用低位量化内核（NeurIPS 2023：在 CPU 上实现高效 LLM 推理）的 LLM 推理运行时，支持 Falcon、LLaMA、MPT、Llama2、BLOOM、OPT、ChatGLM2、GPT-J-6B、Baichuan-13B-Base、Baichuan2-13B-Base、Qwen-7B、Qwen-14B 和 Dolly-v2-3B 等常见的 LLM；
- 先进的压缩感知运行时（NeurIPS 2022：在 CPU 上实现快速蒸馏和 QuaLA-MiniLM：量化长度自适应 MiniLM；NeurIPS 2021：一次剪枝，一劳永逸：对预训练语言模型进行稀疏/剪枝）。

本文将重点介绍其中的 LLM 推理运行时（简称为"LLM 运行时"），以及如何利用基于 Transformer 的 API 在英特尔® 至强® 可扩展处理器上实现更高效的 LLM 推理和如何应对 LLM 在聊天场景中的应用难题。

LLM 运行时（LLM Runtime）

英特尔® Extension for Transformers 提供的 LLM Runtime 是一种轻量级但高效的 LLM 推理运行时，其灵感源于 GGML，且与 llama.cpp 兼容，具有如下特性：

- 内核已针对英特尔® 至强® CPU 内置的多种 AI 加速技术（如 AMX、VNNI），以及 AVX512F 和 AVX2 指令集进行了优化；
- 可提供更多量化选择，例如：不同的粒度（按通道或按组）、不同的组大小（如：32/128）；
- 拥有更优的 KV 缓存访问以及内存分配策略；
- 具备张量并行化功能，可助力在多路系统中进行分布式推理。

LLM Runtime 的简化架构图如下：

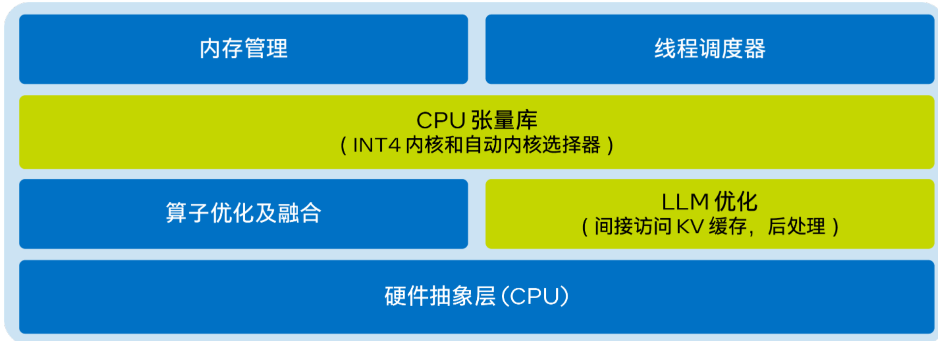


图 1. 英特尔® Extension for Transformers 的 LLM Runtime 简化架构图

使用基于 Transformer 的 API，在 CPU 上实现 LLM 高效推理

只需不到 9 行代码，即可让您在 CPU 上实现更出色的 LLM 推理性能。用户可以轻松地启用与 Transformer 类似的 API 来进行量化和推理。只需将 'load_in_4bit' 设为 true，然后从 HuggingFace URL 或本地路径输入模型即可。下方提供了启用仅限权重的（weight-only）INT4 量化的示例代码：

```
from transformers import AutoTokenizer, TextStreamer
from intel_extension_for_transformers.transformers import AutoModelForCausalLM
model_name = "Intel/neural-chat-7b-v3-1”
prompt = "Once upon a time, there existed a little girl,"

tokenizer = AutoTokenizer.from_pretrained(model_name, trust_remote_code=True)
inputs = tokenizer(prompt, return_tensors="pt").input_ids
streamer = TextStreamer(tokenizer)

model = AutoModelForCausalLM.from_pretrained(model_name, load_in_4bit=True)
outputs = model.generate(inputs, streamer=streamer, max_new_tokens=300)
```

默认设置为：将权重存储为 4 位，以 8 位进行计算。但也支持不同计算数据类型（dtype）和权重数据类型组合，用户可以按需修改设置。下方提供了如何使用这一功能的示例代码：

```
from transformers import AutoTokenizer, TextStreamer
from intel_extension_for_transformers.transformers import AutoModelForCausalLM, WeightOnlyQuantConfig
model_name = "Intel/neural-chat-7b-v3-1”
prompt = "Once upon a time, there existed a little girl,"

woq_config = WeightOnlyQuantConfig(compute_dtype="int8", weight_dtype="int4")
tokenizer = AutoTokenizer.from_pretrained(model_name, trust_remote_code=True)
inputs = tokenizer(prompt, return_tensors="pt").input_ids
streamer = TextStreamer(tokenizer)

model = AutoModelForCausalLM.from_pretrained(model_name,quantization_config=woq_config)
outputs = model.generate(inputs, streamer=streamer, max_new_tokens=300)
```

性能测试

经过持续努力，上述优化方案的 INT4 性能得到了显著提升。本文在搭载英特尔® 至强® 铂金 8480+ 的系统上与 llama.cpp 进行了性能比较；系统配置详情如下：@3.8 GHz，56 核/路，启用超线程，启用睿频，总内存 256 GB (16 x 16 GB DDR5 4800 MT/s [4800 MT/s])，BIOS 3A14.TEL2P1，微代码 0x2b0001b0，CentOS Stream 8。

当输入大小为 32、输出大小为 32、beam 为 1 时的推理性能测试结果，详见下表：

模型	首个 token 时延 (ms)	下一个 token 时延 (ms)	llama.cpp 首个 token (ms)	llama.cpp 下一个 token (ms)
GPT-J-6B	53.17	19.98	546.33	28.4
LLAMA2-7B-Chat	55.52	21.96	344.49	32.19
LLAMA-7B	56.73	22.04	324.85	31.55
MPT-7B	61.8	21.05	321.31	31.55
Falcon-7B	49.44	22.26	344.7	31.2
GPT-NeoX-20B	179.08	61.21	1361.19	100.03
ChatGLM-6B	61.89	22.28	423.8	31.09
ChatGLM2-6B	56.97	21.69	484.1	26.83
Mistral-7B	65.88	25.03	n/a	n/a
Starcoder-3b	126.94	16.91	217.07	33.71
Dolly-v2-3b	57.9	13.41	907.44	23.1
Baichuan-13B	143.8	43.28	842.63	74.06
Baichuan2-13B	129.65	41.64	831.89	69.22
Qwen-7B	135.53	26.87	n/a	n/a
Qwen-14B	115.31	45.87	n/a	n/a

表 1. LLM Runtime 与 llama.cpp 推理性能比较（输入大小 = 32，输出大小 = 32， beam = 1）

输入大小为 1024、输出大小为 32、beam 为 1 时的推理性能的测试结果，详见下表：

模型	首个 token 时延 (ms)	下一个 token 时延 (ms)	llama.cpp 首个 token (ms)	llama.cpp 下一个 token (ms)
GPT-J-6B	685.42	21.72	10953.2	31.4
LLAMA2-7B-Chat	857.86	24.46	11957.3	35.9
LLAMA-7B	848.21	24.26	12239.28	36.92
MPT-7B	1144.15	23.04	27676.7	61.92
Falcon-7B	929.32	27.19	14305.8	34.67
GPT-NeoX-20B	3637.11	65.45	33850.3	85.4
ChatGLM-6B	2897.44	27.38	12315.4	35.113
ChatGLM2-6B	929.97	23.88	10684	29.431
Mistral-7B	790.21	30.69	n/a	n/a
Starcoder-3b	2648.3	22.35	n/a	n/a
Dolly-v2-3b	899.98	15.2	9653.8	26
Baichuan-13B	1426	45.35	58088	89.94
Baichuan2-13B	1216	47.78	44567	102.69
Qwen-7B	1395.04	30.04	n/a	n/a
Qwen-14B	1827.66	48.98	n/a	n/a

表 2. LLM Runtime 与 llama.cpp 推理性能比较（输入大小 = 1024，输出大小 = 32， beam = 1）

根据上表 2 可见：与同样运行在第四代英特尔® 至强® 可扩展处理器上的 llama.cpp 相比，无论是首个 token 还是下一个 token，LLM Runtime 都能显著降低时延，且首个 token 和下一个 token 的推理速度分别提升多达 40 倍[1]（Baichuan-13B，输入为 1024）和 2.68 倍（MPT-7B，输入为 1024）。llama.cpp 的测试采用的是默认代码库。

而综合表 1 和表 2 的测试结果，可得：与同样运行在第四代英特尔® 至强® 可扩展处理器上的 llama.cpp 相比，LLM Runtime 能显著提升诸多常见 LLM 的整体性能：在输入大小为 1024 时，实现 3.58 到 21.5 倍的提升；在输入大小为 32 时，实现 1.76 到 3.43 倍的提升[3]。

准确性测试

英特尔® Extension for Transformers 可利用英特尔® Neural Compressor 中的 SignRound、RTN 和 GPTQ 等量化方法，并使用 lambada_openai、piqa、winogrande 和 hellaswag 数据集验证了 INT4 推理准确性。下表是测试结果平均值与 FP32 准确性的比较。

模型	方法	平均值		
		FP32	int4	
		ACC	ACC	比率
EleutherAI/gpt-j-6b	RTN	0.643375	0.64105	0.996386
decapoda-research/llama-7b-hf	signround	0.688675	0.682925	0.991651
meta-llama/Llama-2-7b-hf	signround	0.69015	0.6868	0.99521
databricks/dolly-v2-3b	GPTQ	0.61345	0.609	0.992746
EleutherAI/gpt-neox-20b	GPTQ	0.673975	0.6687	0.992173
mosaicml/mpt-7b	RTN	0.689	0.683425	0.991909
tiiuae/falcon-7b	GPTQ	0.69815	0.693975	0.99402
baichuan-inc/Baichuan-13B-Base	GPTQ	0.49885	0.49475	0.991781
baichuan-inc/Baichuan2-13B-Base	GPTQ	0.6932	0.68785	0.992282
Qwen-7B	GPTQ	0.683175	0.6726	0.984521

表 3. INT4 与 FP32 准确性对比

从上表 3 可以看出，多个模型基于 LLM Runtime 进行的 INT4 推理准确性损失微小，几乎可以忽略不记。我们验证了很多模型，但由于篇幅限制此处仅罗列了部分内容。如您欲了解更多信息或细节，请访问此链接：https://medium.com/@NeuralCompressor/llm-performance-of-intel-extension-for-transformers-f7d061556176。

更先进的功能：满足 LLM 更多场景应用需求

同时，LLM Runtime 还具备双路 CPU 的张量并行化功能，是较早具备此类功能的产品之一。未来，还会进一步支持双节点。

然而，LLM Runtime 的优势不仅在于其更出色的性能和准确性，我们也投入了大量的精力来增强其在聊天应用场景中的功能，并且解决了 LLM 在聊天场景中可能会遇到的以下应用难题:

1. 对话不仅关乎 LLM 推理，对话历史也很有用。
2. 输出长度有限：LLM 模型预训练主要基于有限的序列长度。因此，当序列长度超出预训练时使用的注意力窗口大小时，其准确性便会降低。
3. 效率低下：在解码阶段，基于 Transformer 的 LLM 会存储所有先前生成的 token 的键值状态（KV），从而导致内存使用过度，解码时延增加。

关于第一个问题，LLM Runtime 的对话功能通过纳入更多对话历史数据以及生成更多输出加以解决，而 llama.cpp 目前尚未能很好地应对这一问题。

关于第二和第三个问题，我们将流式 LLM（Steaming LLM）集成到英特尔® Extension for Transformers 中，从而能显著优化内存使用并降低推理时延。

Streaming LLM

与传统 KV 缓存算法不同，我们的方法结合了***注意力汇聚（Attention Sink）(4 个初始 token)***以提升注意力计算的稳定性，并借助***滚动 KV 缓存保留最新的 token***，这对语言建模至关重要。该设计具有强大的灵活性，可无缝集成到能够利用旋转位置编码 RoPE 和相对位置编码 ALiBi 的自回归语言模型中。

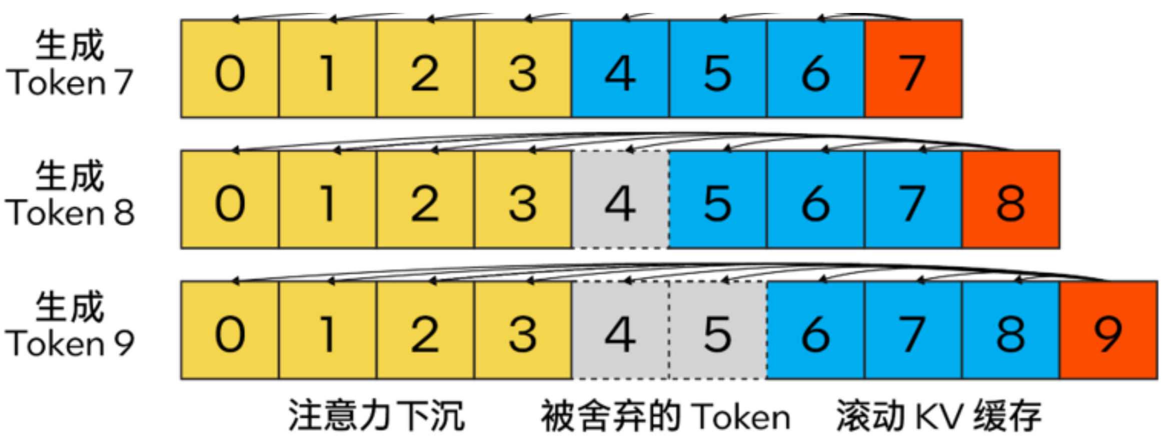


图 2. Steaming LLM 的 KV 缓存（图片来源： 通过注意力下沉实现高效流式语言模型

此外，与 llama.cpp 不同，本优化方案还引入了“*n_keep*”和“*n_discard*”等参数来增强 Streaming LLM 策略。用户可使用前者来指定要在 KV 缓存中保留的 token 数量，并使用后者来确定在已生成的 token 中要舍弃的数量。为了更好地平衡性能和准确性，系统默认在 KV 缓存中舍弃一半的最新 token。

同时，为进一步提高性能，我们还将 Streaming LLM 添加到了 MHA 融合模式中。如果模型是采用旋转位置编码（RoPE)来实现位置嵌入，那么只需针对现有的 K-Cache 应用“移位运算（shift operation)”，即可避免对先前生成的、未被舍弃的 token 进行重复计算。这一方法不仅充分利用了长文本生成时的完整上下文大小，还能在 KV 缓存上下文完全被填满前不产生额外开销。

“shift operation” 依赖于旋转的交换性和关联性，或复数乘法。例如：如果某个 token 的 K-张量初始放置位置为 m 并且旋转了 $m \times \theta_i$ for $i \in [0, d/2)$，那么当它需要移动到 $m-1$ 这个位置时，则可以旋转回到 $(-1) \times \theta_i$ for $i \in [0, d/2)$。这正是每次舍弃 n_discard 个 token 的缓存时发生的事情，而此时剩余的每个 token 都需要“移动” n_discard 个位置。下图以 “n_keep = 4、n_ctx = 16、n_discard = 1” 为例，展示了这一过程。

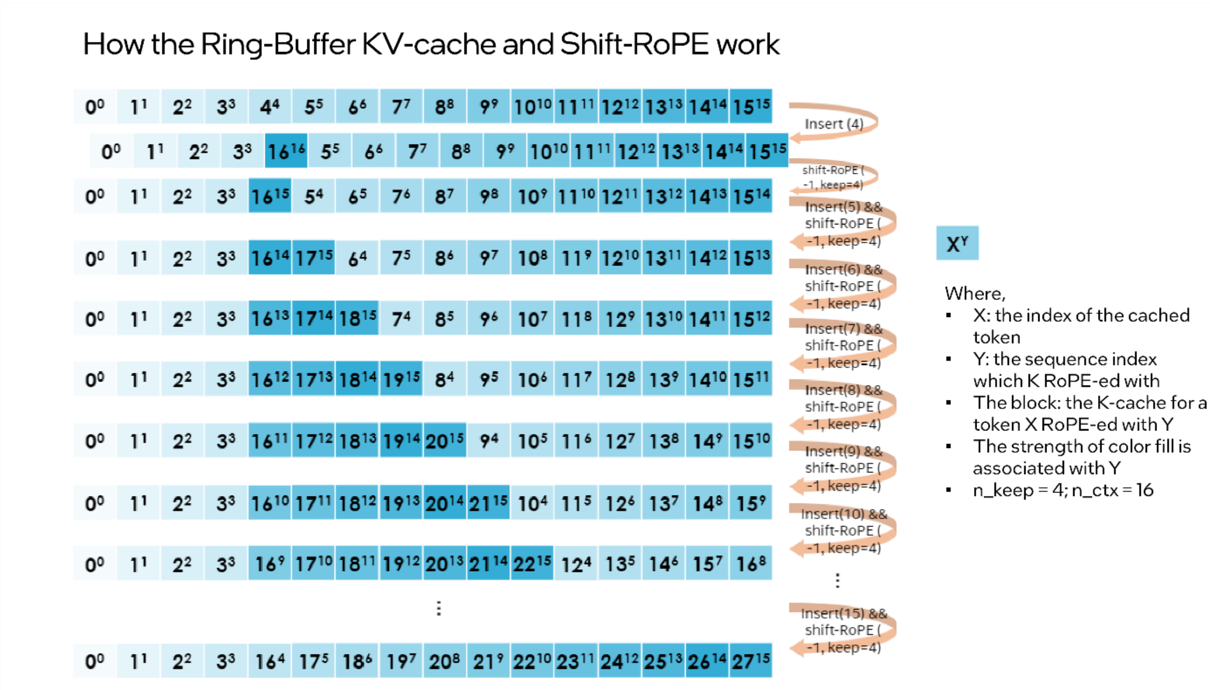


图 3. Ring-Buffer KV-Cache 和 Shift-RoPE 工作原理

需要注意的是：融合注意力层无需了解上述过程。如果对 K-cache 和 V-cache 进行相同的洗牌，注意力层会输出几乎相同的结果（可能存在因浮点误差导致的微小差异）。

您可通过以下代码启动 Streaming LLM:

```
from transformers import AutoTokenizer, TextStreamer
from intel_extension_for_transformers.transformers import AutoModelForCausalLM, WeightOnlyQuantConfig
model_name = "Intel/neural-chat-7b-v1-1"    # Hugging Face model_id or local model
woq_config = WeightOnlyQuantConfig(compute_dtype="int8", weight_dtype="int4")
prompt = "Once upon a time, a little girl"

tokenizer = AutoTokenizer.from_pretrained(model_name, trust_remote_code=True)
inputs = tokenizer(prompt, return_tensors="pt").input_ids
streamer = TextStreamer(tokenizer)

model = AutoModelForCausalLM.from_pretrained(model_name, quantization_config=woq_config, trust_remote_code=True)

# Recommend n_keep=4 to do attention sinks (four initial tokens) and n_discard=-1 to drop half rencetly tokens when meet length threshold

outputs = model.generate(inputs, streamer=streamer, max_new_tokens=300, ctx_size=100, n_keep=4, n_discard=-1)
```

结论与展望

本文基于上述实践经验，提供了一个在英特尔® 至强® 可扩展处理器上实现高效的低位（INT4）LLM 推理的解决方案，并且在一系列常见 LLM 上验证了其通用性以及展现了其相对于其他基于 CPU 的开源解决方案的性能优势。未来，我们还将进一步提升 CPU 张量库和跨节点并行性能。

欢迎您试用 英特尔® Extension for Transformers，并在英特尔® 平台上更高效地运行 LLM 推理！也欢迎您向代码仓库（repository）提交修改请求（pull request）、问题或疑问。期待您的反馈！

特别致谢

在此致谢为此篇文章做出贡献的英特尔公司人工智能资深经理张瀚文及工程师许震中、余振滔、刘振卫、丁艺、王哲、刘宇澄。

[1] 根据表 2 Baichuan-13B 的首个 token 测试结果计算而得。
[2] 根据表 2 MPT-7B 的下一个 token 测试结果计算而得。
[3] 整体性能 = 首个 token 性能 + 31 * 下一个 token 性能。

第四代英特尔® 至强® 可扩展处理器

众多内置加速器，让性能更进一步

在企业和机构寻求扩大规模、降低成本和提供新服务的过程中，通过技术来实现商业价值的重要性日益凸显。面对新的应用场景，他们无需定制系统（这可能会增加复杂性），而是可以通过使用易于扩展的平台来满足现在和未来各种部署的性能需求。

第四代英特尔® 至强® 可扩展处理器旨在为人工智能（AI）、数据分析、存储和科学计算方面快速增长的工作负载提供性能加速。该处理器具备多种内置加速器，帮助人们将零信任安全策略付诸实践，同时利用先进的安全技术，即使面对敏感或受监管的数据，也能解锁新的商业合作机会和洞察。使用这款处理器，您可以跨多个云和边缘环境进行扩展，满足自身的部署需求。英特尔® 至强® 可扩展处理器具有很强的灵活性，可在其上选择不同的云服务，帮助企业顺利实现应用移植。

英特尔几大加速引擎，重新定义性能

重新定义您对处理器的期望。与增加 CPU 内核数相比，内置加速器是一种提升性能更有效的方法。通过内置加速器和软件优化，上一代英特尔® 至强® 可扩展处理器已被证明可以在真实场景下的目标工作负载上实现出色的性能功耗比[1]。这不但可以提高 CPU 利用率，降低功耗，并提高投资回报率（ROI），同时还能帮助企业实现可持续发展目标。

- **英特尔® 高级矩阵扩展（Intel® Advanced Matrix Extensions，英特尔® AMX）**可加速自然语言处理（NLP）、推荐系统和图像识别等深度学习（DL）推理和训练工作负载。
- **英特尔® 数据流加速器（Intel® Data Streaming Accelerator，英特尔® DSA）**可通过优化流数据的传输和转换操作，大幅提升存储、网络和数据密集型工作负载的性能。
- **英特尔® 存内分析加速器（Intel® In-Memory Analytics Accelerator，英特尔® IAA）**可提高数据分析性能，同时从 CPU 内核上卸载任务，为数据库查询及其他工作负载加速。
- **英特尔® 动态负载均衡器（Intel® Dynamic Load Balancer，英特尔® DLB）**可随系统负载的变化将网络数据动态地分配到多个 CPU 内核上，基于硬件高效实现负载平衡。
- **面向 vRAN 的英特尔® 高级矢量扩展（Intel® Advanced Vector Extensions，英特尔® AVX）**在相同功耗范围内可将虚拟无线接入网络（vRAN）的密度较上一代产品提高多达 2 倍[2]。
- **英特尔® 高级矢量扩展 512（Intel® Advanced Vector Extensions 512，英特尔® AVX-512）**支持多达两个融合乘加（FMA）单元，并包含多项优化，可为要求严苛的计算任务提高性能。
- **英特尔® 数据保护与压缩加速技术（英特尔® QAT）**可加速加解密和数据压缩，它通过从处理器内核卸载这些任务，帮助降低系统资源消耗。
- **英特尔® 密码操作硬件加速（Intel® Crypto Acceleration）**降低了实施普遍数据加密的影响，并提高了安全套接字层（SSL）Web 服务器、5G 基础设施和 VPN/防火墙等加密敏感型工作负载的性能。

AI

凭借更优的矢量指令和矩阵乘法运算，第四代英特尔® 至强® 可扩展处理器展现出更为出色的 AI 推理和训练性能。英特尔® AMX 可以显著提高推荐系统、NLP、图像识别、媒体处理和分发以及媒体分析等深度学习工作负载的性能。

科学计算

第四代英特尔® 至强® 可扩展处理器可以提高科学计算工作负载中常见的多线程代码性能。这些工作负载包括制造业仿真、分子动力学、地球系统建模以及 AI 推理和训练。该处理器内置的加速器可提供较高的精度水平，同时还能加速多种 AI 数据类型的处理。该处理器还支持 DDR5 内存、PCIe Gen5、英特尔® 超级通道互联（Intel® Ultra Path Interconnect，英特尔® UPI）2.0 和 Compute Express Link (CXL)，显著提升了整体数据的吞吐量。

数据分析

第四代英特尔® 至强® 可扩展处理器内置的加速器可增强内存数据库、大数据、数据仓库、商业智能 (BI)、企业资源规划 (ERP) 和运营分析的性能。其中，英特尔® DSA 可改善数据处理密集型应用中常见的流数据传输和转换操作，英特尔® IAA 可通过卸载 CPU 内核的任务来提升数据库操作的吞吐量。

网络和存储

英特尔® DLB 可平衡内核间的操作并实现网络数据包优先排序。英特尔® DSA 可卸载数据复制及常见的数据转换操作，从而释放内核周期。这些内置加速器能够实现高效的网络数据分配和企业存储数据传输，并能够改善云计算的内存管理操作，因而使云计算性能更为出色。

加密

英特尔® QAT 现已内置于第四代英特尔® 至强® 可扩展处理器，可为密码操作和压缩操作加速。英特尔® QAT 可以显著提高 CPU 效率和应用吞吐量，同时减少数据占用空间和功耗，赋能企业和机构在不影响性能的情况下强化加密操作。

安全性

英特尔® 软件防护扩展（Intel® Software Guard Extensions，英特尔® SGX）是目前市场上经过深入研究、多次更新和广泛部署的数据中心级机密计算技术，拥有极小的信任边界。

高达 **1.53 倍**
平均性能增益
（与上一代产品的比较）[3]

高达 **10 倍**
PyTorch 实时推理和训练性能提升
这是在启用内置的英特尔® AMX (BF16) 时
与上一代产品 (FP32) 的比较结果[4]

高达 **3 倍**
RocksDB 性能提升
这是在启用内置的英特尔® IAA 时
与上一代产品的比较结果[5]

高达 **1.6 倍**
的 IOPS 提升和高达 37% 的时延降低
这是面对大型数据包顺序读任务，
启用英特尔® DSA 时与上一代产品的比较结果[6]

高达 **2 倍**
容量增益
这是在相同功耗范围内与上一代产品
在 vRAN 工作负载方面的比较结果[2]

高达 **95%**
内核减少幅度以及 2 倍一级压缩吞吐量提升
这是启用内置的英特尔® QAT 时
与上一代产品的比较结果[7]

技术概述

第四代英特尔® 至强® 可扩展处理器采用全新架构，单核性能比上一代产品更高，每路配备多达 60 个内核。每个系统支持单路、双路、四路或八路配置。为了与内核数增加这种情况相匹配，该平台在内存和输入/输出 (I/O) 子系统方面也做了相应改进。DDR5 内存提供的带宽和速度与 DDR4 相比提高多达 1.5 倍，速率达到 4,800 MT/s[8]。此外，该平台还具有每路 80 条 PCIe Gen5 通道的特点，与之前的平台相比， I/O 得到显著提升[9]。本代处理器还可提供 CXL 1.1 连接，支持高网络带宽并使附加加速器能够高效运行。第四代英特尔® 至强® 可扩展处理器支持的技术让您可以根据工作负载要求的变化灵活扩展和调整。此外，本代处理器还可助您实现以下优势:

- 进一步提升网络、存储和计算性能，并通过将繁重的任务卸载到英特尔® 基础设施处理单元 (Intel® Infrastructure Processing Unit，英特尔® IPU) 来提高 CPU 利用率
- 通过英特尔® UPI 2.0 提高多路带宽 (高达 16 GT/s)
- 使用英特尔® Speed Select 技术 (英特尔® SST) 调整 CPU 配置，满足特定工作负载的需求
- 增加三级缓存 (LLC) 共享容量 (所有内核共享多达 100 MB LLC)
- 通过硬件增强型安全功能加强您对安全态势的掌控
- 使用英特尔® Virtual RAID on CPU (英特尔® VROC) ，从而无需再用单独的 RAID 卡

第四代英特尔® 至强® 可扩展处理器的新特性或新功能

PCI Express Gen5 (PCIe 5.0)

带来全新的 I/O 速度，可在 CPU 和互联设备之间实现更高的吞吐量。第四代英特尔® 至强® 可扩展处理器具有多达 80 条 PCIe 5.0 通道，非常适合高速网络、高带宽加速器和高性能存储设备。PCIe 5.0 的 I/O 带宽是 PCIe 4.0 的两倍[9]，仍具备向后兼容性并提供用于 CXL 连接的基础插槽。

DDR5

以更高内存带宽克服数据瓶颈，提高计算性能。与 DDR4 相比，DDR5 的带宽提高多达 1.5 倍[10]，因此有机会提升性能、容量和能效并降低成本。借助 DDR5, 第四代英特尔® 至强® 可扩展处理器提供的速率可高达 4,800 MT/s (1 DPC) 或 4,400 MT/s (2 DPC)。

CXL

借助面向下一代工作负载的 CXL 1.1，降低数据中心的计算时延并帮助减少总体拥有成本 (TCO)。CXL 是另一种跨标准 PCIe 物理层运行的协议，可以在同一链路上同时支持标准 PCIe 设备和 CXL 设备。CXL 可带来的一大关键能力是：在 CPU 和加速器之间创建统一且一致的内存空间，它将革新未来数年数据中心服务器架构的构建方式。

以丰富的选择和出色的灵活性实现扩展——英特尔® 至强® 可扩展处理器

从硬件到系统再到软件，英特尔提供值得信赖的技术基石，帮助企业和机构实现一系列不断扩张的业务目标，同时更好地保护数据安全。无论是提供更大的计算密度来降低功耗，还是加速 AI 工作流程，抑或是支持转向云原生架构，英特尔® 至强® 可扩展处理器都可帮助企业和机构解决至关重要的业务挑战，提供更丰富的云选择和更出色的应用可移植性。

第四代英特尔® 至强® 可扩展处理器概述

英特尔® 至强® 铂金 8400 处理器是打造安全且敏捷的混合云数据中心的基石，专为高级数据分析、AI、高密度基础设施和多云工作负载而设计。这些处理器具备更高性能、强大的平台功能和出色的工作负载加速能力。它们具有更出色的基于硬件的安全性和强大的多路处理性能——特定型号的英特尔® 至强® 铂金 8400 处理器支持多达 8 路配置。借助值得信赖且经过硬件增强的数据服务交付以及全新的 I/O 和连接技术，这些处理器在 I/O、内存、存储和网络技术方面均实现提升，因而能够更好地在数据驱动程度日益加深的世界中挖掘可执行洞察。相关提升包括：

- 每个英特尔® 至强® 可扩展处理器具备多达 60 个内核
- 每个处理器有 8 条内存通道，速率高达 4,800 MT/s（1 DPC）
- 英特尔® AMX 支持下的 AI 加速带来深度学习推理和训练性能的巨大飞跃

英特尔® 至强® 金牌 6400 和英特尔® 至强® 金牌 5400 处理器最多可扩展到四路[11]，并已针对要求严苛的主流数据中心、多云计算以及网络和存储工作负载进行了优化。这些处理器支持更快的内存速度和更大的内存容量，带来了更高的性能、更强大的内存功能、硬件增强型安全性和工作负载加速。

英特尔® 至强® 银牌 4400 处理器可提供基本性能、更快的内存速度和更出色的能效，具备入门级数据中心计算、网络和存储所需的硬件增强型性能。

了解更多信息

有关上述处理器如何推动企业业务发展的更多信息，请见 intel.cn/xeonscalable 和 intel.cn/content/www/cn/zh/products/docs/processors/xeon-accelerated/4th-gen-xeon-scalable-processors.html。

PLATINUM

多达 8 路的可扩展性

4 个英特尔® UPI 端口，速率为 16 GT/s

80 条 PCIe 5.0 通道 + CXL

支持 DDR5，速率高达 4,800 MT/s（每通道 1 个 DIMM）或 4,400 MT/s（每通道 2 个 DIMM）

支持英特尔® 傲腾™ 持久内存 300 系列

英特尔® AVX-512（两个 512 位 FMA）

英特尔® 超线程技术和英特尔® 睿频加速技术

英特尔® AMX

英特尔® SST

先进的可靠性、可用性和可维护性（RAS）

英特尔® SGX 最大飞地容量高达 128 GB（在特定型号的 SKU 上最大飞地容量高达 512 GB）

可通过英特尔® QAT、英特尔® DLB、英特尔® DSA 和英特尔® IAA 加速工作负载

GOLD

多达 4 路的可扩展性

3 个英特尔® UPI 端口，速率为 16 GT/s

80 条 PCIe 5.0 通道 + CXL

支持 DDR5，速率高达 4,800 MT/s（每通道 1 个 DIMM）或 4,400 MT/s（每通道 2 个 DIMM）

支持英特尔® 傲腾™ 持久内存 300 系列

英特尔® AVX-512（两个 512 位 FMA）

英特尔® 超线程技术和英特尔® 睿频加速技术

英特尔® 深度学习加速技术和英特尔® AMX

英特尔® SST

先进的 RAS

英特尔® SGX 最大飞地容量高达 128 GB

可通过英特尔® QAT、英特尔® DLB、英特尔® DSA 和英特尔® IAA 加速工作负载

SILVER

多达 2 路的可扩展性

2 个英特尔® UPI 端口，速率为 16 GT/s

80 条 PCIe 5.0 通道 + CXL

支持 DDR5，速率高达 4,800 MT/s（每通道 1 个 DIMM）或 4,400 MT/s（每通道 2 个 DIMM）

英特尔® AVX-512（两个 512 位 FMA）

英特尔® 超线程技术和英特尔® 睿频加速技术

英特尔® 深度学习加速技术和英特尔® AMX

英特尔® SGX 最大飞地容量高达 64 GB

可通过英特尔® QAT、英特尔® DLB、英特尔® DSA 和英特尔® IAA 加速工作负载

[1] 第三代英特尔® 至强® 可扩展处理器与 AMD EPYC 处理器相比。详情请见以下网址的 [126–130]：https://edc.intel.com/content/www/cn/zh/products/performance/benchmarks/processors/（第三代英特尔® 至强® 可扩展处理器）。结果可能不同。
[2] 详情请见以下网址的 [N9]：https://edc.intel.com/content/www/cn/zh/products/performance/benchmarks/processors/（第四代英特尔® 至强® 可扩展处理器）。结果可能不同。
[3] 详情请见以下网址的 [G1]：https://edc.intel.com/content/www/cn/zh/products/performance/benchmarks/processors/（第四代英特尔® 至强® 可扩展处理器）。结果可能不同。
[4] 详情请见以下网址的 [A16]：https://edc.intel.com/content/www/cn/zh/products/performance/benchmarks/processors/（第四代英特尔® 至强® 可扩展处理器）。结果可能不同。
[5] 详情请见以下网址的 [D1]：https://edc.intel.com/content/www/cn/zh/products/performance/benchmarks/processors/（第四代英特尔® 至强® 可扩展处理器）。结果可能不同。
[6] 详情请见以下网址的 [N18]：https://edc.intel.com/content/www/cn/zh/products/performance/benchmarks/processors/（第四代英特尔® 至强® 可扩展处理器）。结果可能不同。
[7] 详情请见以下网址的 [N16]：https://edc.intel.com/content/www/cn/zh/products/performance/benchmarks/processors/（第四代英特尔® 至强® 可扩展处理器）。结果可能不同。
[8] 详情请见以下网址的 [G2]：https://edc.intel.com/content/www/cn/zh/products/performance/benchmarks/processors/（第四代英特尔® 至强® 可扩展处理器）。结果可能不同。
[9] 第四代英特尔® 至强® 可扩展处理器每个 CPU 有 80 条 PCIe 5.0 通道，并支持 Flex Bus/CXL，而第三代英特尔® 至强® 可扩展处理器则是每个 CPU 有 64 条 PCIe 4.0 通道。
[10] 第四代英特尔® 至强® 可扩展处理器具备 8 通道 DDR5，速率高达 4,800 MT/s（1 DPC），而第三代英特尔® 至强® 可扩展处理器配备 8 通道 DDR4，速率为 3,200 MT/s（2 DPC）。
[11] 仅部分英特尔® 至强® 金牌 6400 处理器提供多达四路的可扩展性。

加速器是否可用视 SKU 而定。更多产品详情，请见英特尔® 产品规格页面。

实际性能受使用情况、配置和其他因素的差异影响。更多信息请见 www.intel.cn/PerformanceIndex。

性能测试结果基于配置信息中显示的日期进行的测试，且可能并未反映所有公开可用的安全更新。详情请参阅配置信息披露。没有任何产品或组件是绝对安全的。

具体成本和结果可能不同。

英特尔技术可能需要启用硬件、软件或激活服务。

英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。

本文中的表述是关于未来计划或期望的前瞻性表述。该表述基于当前预期，且存在许多风险和不确定因素可能造成实际结果与本表述明示或默示的结果有实质性不同。详见英特尔在美国证券交易委员会的备案文件，可见于 www.intc.com。

1222/MG/PRW/PDF 请回收利用

借助英特尔® 高级矩阵扩展（英特尔® AMX）加速人工智能（AI）工作负载

英特尔推出的第四代英特尔® 至强® 可扩展处理器及其内置的英特尔® 高级矩阵扩展（Intel® Advanced Matrix Extensions，英特尔® AMX）可进一步提高 AI 功能，实现较上一代产品 3 至 10 倍的推理和训练性能提升[1]。

优化 AI 流水线

从图书和电影推荐系统到驱动大型电商网站的数字化零售软件，再到支持聊天机器人和机器翻译的自然语言处理（NLP）功能，企业可通过在不同场景中应用人工智能（AI）获得诸多收益。AI 真正的价值在于其解析复杂环境和海量数据的特性，以及解决以往难解之题的能力，而这正是不断推进企业革新的关键潜能。据研究显示，到 2025 年，90% 新发布的企业应用版本都将包含嵌入式 AI 功能[2]。

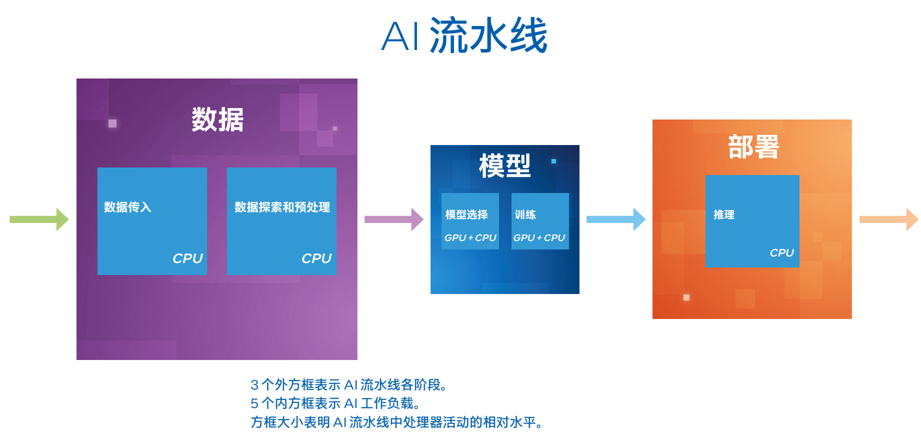


图 1. AI 流水线中的 AI 工作负载和处理器活动

第四代英特尔® 至强® 可扩展处理器内置 AI 加速器——英特尔® AMX，是企业和机构优化 AI 流水线的理想选择。平衡推理是 CPU 在 AI 应用中的主要用例，英特尔® AMX 专为该用例设计并且具备更多训练能力（见图 1）[3]。目前，在所有运行 AI 推理工作负载的已装机数据中心处理单元中，英特尔® 至强® 可扩展处理器的占比高达 70%；因此，为新的 AI 部署选择内置英特尔® AMX 的第四代英特尔® 至强® 可扩展处理器，是一种既高效又具有成本效益的 AI 工作负载加速方式[4]。

选择内置加速器的理由

目前，采用内置英特尔® 深度学习加速技术（Intel® Deep Learning Boost，英特尔® DL Boost）的第三代英特尔® 至强® 可扩展处理器支持 AI 部署，可满足 IT 部门履行客户服务级别协议（SLA）的要求，而内置英特尔® AMX 的第四代英特尔® 至强® 可扩展处理器则将带来新的变革。

图 2 所示为英特尔® AMX 在代际间实现高达 5.7 至 10 倍的 PyTorch 实时推理性能提升的情况；图 3 所示为英特尔® AMX 在代际间实现高达 3.5 至 10 倍的 PyTorch 训练性能提升的情况[5]。凭借更强性能，英特尔® AMX 将进一步提升客户满意度。英特尔® AMX 加速器内置于企业和机构业已熟悉的 CPU 解决方案之中，可显著提升多方面性能，让您在选择适合 AI 应用的 CPU 时更轻松。

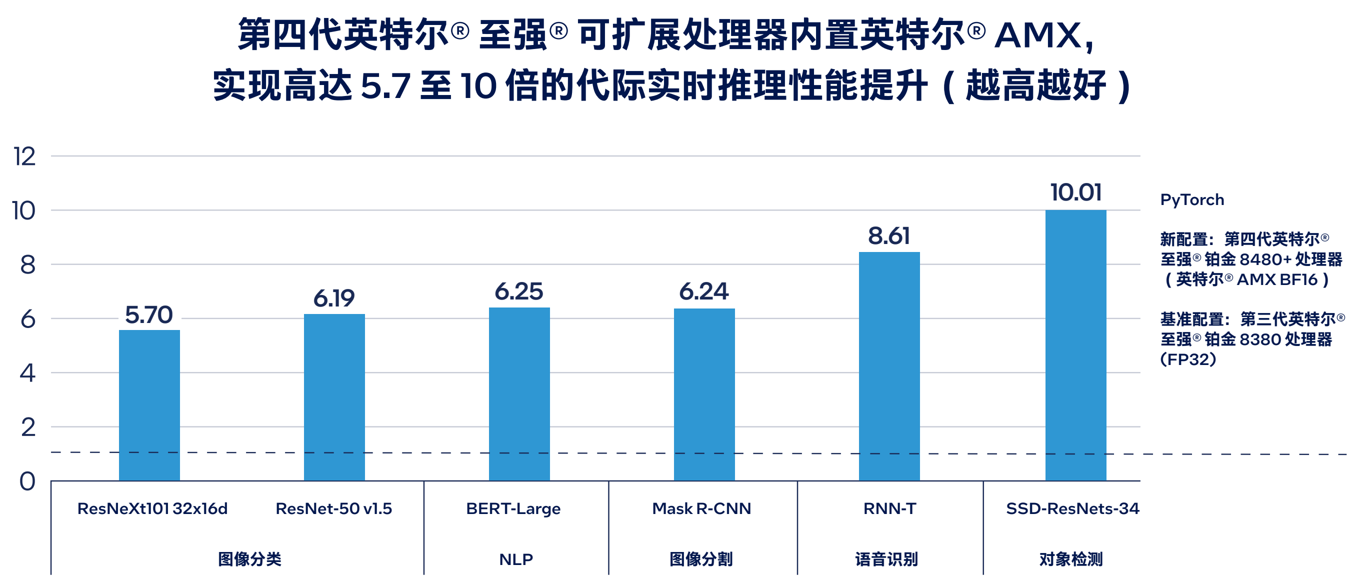


图 2. 第四代英特尔® 至强® 可扩展处理器内置英特尔® AMX，加速 PyTorch 实时推理[5]

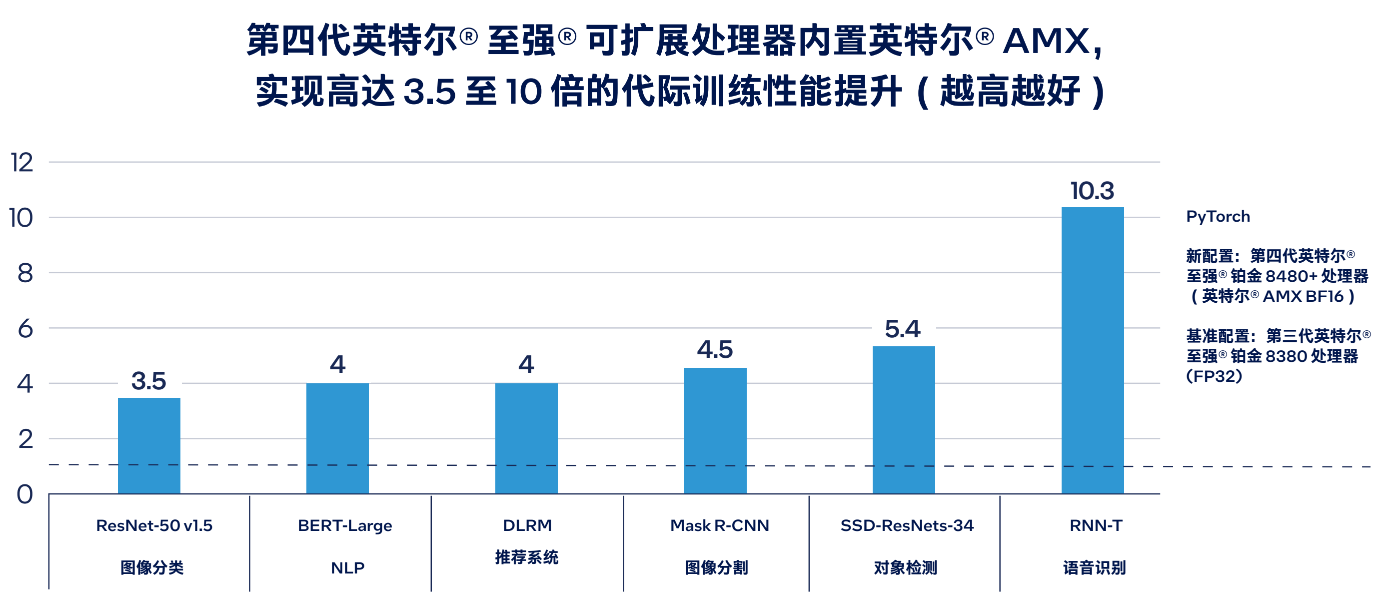


图 3. 第四代英特尔® 至强® 可扩展处理器内置英特尔® AMX，加速 PyTorch 训练[5]

通过图 4 可以看出英特尔® AMX 带来的性能提升远大于每一代产品（从第一代英特尔® 至强® 可扩展处理器开始）通过增加内核所实现的性能提升。

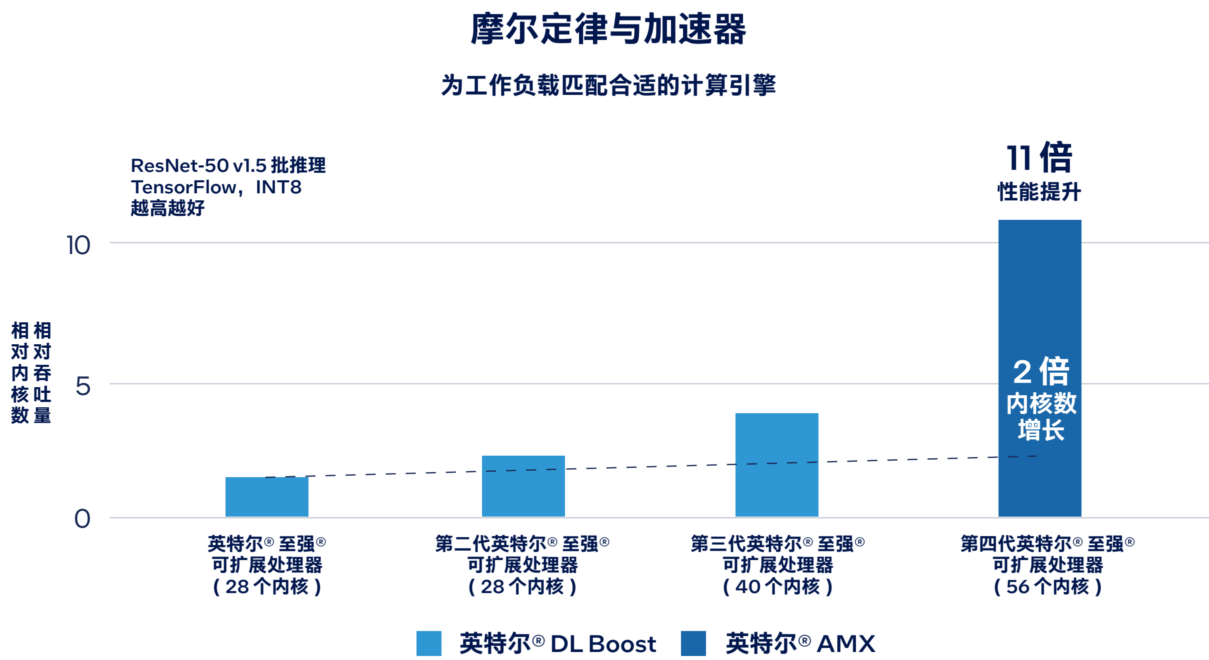


图 4. 以第一代英特尔® 至强® 可扩展处理器为基准，与前几代产品相比，英特尔® AMX 实现非线性性能提升[6]。

英特尔® AMX 是什么？

英特尔® AMX 是内置于第四代英特尔® 至强® 可扩展处理器中的加速器，可优化深度学习（DL）训练和推理工作负载。借助英特尔® AMX，第四代英特尔® 至强® 可扩展处理器可在优化通用计算和 AI 工作负载间快速转换。设想一辆汽车，既有非常出色的城市道路行驶表现，也可以迅速转换模式，展现 F1 赛事级别的竞速性能。第四代英特尔® 至强® 可扩展处理器就具备这种灵活性。开发人员可以编写非 AI 功能代码来利用处理器的指令集架构（ISA），也可编写 AI 功能代码，以充分发挥英特尔® AMX 指令集的优势。英特尔已将其 oneAPI DL 引擎——英特尔® oneAPI 深度神经网络库（Intel® oneAPI Deep Neural Network Library，英特尔® oneDNN）集成至包括 TensorFlow、PyTorch、PaddlePaddle 和 ONNX 在内的多个主流 AI 应用开源工具当中。

英特尔® AMX 架构

英特尔® AMX 架构由两部分组件构成（见图 5）：

- 第一部分为 TILE，由 8 个 1 KB 大小的 2D 寄存器组成，可存储大数据块。
- 第二部分为平铺矩阵乘法（TMUL），它是与 TILE 连接的加速引擎，可执行用于 AI 的矩阵乘法计算。

图 5. 英特尔® AMX 架构由 2D 寄存器文件（TILE）和 TMUL 组成

英特尔® AMX 支持两种数据类型：INT8 和 BF16，两者均可用于 AI 工作负载所需的矩阵乘法运算。

- 当推理无需 FP32（AI 经常使用的单精度浮点格式）的精度时可使用 INT8 这种数据类型。由于该数据类型的精度较低，因此单位计算周期内运算次数就更多。
- BF16 这种数据类型实现的准确度足以达到大多数训练的要求，必要时它也能让 AI 推理实现更高的准确度。

凭借这种新的平铺架构，英特尔® AMX 实现了大幅代际性能提升。与运行英特尔® 高级矢量扩展 512 神经网络指令（Intel® Advanced Vector Extensions 512 Neural Network Instructions，英特尔® AVX-512 VNNI）的第三代英特尔® 至强® 可扩展处理器相比，运行英特尔® AMX 的第四代英特尔® 至强® 可扩展处理器将单位计算周期内执行 INT8 运算的次数从 256 次提高至 2048 次。此外，如图 6 所示，第四代英特尔® 至强® 可扩展处理器可在单位计算周期内执行 1024 次 BF16 运算，而第三代英特尔® 至强® 可扩展处理器执行 FP32 运算的次数仅为 64 次[7]。

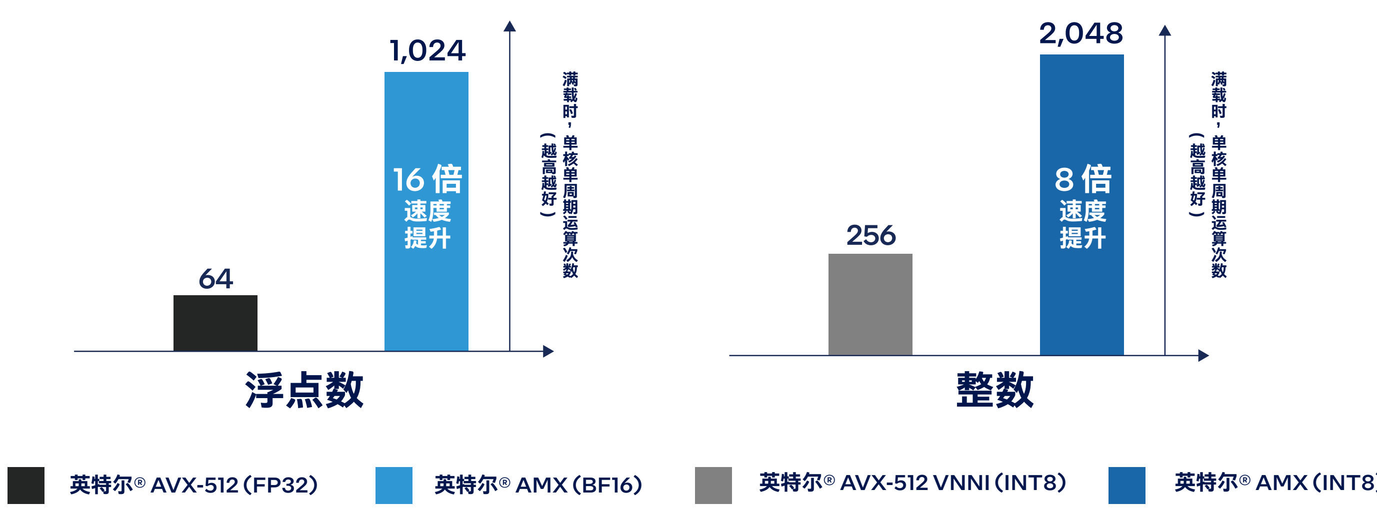


图 6. 与英特尔® AVX-512 VNNI 相比，英特尔® AMX 在处理 INT8 和 BF16 两种数据类型时表现更出色[7]。

AI 用例

内置英特尔® AMX 的第四代英特尔® 至强® 可扩展处理器可用于广泛的深度学习（DL）用例。

推荐系统

无论是推荐电影和图书，还是进行广告精准投放，均可带来个性化的终端用户体验。基于深度学习（DL）创建可将用户实时行为，以及近乎实时的时间和地点等相关场景特征考虑在内的推荐系统。

自然语言处理（NLP）

语言推理、机器学习（ML）等自然语言处理（NLP）应用是企业支持和扩展情绪分析、聊天机器人和机器翻译等多种功能的关键所在，预计到 2029 年，自然语言处理应用的全球市场规模将达到 1,618.1 亿美元[8]。

零售电商软件解决方案

在 PyTorch、TensorFlow 等已面向 AI 优化的框架基础上，通过使用深度学习（DL）推理和训练，大幅缩短交易时间，满足峰值需求，助力零售企业增加收入并提供出色的客户体验。

使用英特尔® AMX 立启新加速

借助英特尔® AMX，几乎无需费力，即可提升性能。这得益于多个默认使用的框架都经过英特尔® oneDNN 的优化。Windows 和 Linux 操作系统、基于内核的虚拟机（KVM）和多个主流虚拟机管理程序都支持英特尔® AMX 指令集。INT8 和 BF16 运算在 TensorFlow 和 PyTorch 等开源框架内可自动优化。开发人员可借助英特尔® 分发版 OpenVINO™ 工具包（Intel® Distribution of OpenVINO™ toolkit）实现 AI 推理的自动化、优化、微调和运行，且几乎或者完全不需要具备编码知识。而且，开发人员只需使用英特尔® Neural Compressor 便可将训练模型量化为 INT8 数据类型。

使用第四代英特尔® 至强® 可扩展处理器为 AI 加速

使用内置英特尔® AMX 的第四代英特尔® 至强® 可扩展处理器，充分挖掘 AI 无限潜能，更好助力千行百业。在数据中心已广泛部署英特尔® 至强® 可扩展处理器的基础之上更进一步，借助全新矩阵乘法运算加速方案，体验更加出色的 AI 训练和推理性能。

有关英特尔® 人工智能和英特尔® AMX 的更多信息，请访问 intel.cn/content/www/cn/zh/artificial-intelligence/overview.html 。

[1] 详情请见以下网址的 [A16, A17]：https://edc.intel.com/content/www/cn/zh/products/performance/benchmarks/processors/（第四代英特尔® 至强® 可扩展处理器）。结果可能不同。

[2] Forbes，“Top Artificial Intelligence (AI) Predictions For 2020 From IDC and Forrester”[IDC 和 Forrester 有关 2020 年人工智能 (AI) 领域的重大预测]，2019 年 11 月，forbes.com/sites/gilpress/2019/11/22/top-artificial-intelligence-ai-predictions-for-2020-from-idc-and-forrester/#4fef9821315a。

[3] The Next Platform，“With AMX, Intel Adds AI/ML Sparkle to Sapphire Rapids”（英特尔凭借 AMX 使第四代英特尔® 至强® 可扩展处理器具备出色的 AI/ML 性能），2021 年 8 月，nextplatform.com/2021/08/19/with-amx-intel-adds-ai-ml-sparkle-to-sapphire-rapids/。

[4] 基于英特尔对截至 2021 年 12 月运行 AI 推理工作负载的全球数据中心服务器装机容量的市场建模。

[5] PyTorch 模型性能配置。**PT-NLP BERT-Large：8480**：单节点，配备 2 个英特尔® 至强® 铂金 8480 处理器的非量产平台，总内存 1,024 GB（16 插槽/64 GB/DDR5-4800），ucode 0x2b0000a1，启用英特尔® 超线程技术（Intel® Hyper-Threading Technology，英特尔® HT 技术），启用英特尔® 睿频加速技术，CentOS Stream 8，5.15.0，1 个 1 TB 英特尔® 固态盘 SC2KW256G8（PT)/Samsung 固态盘 860 EVO（TF)，BERT-Large，推理：SQuAD1.1（序列长度=384），bs=1（4 核/实例），bs=n（单路/实例），bs：FP32=1,56，英特尔® AMX BF16=1,16，英特尔® AMX INT8=1,56，Trg：维基百科 2020 年 1 月 1 日（序列长度=512），bs：FP32=28，英特尔® AMX BF16=56（单实例，单路），框架：https://github.com/intel-innersource/frameworks.ai.pytorch.private-cpu/tree/d7607bdd983093396a70713344828a989b766a66；ModelZoo：https://github.com/IntelAI/models/tree/spr-launch-public，PT：1.13，IPEX：1.13，oneDNN：v2.7，基于英特尔 2022 年 10 月 24 日进行的测试。**8380**：单节点，2 个英特尔® 至强® 铂金 8380 处理器，总内存 1,024 GB（16 插槽/64 GB/DDR4-3200），ucode 0xd000375，启用英特尔® HT 技术，启用英特尔® 睿频加速技术，Ubuntu 22.04 LTS，5.15.0-27-generic，1 个英特尔® 固态盘 SC2KG960G8，BERT-Large，推理：SQuAD1.1（序列长度=384），bs=1（4 核/实例），bs=n（单路/实例），bs：FP32=1,56，INT8=1,56，Trg：维基百科 2020 年 1 月 1 日（序列长度=512），bs：FP32=28，英特尔® AMX BF16=56（单实例，单路），框架：https://github.com/intel-innersource/frameworks.ai.pytorch.private-cpu/tree/d7607bdd983093396a70713344828a989b766a66；ModelZoo：https://github.com/IntelAI/models/tree/spr-launch-public，PT：1.13，IPEX：1.13，oneDNN：v2.7，基于英特尔 2022 年 10 月 24 日进行的测试。**PT-DLRM：8480**：单节点，配备 2 个英特尔® 至强® 铂金 8480 处理器的非量产平台，总内存 1,024 GB（16 插槽/64 GB/DDR5-4800），ucode 0x2b0000a1，启用英特尔® HT 技术，启用英特尔® 睿频加速技术，CentOS Stream 8，5.15.0，1 个 1 TB 英特尔® 固态盘 SC2KW256G8（PT)/Samsung 固态盘 860 EVO（TF)，DLRM，推理：bs=n（单路/实例），bs：FP32=128，英特尔® AMX BF16=128，英特尔® AMX INT8=128，训练 bs：fp32/英特尔® AMX BF16=32K（单实例，单路），Criteo TB 数据集，框架：https://github.com/intel-innersource/frameworks.ai.pytorch.private-cpu/tree/d7607bdd983093396a70713344828a989b766a66；ModelZoo：https://github.com/IntelAI/models/tree/spr-launch-public，PT：1.13，IPEX：1.13，oneDNN：v2.7，基于英特尔 2022 年 10 月 24 日进行的测试。**8380**：单节点，2 个英特尔® 至强® 铂金 8380 处理器，总内存 1,024 GB（16 插槽/64 GB/DDR4-3200），ucode 0xd000375，启用英特尔® HT 技术，启用英特尔® 睿频加速技术，Ubuntu 22.04 LTS，5.15.0-27-generic，1 个英特尔® 固态盘 SC2KG960G8，DLRM，推理：bs=n（单路/实例），bs：FP32=128，INT8=128，训练 bs：FP32=32K（单实例，单路），Criteo TB 数据集，框架：https://github.com/intel-innersource/frameworks.ai.pytorch.private-cpu/tree/d7607bdd983093396a70713344828a989b766a66；ModelZoo：https://github.com/IntelAI/models/tree/spr-launch-public，PT：1.13，IPEX：1.13，oneDNN：v2.7，基于英特尔 2022 年 10 月 24 日进行的测试。**PT-ResNets-34：8480**：单节点，配备 2 个英特尔® 至强® 铂金 8480 处理器的非量产平台，总内存 1,024 GB（16 插槽/64 GB/DDR5-4800），ucode 0x2b0000a1，启用英特尔® HT 技术，启用英特尔® 睿频加速技术，CentOS Stream 8，5.15.0，1 个 1 TB 英特尔® 固态盘 SC2KW256G8（PT)/Samsung 固态盘 860 EVO（TF)，SSD-ResNet-34，推理：bs=1（4 核/实例），bs=n（单路/实例)，bs：FP32=1,112，英特尔® AMX BF16=1,112，英特尔® AMX INT8=1,112，训练 bs：FP32/英特尔® AMX BF16=224（单实例，单路），COCO 2017，框架：https://github.com/intel-innersource/frameworks.ai.pytorch.private-cpu/tree/d7607bdd983093396a70713344828a989b766a66；ModelZoo：https://github.com/IntelAI/models/tree/spr-launch-public，PT：1.13，IPEX：1.13，oneDNN：v2.7，基于英特尔 2022 年 10 月 24 日进行的测试。**8380**：单节点，2 个英特尔® 至强® 铂金 8380 处理器，总内存 1,024 GB（16 插槽/64 GB/DDR4-3200），ucode 0xd000375，启用英特尔® HT 技术，启用英特尔® 睿频加速技术，Ubuntu 22.04 LTS，5.15.0-27-generic，1 个英特尔® 固态盘 SC2KG960G8，SSD--ResNet-34，推理：bs=1（4 核/实例），bs=n（单路/实例），bs：FP32=1,112，INT8=1,112，训练 bs：FP32=224（单实例，单路），COCO 2017，框架：https://github.com/intel-innersource/frameworks.ai.pytorch.private-cpu/tree/d7607bdd983093396a70713344828a989b766a66；ModelZoo：https://github.com/IntelAI/models/tree/spr-launch-public，PT：1.13，IPEX：1.13，oneDNN：v2.7，基于英特尔 2022 年 10 月 24 日进行的测试。**PT-ResNets-50：8480**：单节点，配备 2 个英特尔® 至强® 铂金 8480 处理器的非量产平台，总内存 1,024 GB（16 插槽/64 GB/DDR5-4800），ucode 0x2b0000a1，启用英特尔® HT 技术，启用英特尔® 睿频加速技术，CentOS Stream 8，5.15.0，1 个 1 TB 英特尔® 固态盘 SC2KW256G8 (PT)/Samsung 固态盘 860 EVO (TF)，ResNet-50 v1.5，推理：bs=1（4 核/实例），bs=n（单路/实例），bs：FP32=1,64，英特尔® AMX BF16=1,64，英特尔® AMX INT8=1,116，训练 bs：FP32，英特尔® AMX BF16=128（单实例，单路），ImageNet (224 x 224)，框架：https://github.com/intel-innersource/frameworks.ai.pytorch.private-cpu/tree/d7607bdd983093396a70713344828a989b766a66；ModelZoo：https://github.com/IntelAI/models/tree/spr-launch-public，PT：1.13，IPEX：1.13，oneDNN：v2.7，基于英特尔 2022 年 10 月 24 日进行的测试。**8380**：单节点，2 个英特尔® 至强® 铂金 8380 处理器，总内存 1,024 GB（16 插槽/64 GB/DDR4-3200），ucode 0xd000375，启用英特尔® HT 技术，启用英特尔® 睿频加速技术，Ubuntu 22.04 LTS，5.15.0-27-generic，1 个英特尔® 固态盘 SC2KG960G8，ResNet-50 v1.5，推理：bs=1（4 核/实例），bs=n（单路/实例），bs：FP32=1,64，INT8=1,116，训练 bs: FP32=128（单实例，单路），ImageNet (224 x 224)，框架：https://github.com/intel-innersource/frameworks.ai.pytorch.private-cpu/tree/d7607bdd983093396a70713344828a989b766a66；ModelZoo：https://github.com/IntelAI/models/tree/spr-launch-public，PT：1.13，IPEX：1.13，oneDNN：v2.7，基于英特尔 2022 年 10 月 24 日进行的测试。**PT-RNN-T：8480**：单节点，配备 2 个英特尔® 至强® 铂金 8480 处理器的非量产平台，总内存 1,024 GB（16 插槽/64 GB/DDR5-4800），ucode 0x2b0000a1，启用英特尔® HT 技术，启用英特尔® 睿频加速技术，CentOS Stream 8，5.15.0，1 个 1 TB 英特尔® 固态盘 SC2KW256G8 (PT)/Samsung 固态盘 860 EVO (TF)，Resnext101 32x16d，推理：bs=1（4 核/实例），bs=n（单路/实例），bs：FP32=1,64，英特尔® AMX BF16=1,64，英特尔® AMX INT8=1,116，ImageNet，框架：https://github.com/intel-innersource/frameworks.ai.pytorch.private-cpu/tree/d7607bdd983093396a70713344828a989b766a66；ModelZoo：https://github.com/IntelAI/models/tree/spr-launch-public，PT：1.13，IPEX：1.13，oneDNN：v2.7，基于英特尔 2022 年 10 月 24 日进行的测试。**8380**：单节点，2 个英特尔® 至强® 铂金 8380 处理器，总内存 1,024 GB（16 插槽/64 GB/DDR4-3200），ucode 0xd000375，启用英特尔® HT 技术，启用英特尔® 睿频加速技术，Ubuntu 22.04 LTS，5.15.0-27-generic，1 个英特尔® 固态盘 SC2KG960G8，Resnext101 32x16d，推理：bs=1（4 核/实例），bs=n（单路/实例），bs：FP32=1,64，INT8=1,116，ImageNet，框架：https://github.com/intel-innersource/frameworks.ai.pytorch.private-cpu/tree/d7607bdd983093396a70713344828a989b766a66；ModelZoo：https://github.com/IntelAI/models/tree/spr-launch-public，PT：1.13，IPEX：1.13，oneDNN：v2.7，基于英特尔 2022 年 10 月 24 日进行的测试。**PT-ResNext101：8480**：单节点，配备 2 个英特尔® 至强® 铂金 8480 处理器的非量产平台，总内存 1,024 GB（16 插槽/64 GB/DDR5-4800），ucode 0x2b0000a1，启用英特尔® HT 技术，启用英特尔® 睿频加速技术，CentOS Stream 8，5.15.0，1 个 1 TB 英特尔® 固态盘 SC2KW256G8 (PT)/Samsung 固态盘 860 EVO (TF)，Resnext101 32x16d，bs=n（单路/实例），推理：bs：FP32=1,64，英特尔® AMX BF16=1,64，英特尔® AMX INT8=1,116，框架：https://github.com/intel-innersource/frameworks.ai.pytorch.private-cpu/tree/d7607bdd983093396a70713344828a989b766a66；ModelZoo：https://github.com/IntelAI/models/tree/spr-launch-public，PT：1.13，IPEX：1.13，oneDNN：v2.7，基于英特尔 2022 年 10 月 24 日进行的测试。**8380**：单节点，2 个英特尔® 至强® 铂金 8380 处理器，总内存 1,024 GB（16 插槽/64 GB/DDR4-3200），ucode 0xd000375，启用英特尔® HT 技术，启用英特尔® 睿频加速技术，Ubuntu 22.04 LTS，5.15.0-27-generic，1 个英特尔® 固态盘 SC2KG960G8，Resnext101 32x16d，bs=n（单路/实例），推理：bs：FP32=1,64，INT8=1,116，框架：https://github.com/intel-innersource/frameworks.ai.pytorch.private-cpu/tree/d7607bdd983093396a70713344828a989b766a66；ModelZoo：https://github.com/IntelAI/models/tree/spr-launch-public，PT：1.13，IPEX：1.13，oneDNN：v2.7，基于英特尔 2022 年 10 月 24 日进行的测试。**PT-MaskRCNN：8480**：单节点，配备 2 个英特尔® 至强® 铂金 8480 处理器的非量产平台，总内存 1,024 GB（16 插槽/64 GB/DDR5-4800），ucode 0x2b0000a1，启用英特尔® HT 技术，启用英特尔® 睿频加速技术，CentOS Stream 8，5.15.0，1 个英特尔® 固态盘 SC2KW256G8 (PT)/Samsung 固态盘 860 EVO (TF)，MaskRCNN，推理：bs=1（4 核/实例），bs=n（单路/实例），bs：FP32=1,112，英特尔® AMX BF16=1,112，训练 bs: FP32/英特尔® AMX BF16=112（单实例，单路），COCO 2017，框架：https://github.com/intel-innersource/frameworks.ai.pytorch.private-cpu/tree/d7607bdd983093396a70713344828a989b766a66；ModelZoo：https://github.com/IntelAI/models/tree/spr-launch-public，PT：1.13，IPEX：1.13，oneDNN：v2.7，基于英特尔 2022 年 10 月 24 日进行的测试。**8380**：单节点，2 个英特尔® 至强® 铂金 8380 处理器，总内存 1,024 GB（16 插槽/64 GB/DDR4-3200），ucode 0xd000375，启用英特尔® HT 技术，启用英特尔® 睿频加速技术，Ubuntu 22.04 LTS，5.15.0-27-generic，1 个英特尔® 固态盘 SC2KG960G8，MaskRCNN，推理：bs=1（4 核/实例），bs=n（单路/实例），bs：FP32=1,112，训练 bs：FP32=112（单实例，单路），COCO 2017，框架：https://github.com/intel-innersource/frameworks.ai.pytorch.private-cpu/tree/d7607bdd983093396a70713344828a989b766a66；ModelZoo：https://github.com/IntelAI/models/tree/spr-launch-public，PT：1.13，IPEX：1.13，oneDNN：v2.7，基于英特尔于 2022 年 10 月 24 日进行的测试。推理：ResNet-50 v1.5：ImageNet (224 x 224)，SSD ResNet-34：COCO 2017 (1200 x 1200)，BERT-Large：SQuAD1.1 (序列长度=384)，Resnext101：ImageNet，Mask RCNN：COCO 2017，DLRM：Criteo TB 数据集，RNNT：LibriSpeech。训练：ResNet-50 v1.5：ImageNet (224 x 224)，SSD ResNet-34：COCO 2017，BERT-Large：维基百科 2020 年 1 月 1 日 (序列长度=512)，DLRM：Criteo TB 数据集，RNNT：LibriSpeech，Mask RCNN：COCO 2017.

[6] **用于 INT8 测试的软件配置：**TensorFlow ResNet-50 v1.5，推理：BS=116 (INT8)，单实例/路。oneDNN v2.7，面向英特尔® 架构优化的 TensorFlow 2.10。基于英特尔分别于 2022 年 10 月 24 日（第三代和第四代英特尔® 至强® 可扩展处理器）和 2022 年 7 月 19 日（第二代和第一代英特尔® 至强® 可扩展处理器）进行的测试。**硬件配置：第四代英特尔® 至强® 可扩展处理器硬件配置（已测）**：配备 2 个英特尔® 至强® 铂金 8480 处理器的非量产平台[56 核，350 W 热设计功耗 (TDP)]，1 TB（8 通道/64 GB/4800 MHz）的 DDR5 总内存，使用 BKC 01，使用英特尔® AMX/INT8 和 BF16，CentOS Stream 8，英特尔® AMX 内核 (5.15)，测试结果可能不同。**第三代英特尔® 至强® 可扩展处理器硬件配置（已测）**：单节点，2 个英特尔® 至强® 铂金 8380 处理器（40 核/2.3 GHz，270 W TDP），1 TB（8 插槽/64 GB/3,200 MHz）DDR4 总内存，ucode 0xd0002f2，启用英特尔® HT 技术，启用英特尔® 睿频加速技术，Ubuntu 20.04.2 LTS (Focal Fossa)，5.4.0-73-generic，1 个英特尔® 固态盘 SC2CW480A3 操作系统盘。**第二代英特尔® 至强® 可扩展处理器硬件配置（已测）**：单节点，2 个英特尔® 至强® 铂金 8280 处理器，28 核，英特尔® HT 技术，启用英特尔® 睿频加速技术，384 GB 总内存（12 插槽/32 GB/2,933 MHz)，BIOS：SE5C620.86B.02.01.0013.12152020065（ucode：0x500320a)，CentOS Stream 8，4.18.0-383.el8.x86_64。**英特尔® 至强® 可扩展处理器硬件配置（已测）**：单节点，2 个英特尔® 至强® 铂金 8180 处理器，28 核，英特尔® HT 技术，启用英特尔® 睿频加速技术，384 GB 总内存（12 插槽/32 GB/2,666 MHz)，BIOS：SE5C620.86B.0X.01.0117.021220182317（ucode：0x2006b06)，Ubuntu 20.04.2 LTS，5.4.0-73-generic。

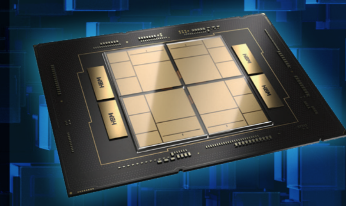
[7] 基于假设 CPU 满载情况下单个内核在单个周期内进行矩阵乘法 + 累加运算的架构峰值处理能力。截至 2021 年 8 月。配合工作负载/配置信息请见 www.intel.cn/PerformanceIndex（前往 Events 项下的 Architecture Day 2021）。结果可能不同。

[8] 有关全球 NLP 市场规模信息的来源：Fortune Business Insights，“Natural Language Processing (NLP) Market Size, Share & COVID-19 Impact Analysis, By Deployment (On-Premises, Cloud, Hybrid), By Enterprise Size (SMEs, and Large Enterprises), By Technology (Interactive Voice Response (IVR), Optical Character Recognition (OCR), Text Analytics, Speech Analytics, Classification and Categorization), By Industry Vertical (Healthcare, Retail, High Tech, and Telecom, BFSI) and Regional Forecast, 2022-2029”{按照部署（本地、云端、混合）、企业规模（中小企业和大型企业）、技术[交互式语音应答 (IVR)、光学字符识别 (OCR)、文本分析、语音分析、分类]、行业垂直领域[医疗，零售，高科技，电信及银行、金融服务和保险 (BFSI)]等类别对 2022 年至 2029 年自然语言处理 (NLP) 市场规模、占有率及新冠影响的分析及区域预测}。2021 年 6 月，fortunebusinessinsights.com/industry-reports/natural-language-processing-nlp-market-101933#。

产品简介

计算加速
系统与图形

英特尔® 至强® CPU Max系列



过去十年，随着人工智能（AI）技术的加入，峰值算力大幅增长，但由于在向内核传输数据时效率低，因此工作负载性能未能同步提升。英特尔® 至强® CPU Max 系列的诞生，使英特尔® 至强® 平台如虎添翼，它是英特尔唯一一个基于 x86 架构并采用高带宽内存的 CPU 系列，可释放和加速内存密集型科学计算和 AI 工作负载。

从天气预报到绘制人类基因组图谱和助力治疗全球范围内的致命疾病，再到设计更节能的材料，科学计算触及到人类生活的方方面面。虽然科学计算和 AI 领域的进步提高了竞争力并将对科学计算的需求带至新的高度，但却没有"放之四海而皆准"的解决方案。传统的科学计算软件种类繁多，如果依据垂直领域和特性对常见的工作负载进行划分，有些属于内存密集型，有些则属于计算密集型；有些需要有大量控制流的小型内核（kernel），有些则需要支持数据并行处理的大型内核；但大多都涉及非常大的数据集。

英特尔® 至强® CPU Max 系列采用高带宽内存（HBM），为英特尔® 至强® 可扩展处理器家族注入新力量，它专为建模、人工智能、深度学习、科学计算和数据分析等数据密集型工作负载设计，专注解锁性能，加速新发现。

更高带宽，更优性能

英特尔® 至强® CPU Max 系列采用全新微架构，支持一系列可提升平台能力的特性，包括更多内核、先进的 I/O 与内存子系统，以及可加速重大发现的内置加速器。英特尔® 至强® CPU Max 系列具有以下特性：

- **多达 56 个 P-core（性能核）**：内核由 4 个小芯片构成，采用英特尔的嵌入式多芯片互连桥接（EMIB）技术连接，功耗为 350 W。
- **64 GB 高带宽封装内存及 PCIe 5.0 和 CXL 1.1 I/O。**英特尔® 至强® CPU Max 系列每核均具备高带宽内存（HBM）容量，可满足大多数常见科学计算工作负载的要求。
- 与其他 CPU 相比，在使用 Numenta 的 AI 技术进行自然语言处理（NLP）时，其高带宽内存（HBM）优势可带来高达 **20 倍的性能**提升[2]。

5 倍
性能提升
（与第三代英特尔® 至强® 可扩展处理器在内存带宽方面的对比）[1]

双路英特尔® 至强® CPU MAX 系列处理器与双路英特尔® 至强® 铂金 8380 处理器相比

加速科学创新

支持快速获得发现、高效开展科研活动。使用英特尔® 至强® CPU Max 系列和第四代英特尔® 至强® 可扩展处理器，您不但可以获得处理要求严苛的工作负载所需的性能与能效，还可得到各种出色的内置加速器的助力。您可以利用面向科学计算和 AI 工作负载的关键加速器，提高 CPU 使用效率、降低功耗、实现更高的投资回报率（ROI）：

- **英特尔® 高级矩阵扩展（Intel® Advanced Matrix Extensions，英特尔® AMX）**：英特尔® AMX 可显著加速基于 CPU 的深度学习推理和训练，从而进一步提升 AI 性能，与英特尔® 高级矢量扩展 512（Intel® Advanced Vector Extensions 512，英特尔® AVX-512）相比，它在进行 INT8/INT32 累加运算时可将峰值吞吐量提高 8 倍[3]。
- **英特尔® 数据流加速器（Intel® Data Streaming Accelerator，英特尔® DSA）**：可通过加速流数据传输，提升数据密集型工作负载的性能。在使用 NVMe over TCP 时，英特尔® DSA 可以将存储 IOPS 提升高达 79%，将时延降低多达 45%[4]。
- **英特尔® 高级矢量扩展 512（Intel® Advanced Vector Extensions 512，英特尔® AVX-512）**：通过矢量化实现性能提升，使更大数据集上的运算速度更快，满足科学仿真、AI/深度学习、3D 建模与分析以及其他密集型工作负载的要求。英特尔® AVX-512 是最新的 x86 矢量指令集，可为要求严苛的计算任务提升性能。
- **I/O 和内存子系统**升级包括：
  - **DDR5**：以更高内存带宽克服数据瓶颈，提高计算性能。与 DDR4 相比，DDR5 的带宽提高多达 1.5 倍[4]。
  - **PCI Express Gen 5（PCIe 5.0）**：带来全新的 I/O 速度，可在 CPU 和设备之间实现更高的吞吐量。第四代英特尔® 至强® 可扩展处理器和英特尔® 至强® CPU Max 系列处理器配备多达 80 条 PCIe 5.0 通道，其 I/O 带宽是 PCIe 4.0 的两倍[4]。
  - **Compute Express Link（CXL）1.1**：提供高网络带宽支持，并使附加加速器高效运行。
- **与英特尔® 至强® 平台实现轻松整合**：由于处理器插槽（Socket）配置相同，可轻松将英特尔® 至强® CPU Max 系列处理器添加到第四代英特尔® 至强® 可扩展平台，并且在大多数部署方案中都无需更改代码。

灵活应对各种科学计算和 AI 工作负载

英特尔® 至强® CPU Max 系列处理器具备出色的灵活性，可根据工作负载的特性，在不同的内存模式或配置下运行：

- **“仅 HBM”模式**：该模式支持内存容量需求不超过 64 GB 的工作负载以及每核 1 至 2 GB 的内存扩展能力，同时无需更改代码和另购 DDR，即可启动系统。
- **“HBM Flat”模式**：该模式可为需要大内存容量的应用提供灵活性，它通过 HBM 和 DRAM 提供一个平面内存区域（flat memory region），适用于每核内存需求大于 2 GB 的工作负载。使用该模式时可能需要更改代码。
- **“HBM 缓存”模式**：旨在提升内存容量需求大于 64 GB 或每核内存需求大于 2 GB 的工作负载的性能。使用该模式时，无需更改代码，且 HBM 可缓存来自 DDR 的事务。

英特尔® 至强® CPU Max 系列	
内核数	32-56
HBM2e 内存	64 GB
HBM 最大传输速率	3200 MT/s
DDR5 最大传输速率	4800 MT/s（1 个 DPC） 4400 MT/s（2 个 DPC）
加速器	AMX，4 个英特尔® DSA
AI/ML 指令	INT8 和 BFLOAT16

跨多架构加速科学计算和 AI 工作负载

整个英特尔® 至强® CPU Max 系列的产品均得到 oneAPI 的支持。oneAPI 是一个统一的、基于标准的开放式通用编程模型，可释放生产力并解锁性能。开发人员可利用英特尔® oneAPI 工具套件以及面向特定领域的专用工具套件，打造跨多种架构运行的通用计算、科学计算和 AI 应用，并对其进行分析、优化和扩展。这些资源包括矢量化、多线程、多节点并行和内存优化方面的前沿技术，让您轻松构建随时能为科学计算所用的高性能、多架构软件。如需了解最新科学计算软件开发人员工具，请访问适用于第四代英特尔® 至强® 可扩展处理器及英特尔® 至强® CPU Max 系列处理器的软件和科学计算软件和工具资源页面。

英特尔® 至强® CPU Max 系列处理器

SKU 编号	内核数	基频 (GHz)	全核睿频 (GHz)	最大睿频 (GHz)	缓存 (MB)	TDP (W)	最大可扩展性	DDR5 内存速度	英特尔® SGX 飞地容量（每个处理器）
9480	56	1.9	2.6	3.5	112.5	350	双路	4800	512 GB
9470	52	2	2.7	3.5	105	350	双路	4800	512 GB
9468	48	2.1	2.6	3.5	105	350	双路	4800	512 GB
9460	40	2.2	2.7	3.5	97.5	350	双路	4800	128 GB
9462	32	2.7	3.1	3.5	75	350	双路	4800	128 GB

如需了解最新信息，请访问
intel.cn/content/www/cn/zh/products/details/processors/xeon/max-series.html

[1] 配合工作负载/配置信息请见 intel.cn/performanceindex（前往 Events 项下的 Supercomputing 22）。结果可能不同。

[2] Numenta BERT-Large

- 英特尔® 至强® 铂金 8480+ 处理器：基于 2022 年 11 月 28 日 Numenta 所做的测试。单节点，2 个英特尔® 至强® 铂金 8480+ 处理器，512 GB DDR5-4800，Ubuntu 22.04 内核 5.17，OpenVINO 2022.3，经 Numenta 优化的 BERT-Large，序列长度为 512，批大小为 1
- 英特尔® 至强® CPU Max 9468 处理器：基于 2022 年 11 月 30 日 Numenta 所做的测试。单节点，2 个英特尔® 至强® CPU Max 9468 处理器，128 GB HBM2e@3200 MT/s，Ubuntu 22.04 内核 5.15，OpenVINO 2022.3，经 Numenta 优化的 BERT-Large，序列长度为 512，批大小为 1

[3] 来源：英特尔网站 P-core（性能核）声明和细节部分有关英特尔® AMX 的描述。“性能指标：2021 年英特尔架构日”，edc.intel.com/content/www/cn/zh/products/performance/benchmarks/（前往 Events 项下的 Architecture Day 2021）。

[4] 更多信息请见 www.intel.cn/PerformanceIndex。实际性能受使用情况、配置和其他因素的差异影响。结果可能不同。

性能测试结果基于配置信息中显示的日期进行的测试，且可能并未反映所有公开可用的安全更新。没有任何产品或组件是绝对安全的。

英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。英特尔技术可能需要启用硬件、软件或激活服务。

产品简介

更高安全保障
英特尔® 至强® 可扩展处理器

英特尔® 安全引擎助力创新加速，增强数据保护

英特尔® 至强® CPU 配备多个英特尔® 安全引擎（Intel® Security Engine)，可在维持出色性能的同时，帮助保护数据机密性与代码完整性。

英特尔® 至强® 可扩展平台支持的机密计算技术可在数据应用过程中确保数据隐私得到保护

如今，对存储和传输状态下的数据进行加密处理已是行业的标准做法。然而，企业在数据保护方面的薄弱之处却是数据在处理器或内存中处于使用状态时。在这种情况下，个人可识别信息、电子病历和金融交易等敏感数据存在较高的被利用风险、很容易发生意外暴露或违反合规要求。

使用机密计算技术，企业可利用敏感数据获得洞察，或者使用敏感数据进行 AI 模型训练，而不会将所用数据暴露给其他软件、合作方或云服务提供商。对于此前因过于敏感或出于监管原因而无法用于分析和其他目的的数据，机密计算技术为企业开辟了利用此类数据的多种可能。

在基于双路英特尔® 至强® 可扩展处理器的服务器中，英特尔® 软件防护扩展（Intel® Software Guard Extensions，英特尔® SGX）飞地可处理高达 1 TB 的数据，因此为需要使用大型数据集的应用创造了更多机会。在完成训练或处理后，隐私信息都可在离开安全飞地前完成删除或重新加密。

基于英特尔® 至强® 可扩展处理器的安全技术，助您释放数据活力，更快向前发展

数据是推动创新与进步的源动力。从检测欺诈性交易到开发响应更迅速的供应链，再到训练具有突破性的 AI 模型，企业可利用数据完成各种各样的任务。可将数据转化为业务洞察的企业能走得更快、更远。

英特尔® 至强® 可扩展处理器的内置安全技术为各种数据（包括敏感、保密和处于监管之下的数据）保驾护航，使其可用于分析，进而帮助企业加速创新步伐。英特尔® SGX 是英特尔的独有技术，能够帮助保护使用中的数据。使用英特尔® 至强® 可扩展处理器的企业不必从数据分析和 AI 模型中剔除敏感数据，而是可通过英特尔® SGX 创建访问受限的数据安全飞地。这样的隔离环境可帮助企业在确保敏感数据始终处于保密状态的前提下，充分发挥其价值。

借助英特尔® SGX 和英特尔® TDX，拥抱机密计算

由英特尔® SGX 提供支持的机密计算可实现应用层面、虚拟机（VM）、容器和功能层面的数据隔离。无论是在云端、边缘还是本地环境，您都能确保自身的计算与数据始终获得私密性和安全性更高的保护，不会暴露给云服务提供商、未经授权的管理员和操作系统，甚至是特权应用。

英特尔® SGX 经过广泛部署和研究，是数据中心可信执行环境（TEE）的重要技术实现，能够大幅减少系统内的攻击面[1]。英特尔® 至强® 可扩展处理器的这一特性为在多个云和边缘部署机密计算解决方案提供了重要支撑。

英特尔® SGX 提供基于硬件的安全解决方案，可通过专用应用隔离技术帮助保护使用中的数据。开发人员可以通过保护选定的代码和数据不被查看或修改，在飞地内执行涉及敏感数据的操作，帮助提高应用的安全性和保护数据的机密性。

英特尔推出英特尔® Trust Domain Extension（英特尔® TDX），进一步提升保护级别。这一全新工具将于 2023 年开始通过特选云服务提供商为企业在虚拟机（VM）层面提供隔离边界和机密保障。 英特尔® TDX 可将客户机操作系统和虚拟机应用都与云端主机、系统管理程序和平台的其他虚拟机隔离开来。虽然英特尔® TDX 的信任边界比英特尔® SGX 应用层面的隔离边界大，但英特尔® TDX 能使机密虚拟机比应用安全飞地更易于进行大规模部署和管理。

英特尔的机密计算技术产品组合在英特尔® SGX 和英特尔® TDX 加持下，允许企业选择他们需要的安全级别，以满足自身的业务需求和监管方面的要求。

客户成功案例：英特尔® 至强® 可扩展处理器提供安全保障，助推创新进程

英特尔® SGX 和英特尔® 至强® 可扩展处理器帮助**全英房屋抵押贷款协会（Nationwide Building Society）**针对演进的“了解客户”（KYC）法律法规，简化合规流程。

了解详情 ›

宾夕法尼亚大学利用英特尔® 至强® 可扩展处理器和英特尔® SGX 优化 3DResUnet 肿瘤分割模型。测试结果：肿瘤边缘检测精度明显提升。

阅读全文 ›

机密计算的选择

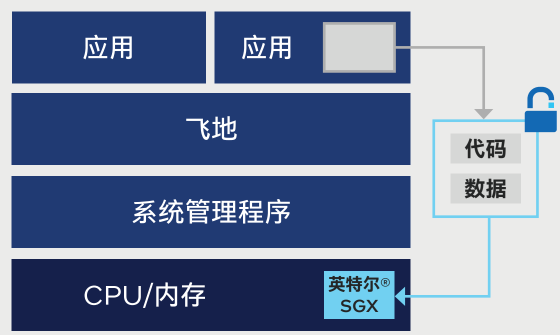


图 1. 英特尔® SGX 通过将敏感数据隔离在容量高达 1 TB 的飞地中，帮助保护敏感数据。

英特尔® SGX 用例

人工智能（AI）/ 机器学习（ML）
使用AI和ML处理敏感或处于监管之下的数据，同时改善隐私法律法规的合规状况。

云基础设施
严格限制服务提供商或其他公有云租户对您私有数据的访问。

可信的多方计算 / 多方分析
支持多方在云端就共享数据开展协作，确保敏感数据始终处于保密状态。

安全密钥管理
使用安全飞地保护密钥并提供类似硬件安全模块（HSM）的功能。

区块链
增强交易处理、共识机制、智能契约和密钥存储的隐私性和安全性。

网络功能虚拟化（NFV）
为虚拟化网络功能建立信任机制。

利用英特尔® 密码操作硬件加速增强安全性，提升数据保护性能

如今，除了传统边界防御，数据中心还依靠加密技术来保护网络传输、存储和数据压缩等进程。随着加密技术的发展，CPU 需要执行的加密周期数量也呈爆炸式增长，这可能对性能和用户体验带来潜在影响。

第四代英特尔® 至强® 可扩展处理器内置多项先进的加密加速技术，无需为数据中心增设更多内核或处理器，即可实现更高级别的加密安全性，提升性能并打造更加顺畅的用户体验。

英特尔® 数据保护与压缩加速技术（Intel® QuickAssist Technology，英特尔® QAT）是一项成熟的数据压缩和加密加速技术，作为全新的内置加速器引入第四代英特尔® 至强® 可扩展处理器，用于支持动态数据压缩/解压缩和加密工作负载。通过卸载计算密集型工作负载，英特尔® QAT 可将更多内核容量释放给其他工作负载，同时显著降低成本和压缩数据的占用空间[2]。

英特尔® 密码操作硬件加速（Intel® Crypto Acceleration）指令采用更加严格的加密协议，例如更长的密钥长度、更强大的算法和更多的加密数据类型[3]，以尽可能降低对用户体验的影响。通过使用更快的加密算法，用户不仅可获得性能提升和支持更高等级的服务级别协议（SLA），还可缩短计算周期，尤其是加密处理阶段的计算周期。

在算法层面，密码操作硬件加速技术主要通过加密计算的以下三个方面实现性能提升：

公开密钥加密：在安全套接字层（SSL）、前端网页和公开密钥基础设施等用例中，公开密钥的加解密速度可提升高达 6 倍[4]。

批量加密：在安全数据传输、磁盘加密和流视频加密[6] 等用例中，使用英特尔® 高级矢量扩展 512（Intel® Advanced Vector Extensions 512，英特尔® AVX-512）可将加密的速度和性能提升高达 4 倍[5]。

哈希：在数字签名、身份验证和完整性检查等用例中，例如安全套接字层（SSL）所用的安全哈希算法 1（SHA-1）和安全哈希算法 2（SHA-2，也称 SHA-256），安全哈希性能可提升高达 2 倍[7]。

微软、SAP 和 Oracle 等公司所提供的多款商业软件包均已完成相关优化，可利用英特尔® 密码操作硬件加速。英特尔已对多款开源软件（众多 Linux 分发版、NGINX、Java OpenJDK Runtime 和 OpenSSL 库）完成优化，可支持英特尔® 密码操作硬件加速。

包括加密 API 工具套件在内的开发人员工具可在英特尔® SGX 安全飞地内以更加安全的方式运行加密操作。此外，英特尔® 集成性能原语（Intel Integrated Performance Primitives，英特尔® IPP）加密库还可自动使用可用的 CPU 资源，而面向 OpenSSL 的英特尔® QAT 引擎则可使网络安全软件解决方案以更加直接的方式，充分发挥英特尔® 密码操作硬件加速的性能。

您可借助英特尔® 至强® 可扩展处理器的内置加密加速技术，缩短加密处理阶段的计算周期，提升开发人员的敏捷性，优化 DevOps 效率，并提升企业的用户体验。

提高监管合规，加速数据分析

对企业有价值的数据经常受到严格的隐私法律法规约束，例如欧洲的 GDPR（《通用数据保护条例》）、美国的 HIPAA（《健康保险携带和责任法案》）和中国的 PIPL（《个人信息保护法》）。违反这些法律法规或会导致高额罚款和其他处罚，因此企业和机构会因面临风险而无法充分利用敏感数据。目前，在使用个人可识别信息方面，已有变通方法可用，例如煞费苦心的匿名化处理，但这项耗时耗力的工作会大幅减缓分析速度，甚至还会影响准确性。借助英特尔® 至强® 可扩展处理器和内置的英特尔® SGX 技术，企业可以创建加密安全飞地，确保数据和应用的保密状态，从而改善合规状况，提升数据的可用性。

> *“到 2023 年，个人信息受现代隐私法律法规监管的全球人口比例将由当前的 10% 增加到 65%。”*
>
> —Gartner[8]

克服敏感数据共享的障碍

在企业和机构间共享数据可以大幅提升准确性，加快神经网络训练等流程。英特尔® 至强® 可扩展处理器支持联邦学习等可信的多方计算模型，使共享机密数据成为可能。使用内置英特尔® SGX 安全飞地的英特尔® 至强® 可扩展处理器，多个参与方就能够汇集敏感数据，共享共同分析带来的益处，而无需将各自的私有数据暴露给彼此。英特尔® SGX 的认证功能让我们更加确信：运行在安全飞地中的软件完全符合各方的预期和既定规约。

助力博世跨越安全难关

英特尔携手工程技术领导企业博世和软件创新品牌 Edgeless Systems 全力加速博世自动驾驶辅助项目的开发进程。为训练计算机视觉模型，博世使用了车辆未来行驶的街道和地点的真实视频与图像。由于此类视频片段包含面部图像、车牌号等处于监管之下的个人可识别信息，因此需要经过匿名化处理，以便博世的工作人员对其进行访问。然而，对数据进行匿名化处理常常会降低 AI 训练数据的准确性。借助英特尔® SGX，博世可在英特尔® SGX 数据的安全飞地中使用原始实时视频片段训练模型，在提升训练速度和训练结果质量的同时，始终做到遵守数据隐私法律法规。

在云端和数据中心建立广泛且可扩展的信任机制

英特尔® 安全技术在帮助企业利用云的灵活性和可扩展性的同时，能够降低暴露敏感数据的风险。英特尔® 至强® 可扩展处理器所支持的机密计算可将您的敏感数据与云服务提供商的软件、管理员和其他租户隔离开来。数据拥有者可通过远程认证功能，验证其安全飞地是否真实可信，是否处于最新状态，且只运行自身期望运行的软件。

选择英特尔® 至强® 可扩展处理器，挖掘更多数据价值

现在，通过全球范围内的云服务提供商和系统制造商，都可获得内置英特尔® SGX 等安全功能的英特尔® 至强® 可扩展处理器。这些处理器不仅可为新服务提供支持，还可增加交易价值、防范金融犯罪、缩短研发周期，并推进使用敏感、有价值或处于监管之下的数据的应用不断向前发展。未来属于那些拥有数据的人。英特尔® 加速引擎助您早日成为数据王者。

进一步了解英特尔® 安全引擎如何为您业务中最关键的工作负载带来出色性能和安全保障。

英特尔–机密计算

1 英特尔® 软件防护扩展保护数据

2 https://www.intel.cn/content/www/cn/zh/developer/articles/technical/offloading-compression-and-encryption-in-ceph.html

3 英特尔解决方案，“Tapping into Cryptographic Acceleration”（掘金于加密加速），https://www.intel.co.jp/content/dam/www/central-libraries/us/en/documents/2022-08/tapping-into-cryptographic-acceleration-sb.pdf

4 英特尔解决方案，“Tapping into Cryptographic Acceleration”（掘金于加密加速），https://www.intel.co.jp/content/dam/www/central-libraries/us/en/documents/2022-08/tapping-into-cryptographic-acceleration-sb.pdf

5 英特尔解决方案，“Tapping into Cryptographic Acceleration”（掘金于加密加速），https://www.intel.co.jp/content/dam/www/central-libraries/us/en/documents/2022-08/tapping-into-cryptographic-acceleration-sb.pdf

6 英特尔解决方案，“Tapping into Cryptographic Acceleration”（掘金于加密加速），https://www.intel.co.jp/content/dam/www/central-libraries/us/en/documents/2022-08/tapping-into-cryptographic-acceleration-sb.pdf

7 英特尔解决方案，“Tapping into Cryptographic Acceleration”（掘金于加密加速），https://www.intel.co.jp/content/dam/www/central-libraries/us/en/documents/2022-08/tapping-into-cryptographic-acceleration-sb.pdf

8 “Gartner 表示，到 2023 年，个人信息受现代隐私法律法规监管的全球人口比例将达 65%，”Gartner，2020 年 9 月，gartner.com/en/newsroom/press-releases/2020-09-14-gartner-says-by-2023--65--of-the-world-s-population-w。

扫码查看专栏文章