本科毕业设计（论文）报告

题　目：基于 BigDL-LLM 和 Intel TDX 的高性能安全聊天机器人系统设计与实现

学　　号：71120105

姓　　名：曹泽桦

学　　院：软件学院

专　　业：软件工程

指导教师：凌振, 史栋杰（企业）

起止日期：2023.12-2024.6

东南大学毕业（设计）论文独创性声明

本人声明所呈交的毕业（设计）论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得东南大学或其它教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。

论文作者签名：_____　　日期：_____年_____月_____日

东南大学毕业（设计）论文使用授权声明

东南大学有权保留本人所送交毕业（设计）论文的复印件和电子文档，可以采用影印、缩印或其他复制手段保存论文。本人电子文档的内容和纸质论文的内容相一致。除在保密期内的保密论文外，允许论文被查阅和借阅，可以公布（包括刊登）论文的全部或部分内容。论文的公布（包括刊登）授权东南大学教务处办理。

论文作者签名：_____　　导师签名：_____

日期：_____年_____月_____日　　日期：_____年_____月_____日

摘　要

本文设计并实现了一个基于模型的低比特量化的、在云环境中既注重性能又强调隐私保护的聊天机器人系统。随着大语言模型在文本分类、对话生成和上下文补全等多种任务中表现出色，其应用领域也日益扩展。然而，模型的参数规模随着其能力的增强而迅速增大，这不仅使得模型部署和推理计算对硬件的要求越来越高，也使得个人学习者难以承受昂贵的成本。此外，在处理大数据集和当前无法解决的模型幻觉问题时，如何在云环境中保护大规模语言模型的隐私，也成为了一个重要挑战。

因此，本文首先以分层可扩展的方式实现了具有基本功能的聊天机器人系统，之后，以 Llama2 模型为基础，基于绝对值量化和零点量化，实现了一种 4 位整数逐组量化方式，且充分利用 Intel 处理器的 AVX512 指令集进行优化，显著提高了模型性能（加载时间、内存消耗、困惑度、能力评分等）；同时基于 BigDL-LLM 这一先进的大语言模型优化库，进一步加速了模型的推理速度。此外，本文还设计了系统的容器化及云端部署方案，同时利用 Intel TDX 硬件加密技术实现了系统在云环境中的隐私保护推理，最大程度降低了用户隐私泄露的风险。

评估结果表明，本文实现的量化方法对模型能力的影响可以控制在 5% 左右，但减少了约 75% 的内存占用和 85% 的模型加载时间。同时，该系统的云端部署方案具有良好的负载均衡能力，能够充分利用集群中各节点的计算资源，实现约 50% 的响应时间加速，并在引入最多 9.5% 的额外延迟的情况下，成功在 Intel TDX 环境中集成，提供了完整的隐私防护。

关键词：大语言模型，低比特量化，对话系统，隐私保护

ABSTRACT

This thesis designs and implements a chatbot system that is both performance-focused and privacy-conscious within a cloud environment, based on low-bit quantization of models. As large language models excel in tasks such as text classification, dialogue generation, and context completion, their application areas are expanding. However, the rapid increase in model parameter size with enhanced capabilities raises hardware requirements for model deployment and inference computations, making it costly for individual learners. Additionally, privacy protection of large language models deployed in cloud environments presents a significant challenge when handling large datasets and unresolved model illusions.

To address these challenges, this thesis first implements a chatbot system with basic functionalities in a scalable, layered manner. Building on the Llama2 model, it employs a group-wise 4-bit integer quantization approach using absolute value and zero-point techniques, optimized with Intel's AVX512 instruction set, significantly improving model performance in terms of load time, memory consumption, perplexity, and capability scores. Furthermore, leveraging the BigDL-LLM library, an advanced optimization tool for large language models, enhances inference speed. The system's containerization and cloud deployment strategy, utilizing Intel TDX hardware encryption, ensure rigorous privacy protection in cloud environments, minimizing the risk of user privacy breaches.

Evaluation results indicate that the implemented quantization method controls the impact on model capabilities to about 5%, while reducing memory usage by approximately 75% and model loading time by 85%. The cloud deployment strategy offers effective load balancing, maximizing the computational resources of the cluster nodes, accelerating response times by about 50%, and integrating into the Intel TDX environment with an additional delay of only up to 9.5%, thereby providing comprehensive privacy protection.

KEY WORDS: Large Language Models, Low-bit Quantization, Chatbot System, Privacy Protection

目　　录

摘　　要 I

ABSTRACT II

目　　录 III

第一章　绪论 1

1.1 课题背景和意义 1

1.2 研究现状 2

1.2.1 大语言模型的发展 2

1.2.2 模型量化技术 3

1.2.3 大语言模型应用中的隐私问题 5

1.3 本文研究内容 6

第二章　方案概述 8

2.1 方案难点与贡献 8

2.2 系统概述 9

第三章　系统设计与实现 11

3.1 系统功能性设计 11

3.1.1 控制器设计与实现 11

3.1.2 工作节点设计与实现 13

3.1.3 用户接口层的设计与实现 14

3.2 低比特量化实现 16

3.2.1 技术基础 16

3.2.2 量化过程 18

3.2.3 量化粒度选择 20

3.2.4 BigDL-LLM 优化 23

3.3 容器化及云端部署方案 24

3.3.1 Docker 容器化实现 24

3.3.2 Intel TDX 集成 25

3.3.3 云端部署方案 27

第四章　实验分析 29
4.1 模型基础指标评估 29
4.2 模型能力测试 32
4.3 系统性能评估 33
第五章　总结与展望 39
5.1 工作总结 39
5.2 工作展望 39
参考文献 40
附录 A　正文符号定义 44
附录 B　量化算法核心代码 46
附录 C　实验部分涉及数据集 49
致　　谢 50

第一章 绪论

1.1 课题背景和意义

近年来，随着关键的硬件及软件的不断发展，大语言模型（Large Language Models, LLMs）得到飞速发展，在理解和生成自然语言方面取得了显著成就，为多种应用开辟了新的可能性。从简单的文本生成到复杂的对话系统，再到深入的语言理解和知识抽取，LLMs 的影响范围已经远远超出了最初的预期。

在硬件方面，GPU（图形处理单元）和 TPU（张量处理单元）[1] 的持续进步为训练和运行这些庞大的模型提供了必要的计算力。特别是 TPU 的引入，其为处理大规模并行计算任务特别设计，极大地加速了模型训练过程，使得开发者能够在实际操作中训练拥有数百亿乃至数万亿参数的模型。此外，高速互联网和数据中心的发展也为大规模数据集的存储和处理提供了支撑，这对于训练深度学习模型尤为重要。

在软件方面，开源框架如 TensorFlow、PyTorch[2,3] 的出现极大地降低了开发和实验复杂模型的门槛。这些框架提供了高效、灵活的计算图库和丰富的 API，支持自动化的梯度计算和高性能的数值计算，从而允许研究人员和开发者专注于模型设计和实验，而不是底层的编程细节。更重要的是，这些框架的社区活跃，提供了大量预训练模型和教程，促进了知识的共享和技术的快速迭代。

这些关键的硬件和软件条件的结合，不仅推动了大语言模型的技术进步，也为其在多个领域的应用提供了可能[4]。今天，LLMs 已经在文本摘要、情感分析、机器翻译、自动编程、内容创作等众多领域展示了其强大的能力。更进一步，它们正被用于探索更为复杂的任务，如跨语言和文化的交流、多模态理解（结合文本、图像、声音等多种数据类型的理解）以及更智能的人机交互系统。

然而，大语言模型的进步也伴随着诸多问题和挑战。特别是在性能和安全方面。首先，巨大的资源消耗和环境影响、模型泛化能力的局限性，以及模型决策过程的不透明性，成为了性能提升的主要障碍。其次，在安全性方面，数据隐私泄露的风险、模型输出中的偏见和不公平现象、以及模型的误用和恶意使用，都引发了社会和伦理上的深刻关切。这些问题不仅挑战了技术开发者和研究者寻找创新解决方案的能力，也促使社会各界对于技术伦理和责任的讨论。因此，在追求技术进步的同时，如何有效应对这些挑战，确保技术的安全、公正和可持续发展，已成为大语言模型研究和应用中不可或缺的一部分。

1.2 研究现状

1.2.1 大语言模型的发展

自 2017 年 Transformer[5]模型首次提出以来，其独创的自注意力机制彻底改变了自然语言处理（NLP）领域的研究方向，为后续的技术进步奠定了坚实的基础。这一创新架构通过有效捕捉序列数据中的长距离依赖，显著提高了处理语言任务的效率和准确性。紧接着，基于 Transformer 的 GPT[4]系列和 BERT[6]等模型的出现，不仅继承了其核心优势，还通过规模扩张和策略创新，进一步推动了整个领域的飞速发展。

GPT 系列，尤其是 GPT-3[7]，通过其庞大的模型规模（高达 1750 亿参数）和在广泛文本上的预训练，展示了惊人的语言生成和理解能力。这种模型不仅能够在多种 NLP 任务上实现令人印象深刻的性能，还展示了在接受少量指导示例后迅速适应新任务的能力，但其训练数据集极其庞大，无法通过人工完整评估，也无法剔除数据集中重复、受污染的数据。与 GPT 系列并行的 BERT 模型，通过引入双向预训练机制，能够更加深入地理解语言的上下文含义，从而在文本理解任务上设置了新的性能标准。BERT 模型的变体 RoBERT[8]通过仔细测量关键超参数和训练数据大小对结果的影响，获得了更优的性能。但模型性能的提升往往伴随着计算预算的增加，但当模型或数据集大小固定后，性能提升的增长比率会逐渐降低，显示出明显的收益递减趋势，这不仅意味着巨大的经济成本，也指向了对更高效算法和技术的需求。ALBERT[9]提出了两种参数减少的技术，以降低记忆体消耗并增加 BERT 训练速度，ALBERT 比原始 BERT 模型具有更好的扩展性和稳定性。DistilBERT[10]在预训练阶段使用知识蒸馏，在减小模型尺寸的同时保留了语言理解能力并提升速度。

Llama[11]模型是最近提出的一种大规模语言模型，旨在通过改进的架构和训练技术，进一步提升模型的效率和性能。在前文讨论了 GPT、BERT 及其衍生模型的基础上，Llama 模型特别注重在保持预训练语言模型高效性的同时，优化模型的计算资源消耗。Llama 模型采用了一种新的训练策略，这种策略使其能够在较小的数据集上也能实现较高的性能。与 GPT-3 这样的模型相比，Llama 在处理少量样本学习时表现出更高的效率，这主要得益于其优化后的参数共享技术和训练过程中更精细的注意力机制。此外，Llama 模型在预训练阶段引入了更高级的数据清洗和预处理技术，有效地减少了数据冗余和噪声，提高了模型训练的质量和稳定性。另一个 Llama 模型的显著优势是其能力在多任务学习方面的提升。通过更灵活的网络结构和参数调整，Llama 模型不仅在标准 NLP 任务上表现出色，还能更好地适应特定领域的需求，如生物医药、法律和金融等专业领域。总的来说，Llama

模型在设计和训练策略上的创新，使其在模型性能、效率和适应性方面具有明显优势，为解决大规模语言模型中存在的问题提供了新的解决方案。这些优势使得 Llama 模型成为继 GPT、BERT 及其变体之后的又一重要进展。

1.2.2 模型量化技术

在 2022 年之初，机器学习领域广泛认同一个观点：模型的规模与其性能成正比关系。尤其是当模型规模超过某一临界点时，它们的性能会经历一个质的飞跃，这一现象被称作涌现能力[12]。然而，大型语言模型的训练不仅需要庞大的数据集和强大的工程支持，还依赖于重要的基础设施建设。例如，Microsoft 和 Nvidia 共同开发的 Megatron-Turing NLG 530B[13]模型，就需要使用数百台价值 19 万 9000 美元的 DGX A100 GPU 服务器。因此，对于大多数组织来说，构建如此规模的大型语言模型几乎是不可能的任务。

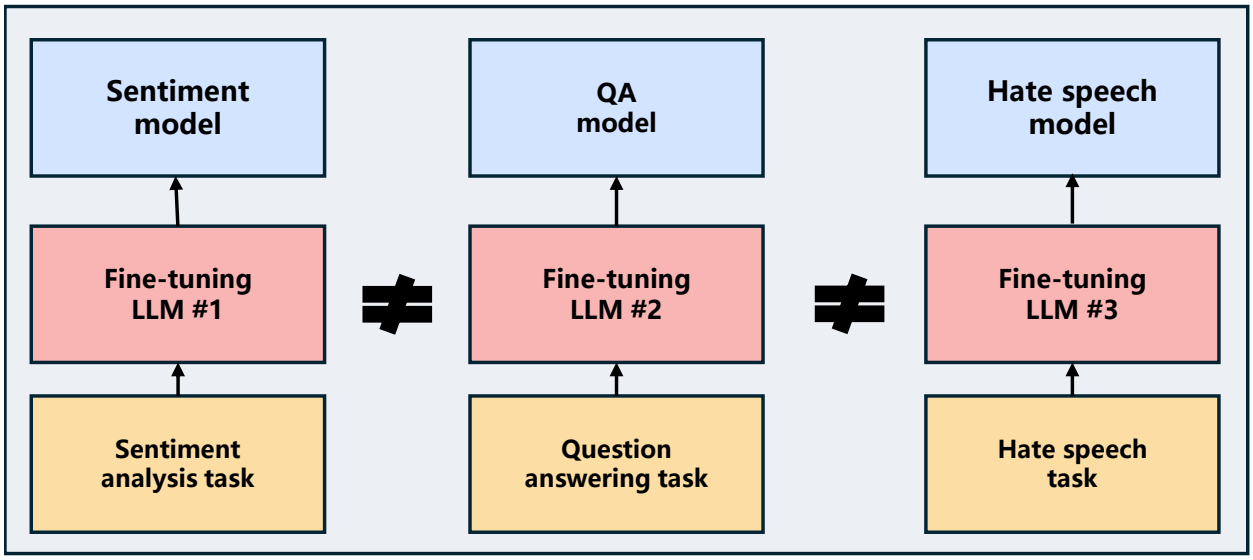


图 1-1 下游具体任务微调示例

到了 2023 年，仅解码器（decoder-only）式的 Transformer[5]模型经历了显著的增长，众多大型语言模型在开源社区中相继出现，这为广大个人和组织提供了使用大型预训练模型的标准选择。对于那些需要将大型语言模型应用于特定任务的情况，可以采用微调（Fine-tuning）的方法。具体而言，先在大规模的文本数据上进行预训练，然后再在较小规模、针对特定任务的文本数据上继续进行训练。即使在这样的背景下，模型的推理计算（即生成预测结果的过程）还是需要将整个模型加载到内存中，这本身就是一个资源密集型的过程。以 Llama-2 模型为例，参数规模从 70 亿到 700 亿不等，即使对于 Llama-2-7b 模型，很多个人设备也无法支撑模型加载所需要的内存开销。而对于全模型微调，必须使用额外的数据存储空间存储模型副本，并且在用于具体任务时进行加载，这也需要额外内存

开销，如图1-1所示，对于不同任务如情感分析、问答等，都需要针对任务生成新模型。总而言之，要想学习和使用大语言模型，降低其内存消耗和计算开销是必不可少的任务。

而模型量化技术可以降低模型的内存开销和计算复杂度，因此成为一个关键的研究方向。模型量化通常涉及减少模型参数的数据表示精度，从而减轻存储和计算负担，同时尽量保持模型性能。近年来对模型量化的研究层出不穷，模型量化在最初的定义里是为了压缩模型参数，比如 2016 年的研究[14]首次提出了参数量化方法。其使用 k-mean 聚类，让相近的数值聚类到同一个聚类中心，复用同一个数值，从而达到用更少的数值表示更多的数，这是量化操作的一种方案。反过来，从量化数变到原始数的过程，称之为反量化，反量化操作完之后，模型就可以按照原来的方式进行正常的计算。

早期的量化方案并不一定能带来运算速度的提升，因为量化数值的计算未必能在部署硬件获得更高峰值性能，同时量化算法带来的额外计算也可能很大。对于模型参数的量化所节省的内存也只占计算时的很小一部分，大部分来源于激活值 activation。此外，比特越大的量化带来的精度损失也越大，因此模型的量化方案还存在很多困难和挑战。2022 年提出的 LLM.int8()[15]量化方法对于绝大部分的权重和激活使用 8bit 量化，对离群值保留 16bit 进行矩阵分解，在进度上超过了各种 int8 量化方案，但带来的额外计算开销较为影响性能，其保留的 16bit 离群值对硬件不太友好。ZeroQuant[16]采用了更细粒度的量化，通过逐层量化，在 BERT 和 GPT3 风格的模型上获得了更好的精度，且能量化至 4bit，但其加速效果非常不明显。AWQ[17]仅对权重做 3/4bit 量化，通过引入平滑因子来保护重要权重，在 Llama 和 OPT 模型上取得了更好的效果。SmoothQuant[18]和 AWQ 的思想类似，在量化过程中，将激活值的量化难度转嫁部分给权重值，相较于朴素量化如 LLM.int8() 在精度上取得了较好的效果。

此外，除模型部署和推理本身的巨大性能开销外，模型的微调性能开销也不容忽视。LoRA（Low-Rank Adaptation）[19]和其后续扩展 QLoRA[20]代表了在大型语言模型微调领域的重要进步。LoRA 通过向 Transformer 架构的每一层注入可训练的低秩分解矩阵，显著减少了微调过程中需要调整的参数数量，在不牺牲性能的前提下，大幅降低了微调的计算成本和内存需求。接着，QLoRA 在 LoRA 的基础上进一步引入了量化技术，使用 4 位 NormalFloat 数据类型和双重量化策略，实现了对大型模型（如 65B 参数）在资源有限的硬件（例如单张 48GB GPU）上进行高效微调，而不损失任务性能。这些技术的结合推动了在计算资源受限环境中实现复杂 NLP 任务的可能性，展示了技术创新在提升模型性能

与可达性方面的巨大潜力。

1.2.3 大语言模型应用中的隐私问题

在 LLMs 自然语言处理能力不断提升的同时，也带来了用户隐私方面的重要挑战。首先，在大规模语言模型的预训练阶段，需依赖于广泛的数据集，这些数据集因其庞大的规模而难以全面审核其质量，进而可能含有可用于个人身份识别的信息。先前的研究成果已经展示[21]，在一个涵盖真实 Reddit 用户资料的数据集情境中，现代 LLMs 能够以远低于人工成本和时间的方式，推断出广泛的个人属性（如地点、收入和性别等）。随着人们日益频繁地与基于 LLMs 的聊天机器人在生活各领域进行交互，个人隐私泄露的风险亦随之增加。近期研究[22]通过对 BERT、DistilBERT 及 OpenAI GPTs 等流行 LLMs 执行实证攻击，揭露了它们在处理多个现实世界语言数据集时存在的显著隐私安全漏洞。这些发现凸显了在开放网络环境下保护用户隐私的紧迫挑战。

尽管通过 Direct Preference Optimization (DPO)[23]等方法，LLMs 能够更精准地根据人类偏好进行行为调整，增强模型的社会化能力，这种精细化的调整过程却可能涉及处理大量个人信息，从而增加隐私泄露的风险。进一步的研究揭示[21]，LLMs 凭借推理能力能够从文本中推断出个人属性，如地点、收入和性别等，即使在实施了文本匿名化和模型对齐等常用隐私保护措施后，个人隐私仍然面临被侵犯的威胁。Safe RLHF[24]、DAEA[25]、ToolEmu[26]都试图通过模型编辑、防御等方法从模型权重中完全删除敏感信息，但都无法完全全面删除，任存在较大风险发生隐私泄露、财务损失等问题，且识别风险是劳动密集型任务。随着 LLMs 在数据隐私和安全性、模型偏见与不公平，以及误用和恶意使用等方面的风险日益显著，如何在发展 LLMs 技术的同时确保用户隐私和公平性，防止技术被滥用，成为了当前亟需关注和解决的问题。

此外，对于大语言模型的部署和应用场景，一般来说都基于服务器厂商所提供的云平台环境，这种部署模式可能将个人敏感信息暴露给不受信任的云服务管理员或其他云服务用户。带来了一系列与数据安全和隐私保护相关的额外挑战。云服务提供的便利性和可扩展性虽然为 LLMs 的部署提供了强大支持，但同时也使得用户数据可能通过多个租户和区域进行传输和存储，增加了数据被非法访问或滥用的风险。确保在这样的环境中用户数据的安全和隐私，需要采取一系列的技术和政策措施。Intel TDX[27]和 AMD SEV[28]这样的硬件安全技术，为云环境中的数据安全提供了重要的支撑。这些技术能够在物理硬件层面提供强大的安全保障，使得即便在共享的云平台上，数据也能得到有效的保护。Intel TDX

（Trust Domain Extensions）和 AMD SEV（Secure Encrypted Virtualization）都是针对虚拟化环境设计的技术，它们允许数据在处理和存储过程中保持加密状态，即使是云服务提供商也无法访问未加密的数据。

Intel TDX 技术的核心在于创建一个安全的隔离环境——信任域（Trust Domain），使应用程序和数据能够在其中安全运行，而不受外部操作系统的影响。这种隔离确保了即便是在操作系统级别发生安全漏洞，攻击者也无法访问或篡改在信任域中的数据和代码。AMD SEV 技术则是在虚拟机层面提供加密支持，确保虚拟机的内存始终加密，只有在本地 CPU 上才能解密。这不仅防止了来自其他虚拟机的潜在侵犯，也保护数据免受物理攻击，如冷启动攻击和行侧信道攻击。

因此，通过结合先进的硬件安全技术与严格的管理措施，能够在云环境中有效地保护大语言模型的部署与应用，确保数据安全与隐私得到切实维护。尽管这些措施无法直接解决大语言模型在隐私保护方面的内在问题，但它们提供了一种可行的思路：构建一个全面的安全框架，从而提升云服务的可靠性，使用户无需对安全和隐私问题过分忧虑。这不仅有助于增强信任，也推动了技术的广泛应用和发展。

1.3 本文研究内容

在当前大语言模型迅速发展的背景下，商业聊天机器人系统虽已相对成熟[29–31]，但个人开发者和学习者在部署、微调及高效应用 LLMs 时，仍面临着显著的硬件资源和成本挑战。同时，庞大的数据量和高复杂度处理也对个人隐私保护构成了前所未有的挑战。鉴于这些问题，本研究致力于设计并实现一个既能在资源有限的设备上运行，又能确保数据安全性和隐私保护的高性能安全聊天机器人系统。基于现有的模型量化方法进行深入研究，设计了一种在尽量减少精度损失的同时能最大限度降低内存消耗的量化策略，并利用 BigDL-LLM 库优化模型计算，以进一步加速模型推理速度。在系统完成云环境部署之后，结合 Intel TDX 硬件加密技术，确保数据处理过程的绝对安全。本研究旨在突破现有技术限制，推动大语言模型在更广泛的应用环境中得到实际应用。

就如上的需求，本研究在后续分析了系统设计所面对的难点，并主要进行了如下的系统实现工作：

系统设计与实现：采用控制器-工作节点-用户接口分层实现系统。本研究考虑多个工作节点调度问题，通过 Heart-beat 保活机制维护工作节点状态，通过 LOTTERY 和 SHORT-

EST_QUEUE 两种调度方式实现并发场景下的负载均衡；工作节点实现了基本的通信接口和模型加载；用户接口层实现了 API 服务器和图形化用户界面。

大语言模型的量化方案：以 Llama2 模型为例实现了 int4 低比特量化，并通过集成并应用 BigDL-LLM 中的先进优化技术，本研究关注于模型加载、推理和微调过程的效率提升。特别是，研究围绕降低内存消耗、加速推理过程以及保障分布式计算的可行性三大核心目标展开，确保即使在资源受限的环境下也能保持高效的模型性能。

系统云端部署方案设计：采用本地-容器化-云部署的技术路线，本研究从裸机环境开始，首先实现 Docker 容器的封装，之后通过 kubernetes 进行云端部署，实现系统的高可用性和可扩展性。这种设计不仅满足了从单机到分布式环境的平滑过渡，也为系统提供了灵活的部署和管理能力，以应对不同规模的负载需求。为了进一步增强系统的数据安全性，本研究在云端部署方案中集成了 Intel TDX 硬件加密技术。通过为数据处理和存储提供安全的执行环境，Intel TDX 技术能有效防止隐私泄露和未授权访问，为用户的数据安全提供坚固的保障。

安全性与性能评估：针对所设计系统，本研究制定了全面的安全性和性能评估方案。通过综合评估包括响应时间、系统吞吐量、内存消耗在内的性能指标，以及涵盖存储、计算和通信等过程的安全性指标，本研究旨在全方位地验证系统设计的有效性和优越性。

第二章 方案概述

2.1 方案难点与贡献

本研究从当前聊天机器人系统的发展趋势出发，深入分析了现有的部署模式，主要集中于两种典型方式：首先，是基于调用商业化大语言模型 API 的构建方式，如 OpenAI 提供的 API 服务。此模式主要通过 API 调用计费，将计算任务完全迁移到云端，而终端设备仅承担输入 prompt 和网络 API 调用等功能性需求。其次，是依托 Hugging Face 开源社区提供的众多 Transformer 模型，进行本地加载、微调和推理计算的自主部署方式。在第一种部署模式中，虽然用户能够避免本地运行大语言模型所需的计算资源开销，但这种方式使得用户对模型的运行机制和处理过程的了解十分有限。这对于致力于研究和学习的个体而言，并不利于其深入理解模型工作原理；同时，该模式也大幅度增加了隐私泄露的风险。而第二种部署模式虽然首先需要解决个人设备上运行大规模语言模型的计算资源问题，但它为提供一个完整的、高度可用且可扩展的聊天机器人系统提供了可能，同时也为保护用户隐私提供了更多的空间。因此，本研究认为，第二种部署方式更符合本文的研究目的，旨在探索能够同时优化性能和保障用户隐私的聊天机器人系统部署策略。此种自主部署方式，不仅促进了对语言模型深入理解的可能性，也为确保用户隐私安全提供了更为可行的解决方案。

在个人设备上部署并运行大型语言模型时，面临的主要挑战之一是内存与计算资源的限制。本文首先实现了 int4 低比特量化，能够对 Llama2 模型实现量化来实现大幅度的内存降低。之后基于 BigDL-LLM 库进行进一步优化，BigDL-LLM 库，专为在 Intel GPU 和 CPU 上加速大型语言模型的运行而设计，已经整合了多个领先项目的成果，包括 llama.cpp、bitsandbytes、vLLM[32]、qlora[20]、AutoGPTQ[33]等，显著缩减了本研究在技术选型和实践操作上的时间成本。本文旨在开发一种可扩展至云端的聊天机器人系统。鉴于该系统基于 Intel 硬件架构，采用 Intel TDX[27]进行机密计算，为隐私保护提供了一种可行的方案。通过集成 Intel TDX 技术，虽然无法完全避免在 LLMs 运行过程中对个人属性信息的暴露，但能够有效防止在用户交互过程中的隐私泄露风险。此外，在系统云端部署完成之后，本文进一步探讨了系统在高并发场景下的调度与负载均衡能力。特别是模拟多用户并发场景，通过这种模拟，有助于了解系统在扩展独立计算资源时的处理能力，从而增强整个系统的响应能力和效率。这些研究不仅确保了系统的高性能运行，也为未来在复杂应用场景中的

实际部署提供了技术基础。

系统设计面临的难点：

（1）资源限制与管理：在个人设备上部署大型语言模型时，面对的首要挑战是内存和计算资源的限制。合理分配和优化有限的资源，以支持 LLMs 的高效运行，是系统设计中的一大难点。

（2）量化策略的设计与实现：在量化过程中，首先需要选取合适的量化方案和量化粒度，其次需要确保精度与性能的平衡，而且还需要考虑硬件特性，以确保最好的计算性能和兼容性。

（3）隐私保护：随着 LLMs 的应用越来越广泛，如何在不可信的云环境中，有效保护用户隐私，避免敏感信息泄露，成为设计中必须考虑的问题。

（4）技术栈的选择与整合：选择合适的技术栈，并将各种技术（如 BigDL-LLM、Intel TDX 等）有效整合到系统设计中，以发挥它们的最大优势，需要充分的技术洞察和实践经验。

本文的贡献如下：

(1) 基于 Llama2 模型实现了一种高效且保证模型精度损失的量化方法，并集成了 BigDL-LLM 优化来加速推理计算，本研究提出了一种在个人设备上执行大型语言模型推理计算的有效方案。

(2) 设计了一个具有对话、补齐等基本功能的聊天机器人系统，且功能性可扩展，能够兼容多种硬件平台，同时利用容器化和云原生技术提供了云端部署方案。

(3) 将 Intel TDX 技术应用到了聊天机器人系统中，从而最大程度减少了未授权访问和不可信云平台上隐私泄露的风险，为 LLMs 的隐私保护提供了新的思路。

2.2 系统概述

本系统采用三层架构，将用户接口层、控制器和负责大语言模型推理计算的工作节点分开设计，对外向用户提供了上下文补全、对话等功能，核心在于如何对预训练的大模型进行量化，以满足系统在使用时的内存和计算资源限制。图2-1展示了这一聊天机器人系统的概览。在本系统中，控制器负责工作节点的调度；工作节点需要实现和控制器的通信等基础功能，同时需要完成模型的量化和加载以及集成 BigDL-LLM 库的优化；用户接口层有图形用户界面和 API 服务器两种方式，其中 API 以 openai 规范为标准实现。

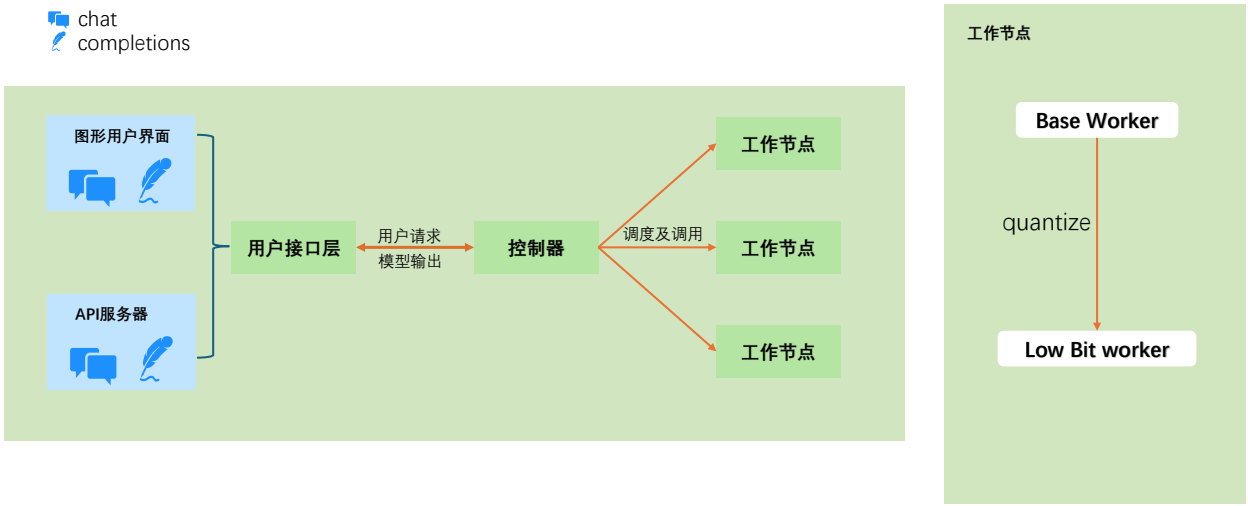


图 2-1 聊天机器人系统概览

系统实现的步骤为：

（1）控制器实现：涵盖了对工作节点的调度策略及其保活机制。该系统采用了彩票调度（LOTTERY）和短队列调度（SHORTEST_QUEUE）策略。通过心跳机制（heart-beat）监控工作节点的状态，以确保系统稳定运行

（2）工作节点实现：工作节点首先需要提供控制器调度策略中所需要的接口，以及发送 heart-beat 等状态信息，此外关于加载模型部分，需要集成量化算法以及 BigDL-LLM 优化。关于量化的实现，主要涉及到基础方案的选择、量化粒度的选择，并且需要考虑硬件特性，尽可能减少量化过程所带来的额外计算开销。

（3）用户接口层实现：此层的主要任务是将遵循 OpenAI 规范的请求转化为模型在预训练阶段所接收的 prompt 格式，解决了用户输入与模型需求不匹配的问题。同时，系统还实现了流式响应接口，以优化用户体验。

（4）云端部署方案：由于本系统需整合 Intel TDX 硬件加密技术，TDX CoCo（Confidential Containers）允许用户将现有的 Docker 镜像无缝迁移到 TDX 环境。基于此，系统在已有的功能性架构上，提供了一个容器化解决方案，并进一步制定了 k8s 的部署方案。通过在 k8s 系统中集成 TDX CoCo，降低了系统在不可信云环境中受攻击和用户隐私泄露的风险。

第三章 系统设计与实现

系统功能架构已经在 §2.2中进行概述，本章将对系统进行详细描述。

3.1 系统功能性设计

3.1.1 控制器设计与实现

控制器 (controller) 的目标在于管理和调度可能涉及到的多个模型工作节点，并尽可能将最合适的工作节点返回给用户接口层 (GUI 或者 api server) 进行调用来满足系统的负载均衡。因此控制器首先需要维护所有工作节点的信息以便实现调度。

在多个工作节点的系统中，控制器需要知道所有工作节点的状态，调度可用节点并清理下线的节点。为了实现上述目标，控制器需要维护工作节点的一些信息，如表3.1

表 3.1 控制器中需要维护的工作节点信息

变量名	类型	注释
model_names	List[str]	工作节点运行的模型名称
speed	int	工作节点的响应速度，用于调度
queue_length	int	待处理任务的队列长度，用于调度
check_heart_beat	bool	心跳信息，标记工作节点是否可用
last_heart_beat	str	记录上次发送心跳的时间，用于保活

Heart-beat 保活机制

关于 API 端点的设计，包括和工作节点以及用户接口层的交互，将统一采用 **FastApi** 框架进行设计。由于控制器必须确保发送给用户接口层的工作节点是可用的，没有因为网络、硬件资源等原因下线，所以必须引入工作节点的保活机制。而在控制器和工作节点这种情况下，需要一个能够快速或实时监控连接状态的机制，而 heart-beat 是一个合适的方案。

Heart-beat 与 TCP 连接中实现的 keepalive[34] 类似，都是发送一个信号给对方，如果多次都没有响应则判断连接中断。在控制器和工作节点这种情景下, 工作节点作为被保活的对象，需要发送心跳信号，而控制器则需要接收心跳信号并判断节点状态。

在 controller 中会单独启动一个线程用于检测 worker 的状态，通过设定一个过期时间 EXPIRATION 来定期清理没有在规定时间内发出心跳的 worker。每一个 worker 在启动并

注册到 controller 时候也会启动一个线程来负责以 EXPIRATION 为间隔来发送心跳信号。controller 中有对应的接口来接收心跳信号，并且将接收到的心跳信息记录到3.1中所述的数据结构当中。工作节点不可用的条件是上一次发出心跳的时间大于所设置的过期时间。此时 controller 会将其删除，标记为已下线。该流程如图3-1所示。

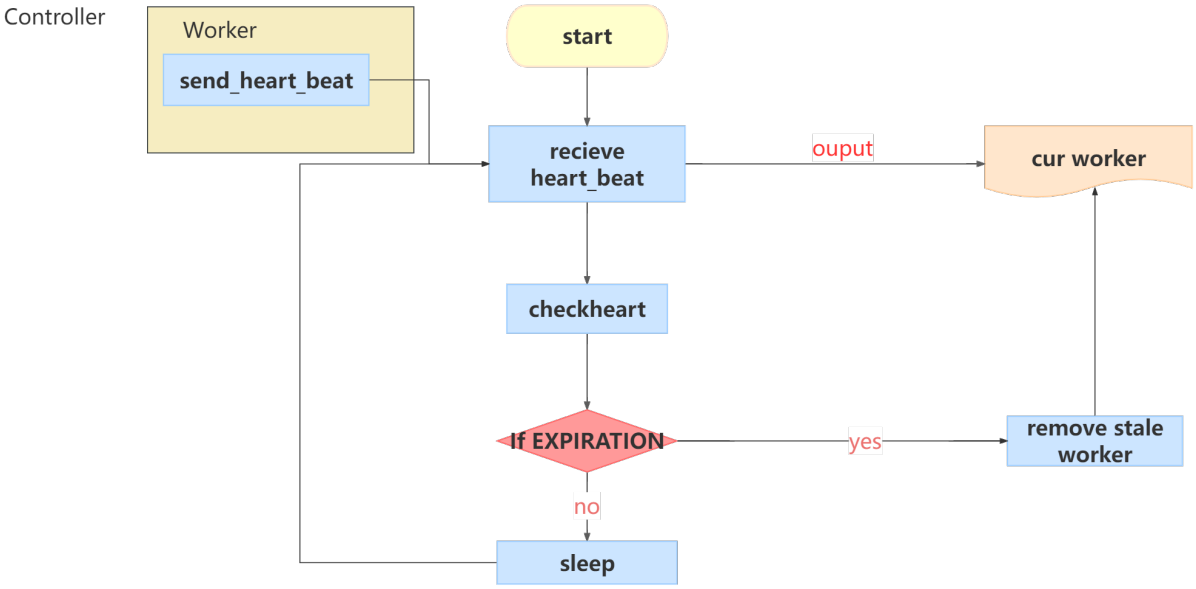


图 3-1 控制器保活机制 heart-beat 流程

调度策略设计

控制器还需要解决的另外一个难点则是关于工作节点的调度问题，需要选择出最合适的工作节点交给用户去调用，以保证整个系统的负载均衡。根据研究[35]中的多种调度算法，本系统选择实现了 LOTTERY 和 SHORTEST_QUEUE 两种调度方式。

在操作系统中，LOTTERY 调度为进程分配一定数量的彩票，而彩票的多少决定了调度决策中被选中的概率大小。具体来说，每个进程根据其优先级和资源需求获得一定数量的彩票。而应用到该系统中，彩票数将转化为每个工作节点的 speed，即代表了工作节点处理任务的能力，该值越高，工作节点被选中的概率也就越大。

SHORTEST QUEUE 调度是一种用于负载均衡的调度算法，此调度方法旨在选择等待队列最短的服务器或资源来处理新的任务或请求，从而优化整体系统性能和响应时间。具体而言，控制器首先收集了所有工作节点的队列长度，之后根据队列长度的大小来寻找节点。而为了更优的实现负载均衡，实现时将计算工作节点的队列长度与速度的比值来选择，因为对于速度较快的节点，即使队列长度稍长，也可以是好的选择。最后被选中的节点队

列长度加 1。该算法可以表示为:

设 N 为支持特定模型的工作节点集合。对于每个节点 $n \in N$, 设 L_n 为节点 n 的队列长度，S_n 为节点 n 的处理速度。定义比值 R_n 为：$R_n = \frac{L_n}{S_n}$ 在所有节点中，选择 n^* 使得 R_{n^*} 最小，即 $n^* = argmin_{n \in N} R_n$，最后更新选择节点的队列长度：$L_{n^*} = L_{n^*} + 1$。

3.1.2 工作节点设计与实现

工作节点 (worker) 作为系统中实际调用模型完成推理计算的部分，需要完成加载相应模型、对外提供对话补齐等功能。而关于如何与控制器进行通信以及保活机制已经在§3.1.1中阐述。加载模型需要调用 Transformers 库中所提供的标准接口，关于如何实现这一过程的量化将在 §3.2中详细介绍。本章重点介绍工作节点如何提供对话服务相关的功能性接口。

LLMs 本质上都是基于语言的模型，因此它们通常提供的是简单的语言模型调用方式：接受一个字符串格式的输入，通过计算预测一个字符串作为输出。然而，这种单纯的文本输入输出模式并不直接适用于提供对话服务。在这个过程中，关键问题是如何将多轮对话按照模型在训练时采用的格式转换成模型的输入。作为最早推出开放式对话服务的公司之一，OpenAI 的接口设计被证明是实现此类功能的一种有效方式。OpenAI 风格的对话接口主要包含两部分：角色（role）和内容（content），其中角色分为 system、user 和 assistant。然而，不同的 LLM 在训练时所采用的模板各不相同，这就需要将 OpenAI 风格的对话接口转化为适合特定模型的模板。为了在工作节点上提供对话功能，系统首先需要接收采用 OpenAI 风格的对话输入，然后根据不同模型的特定模板进行相应的转换。开源项目 FastChat 已经为主流的大语言模型封装了 Conversation 类，便于将用户的输入转化为模型所需的特定模板。因此在多模型环境中笔者直接调用这样的封装，免去了对模型进行逐个适配的繁琐工作。

在解决模型输入的基础上，另一个问题在于如何解决用户的并发请求。这里借鉴了操作系统中的同步机制[36]，采用信号量机制来解决。信号量机制主要用于控制多个线程或进程对共享资源的访问。而信号量本身是一个整数值，用来表示资源的数量。对于工作节点而言，信号量机制即允许同时处理的并发请求数，而多个进程则可以理解为调用工作节点的请求。在设计并发请求的处理逻辑时，通过维护一个信号量计数，可以有效地控制并发访问资源的数量，从而防止系统过载和可能的服务拒绝（Denial of Service, DoS）攻击。下面是如何在处理用户请求的场景中应用信号量的具体方法：

（1）**初始化信号量：**在系统启动时，根据工作节点的处理能力，初始化一个信号量，其值设定为该节点能同时处理的最大请求数。

（2）**请求处理：**当用户请求到达时，系统首先执行 P 操作（等待），尝试将信号量的值减 1。如果信号量的当前值大于 0，则请求可以被处理；如果信号量的值为 0，表示所有工作节点都在忙碌中，该请求将被放入等待队列或提示用户稍后再试。

（3）**请求完成：**一旦请求被成功处理，执行 V 操作（信号），将信号量的值增加 1，表示一个工作单元已释放，其他正在等待的请求可以继续被处理。

3.1.3 用户接口层的设计与实现

用户接口层是系统中直接与用户进行交互的部分，负责接收用户指令并完成完成对话功能。如 §3.1.2中所述，openai 接口作为标准实现，本系统中的 API 服务器也以 openai 为标准。而在对话或补全功能中，除去用户输入这一信息，还有一些较为关键的参数决定了模型如何做出回应，这些参数设计到模型在推理时所涉及到的预测方法。表3.2是关于这些参数的介绍。

表 3.2 推理时涉及到的预测方法参数介绍

参数名称	描述
`do_sample`	若为 True，采用随机采样生成文本；若为 False，使用贪婪解码。
`num_beams`	指定束搜索策略的束数量，数值越大，搜索空间越广，计算代价越高。
`num_beam_groups`	将束分组以提高束间多样性，每组独立搜索后选择最佳输出。
`penalty_alpha`	在对比搜索中平衡模型置信度与新表达方式的探索。
`use_cache`	若为 True，利用之前计算的键值对加速解码，减少计算资源消耗。

在接收到用户请求之后，推理主要调用模型的 generate 接口，这一接口已经在 Transformers[37]库中实现，但在输入输出长度较长的情况下，generate 消耗的时间很长，会导致用户在这段时间内无法看到系统响应，因此需要设计流式响应接口 generate_stream，其原理是在模型产生 token 时，能够将 token 及时返回给用户。在模型推理过程中单独启动一个线程负责 token 的发送，同时在 api 服务器中产生流式响应返回给用户。

设 T 为生成总的 token 数量，t 为当前时间步的 token。在每一时间步 t，模型 M 生成一个新的 token x_t。模型生成 x_t 的时间记为 Δt。可以将模型的推理过程定义为一个序列生

成函数：

$$M(t) \to x_t, \quad t \in \{1, 2, \ldots, T\}$$

同时，设 S 为一个独立的发送线程，负责将生成的 token x_t 发送给用户。发送操作的时间很短，可以近似认为是瞬时的，即：

$$S(x_t) \to \text{send } x_t$$

对于流式响应接口 `generate_stream`，可以表示为一个并行处理过程，其中模型 M 和线程 S 同时运行：

$$\text{For each } t \in \{1, 2, \ldots, T\}:$$

1. $M(t) \to x_t$
2. $S(x_t) \to \text{send } x_t$

这里的关键是 M 和 S 是并行运行的，M 在生成下一个 token 的同时，S 负责将已生成的 token x_t 发送给用户，这样用户可以即时看到响应，而不需要等待所有 token 都生成完毕。该接口的结束条件是预测至最大长度或者遇到提取设定好的停止标志。这种设计大幅减少了用户等待的时间，提高了系统的响应速度和用户体验。

因此，在启动控制器、工作节点以及 api 服务器后，系统可以响应用户的 openai 风格请求，示例如图3-2所示：

```
(base) [root@spr01 ~]# curl http://localhost:8000/v1/chat/completions \
>   -H "Content-Type: application/json" \
>   -d '{
>     "model": "Llama-2-sym_int4_quant",
>     "messages": [{"role": "user", "content": "Hello! What is your name?"}]
>   }' | jq
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   518  100   399  100   119     94     28  0:00:04  0:00:04 --:--:--   123
{
  "id": "chatcmpl-aTEhhCTgxEhtBqsoJZ5guc",
  "object": "chat.completion",
  "created": 1716735479,
  "model": "Llama-2-sym_int4_quant",
  "choices": [
    {
      "index": 0,
      "message": {
        "role": "assistant",
        "content": " Hello! My name is LLaMA, I'm a large language model trained by a team of researcher at Meta AI. Unterscheidung. ☺"
      },
      "finish_reason": "stop"
    }
  ],
```

图 3-2 api 服务器响应示例

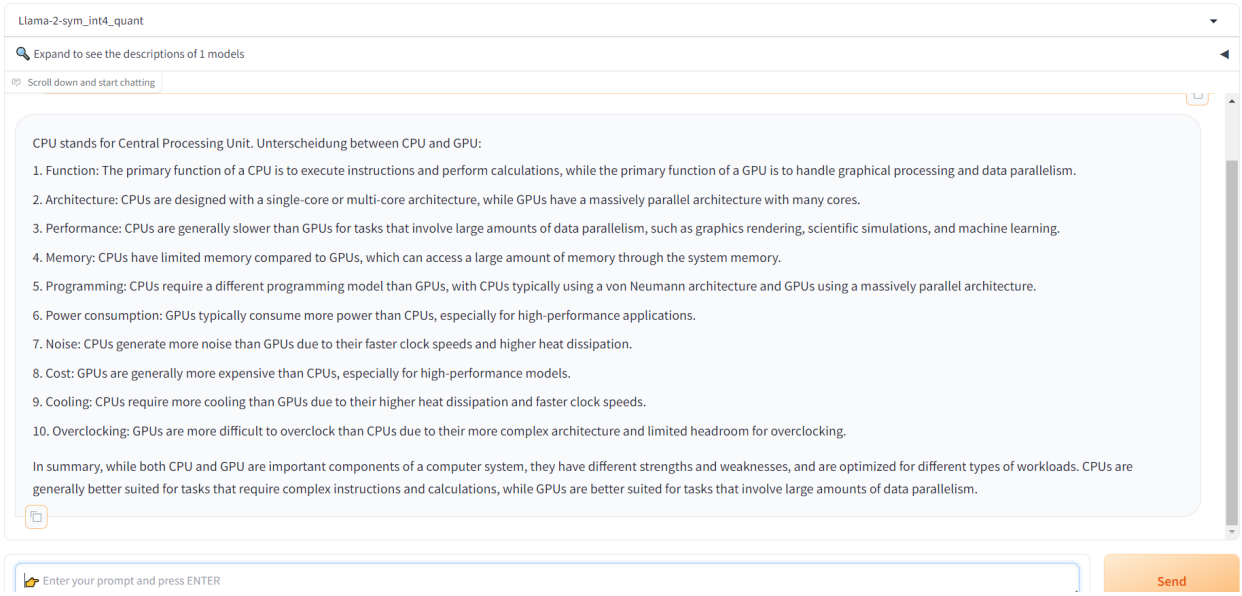


图 3-3 图形用户界面

关于图形化界面的设计，将在 api 服务器的基础上，采用 Gradio 库来实现。Gradio 旨在帮助开发者快速创建简洁、可分享的网页界面，以便与机器学习模型或任何 Python 脚本进行交互。这一点非常符合聊天机器人系统的需求。图**??**为该系统的图形用户界面，用户可以通过 ui 与量化后的 Llama2 模型进行对话。

3.2 低比特量化实现

模型的量化是压缩模型体积，使得大语言模型在资源受限设备上运行的基础，也是本系统的核心所在，本文主要以 Llama-2 模型为参考，讨论 int4 量化的实现，并将量化方法集成至工作节点。

3.2.1 技术基础

随着 Transformer[5] 及 MoE[38] 架构的发展，模型的尺寸越来越大，对计算资源的需求也越大，一般的个人设备很难满足需求，这就需要对模型进行压缩以降低模型部署的成本。模型量化的本质是将浮点数类型的权重（float32 或 float16）映射到 int 型的离散值数据范围内。因为 int 型整数占用更小，因此可以显著的节约内存，对于为低精度运算有过优化的硬件，模型的推理速度也会大大加快。量化主要有两类方法：PTQ（Post-Training Quantization）和 QAT（Quantization-Aware Training）。PTQ 是一种将已经训练完成的模型权重转换为较低精度格式的技术，其实施过程无需进行再训练。虽然这种方法实施简便，但可能伴随模型性能的潜在降低。相比之下，QAT 在模型的预训练或调优阶段就集成了

量化过程，将权重转化为低比特类型，从而提高了模型的运行性能。然而，QAT 的实施成本较高，且对训练数据的代表性有较高的要求[39]。本文主要探讨的是 PTQ，以开源模型 Llama2 为研究对象。

在大模型训练中，确实常常采用 float 数据类型，特别是 32 位浮点数（float32）。这种数据类型在机器学习和深度学习中广泛使用，因为它提供了足够的精度，同时计算效率较高。float32 数据类型能够表示一个广泛的数值范围，这对于处理各种大小的数据特别重要。此外，浮点数类型允许模型在训练过程中捕捉细微的数值变化，这对于优化算法（如梯度下降）来说是必需的，以确保模型能有效地从数据中学习。然而，使用 32 位浮点数也有其缺点，如较高的内存消耗。因此，为了在性能和内存使用之间取得平衡，一些情况下也会采用 16 位浮点数（float16）来训练模型，被称为半精度训练。半精度训练显著加快了训练速度，并且减少了内存需求，但可能牺牲一定的数值精度。一个标准的浮点数可以用如下公式表示：

$$X_{FP} = (-1)^s 2^{p-b}(1 + \frac{d_1}{2} + \frac{d_2}{2^2} + \ldots + \frac{d_m}{2^m}) \tag{3.1}$$

其中 $s \in \{0, 1\}$ 是符号位。$d_i \in \{0, 1\}$ 是第 i 个尾数位，m 表示尾数位的数量。p 是一个在 $[0, 2^e - 1]$ 范围内的整数，e 表示指数位的数量。b 是一个整数的指数偏移。一个有 j 个指数位和 k 个尾数位的浮点数表示为 FP 格式 E_jM_k。

	float32	float16	INT8	INT4
位宽	32	16	8	4
数据范围	-3.40E38 to 3.40E38	-65505 to 65503	0 to 31	0 to 15

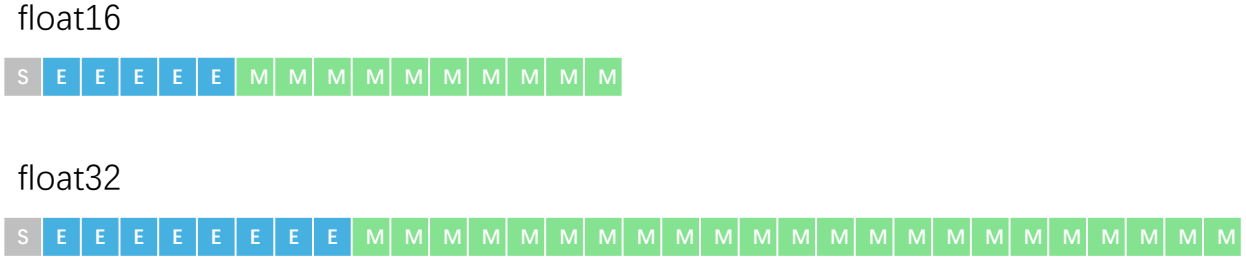


图 3-4 常见数据范围及内存开销示例

但无论是全精度还是半精度都会消耗大量的内存，而整数量化旨在用更少的位宽如 8 位整数 (int8) 或者 4 位整数 (int4) 来表示数据类型，从而减少内存的开销，图3-4展示了常

见数据类型的内存占用和范围。

在 LLM.int8()[15] 中就提到了关于 int8 的混合精度量化技术，总体思路是对大部分权重和激活做 vector-wise 的 8bit 量化。对于离群值则保留 16bit 数据类型。而本研究采用 int4 量化方法进一步压缩了模型的体积。通过 Llama2 的模型结构可以发现，一个 Transformer Block 中总共有 7 个 Linear 层：生成 Q、K、V 的三个 Linear、Attention Block 中最后一个 Linear 和 FeedForward Block 中的 3 个 Linear。虽然这些 Linear 层处理的 tensor 维度不同，但是可以统一使用预定义好的低比特 Linear 层去替换以实现量化。

3.2.2 量化过程

int4 量化的实现和 int8 不同的地方在于使用无符号整数进行表示，即范围是 [0, 15]，因为相比较 int8 量化，数据范围更加有限，无符号整数可以更好地利用这个有限的数据范围。

绝对值量化（absolute maximum Quantization）通过乘以一个比例因子 $scale_{x_{f32}}$ 将输入缩放到 4bit 范围 [0, 15] 内，其中 $scale_{x_{f32}}$ 是 8 除以整个张量中的绝对最大值。这相当于除以无穷范数并乘以 8。因此，对于一个以 32 位浮点数 (float32) 输入矩阵 $X_{f32} \in \mathbb{R}^{s \times h}$，int4 的 absmax 量化可以表示位：

$$X_{i4} = \left\lfloor \frac{8 \cdot X_{f32}}{\underset{ij}{max}(|X_{f32_{ij}}|)} \right\rceil + 8 = \left\lfloor \frac{8}{||X_{f32}||_{\infty}} X_{f32} \right\rceil + 8 = \left\lfloor s_{x_{f32}} X_{f32} \right\rceil + 8 \tag{3.2}$$

其中 $\lfloor \rceil$ 表示四舍五入到最近的整数，加 8 确保数据为无符号整数。

零点量化（Zeropoint quantization）通过归一化动态范围 nd_x 进行缩放，并通过零点 zp_x 进行偏移，将输入分布移至完整范围 [0, 15]。通过这种仿射变换，任何输入张量都将利用数据类型的所有位，从而减少非对称分布的量化偏差。例如，在绝对值量化中，范围 [0, 8] 内的所有值都未被利用，而在零点量化中，完整的 [0, 15] 范围都得到了利用。零点量化由以下方程给出：

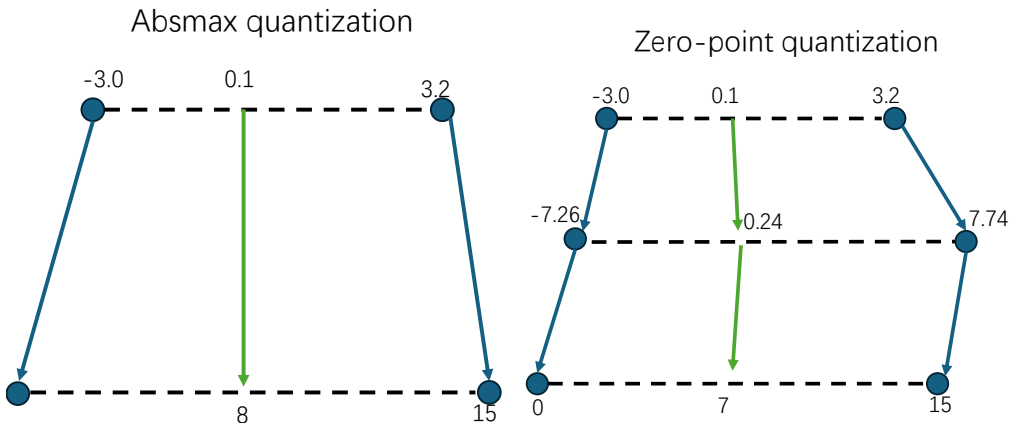
$$ndx_{f32} = \frac{15}{\underset{ij}{max}(X_{f32}^{ij}) - \underset{ij}{min}(X_{f32}^{ij})} \tag{3.3}$$

$$zp_{xi32} = \left\lfloor ndx_{f32} \cdot \underset{ij}{min}(X_{f32}^{ij}) \right\rceil \tag{3.4}$$

$$X_{i4} = \lfloor ndx_{f32} X_{f32} \rceil - zp_{xi32} \tag{3.5}$$

图3-5展示了这两种量化的过程。在绝对值量化3-5(a)中，最小值被映射到了 1，[0, 15]

的数据范围未被有效利用。



(a) 绝对值量化　　(b) 零点量化

图 3-5 量化过程示例

在完成量化之后，需要确定量化对精度造成的影响，而对精度造成的影响主要取决于量化过程的误差，即反量化之后的权重与原始权重的差值，希望这个差值尽可能趋于零。笔者通过绘制反量化权重和原始权重的分布图来展示这一过程的影响。因为以 Llama2[11]模型为研究对象，需要确定其架构特点，选择合适的量化策略。目前主流的 LLMs 处理模型都是基于 Transformer 进行构建的，LLama2 也不例外，而 LLMs 这种生成式的任务是根据给定输入文本序列的上文信息预测下一个单词或 token，Llama2 只需要使用 Transformer Decoder 部分，即在计算时引入 Mask 以确保当前位置只能关注到前面已经生成的内容。

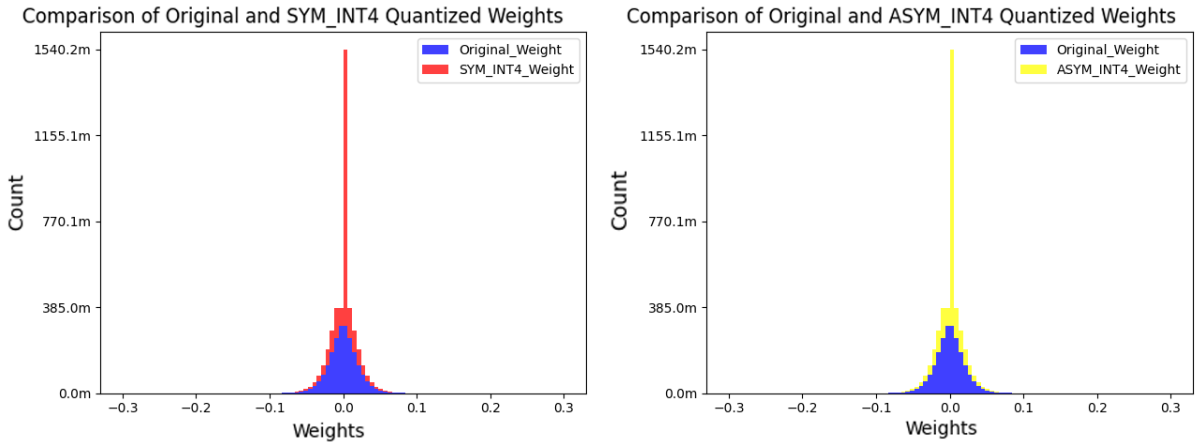
对于被量化的线性层部分，通过可视化的方法绘制反量化权重和原始权重的分布图是评估量化过程是否存在损失的有效手段。如果量化过程存在显著的损失，反量化后的权重分布将与原始权重分布产生较大差异。图3-6绘制了量化和反量化后的权重，左侧为 sym_int4 量化权重比较，右侧为 asym_int4 量化权重比较。两个图表非常相似，0 附近的权重数非常之多，这表示了量化过程中的损失导致反过程无法还原原始值，且在使用绝对值量化时尤其明显。通过这种可视化分析，我们可以得出以下结论：

（1）对称和非对称量化的比较：无论是对称量化还是非对称量化，在反量化后，权重值都集中在 0 附近，显示了量化过程中不可避免的精度损失。特别是在低比特量化时，这种精度损失更加显著。

（2）量化损失的影响：权重分布集中在 0 附近的现象表明，许多原始权重在量化过程中被近似为 0，反量化后无法准确还原。这种现象对模型的性能可能产生负面影响，因为权重的精度下降可能导致模型在推理时的表现不如预期，所以我们在实现量化方法后，有

必要对模型的精度做全面评估，以确定量化方案是否可用。

（3）量化策略的选择：虽然 sym_int4 和 asym_int4 量化策略在一定程度上都造成了损失，但在实际应用中，我们需要根据具体需求和模型特点选择合适的量化策略。例如，对于对称量化，其实现相对简单，但在某些情况下可能不如非对称量化精确。



(a) 绝对值量化　　(b) 零点量化

图 3-6 量化权重损失

实际量化时，零点量化会增加很多的计算量，因此两种方法各有利弊，实际系统中基于公式(3.2)和(3.5)实现了 sym_int4 和 asym_int4 两种量化方法，分别对应了绝对值量化和零点量化。在完成线性层权重的量化后，需要将 float32 格式的输入转化为 int8，进行后续的计算操作，这一点会在 §3.2.3中给出说明。图3-7中给出了这一过程。

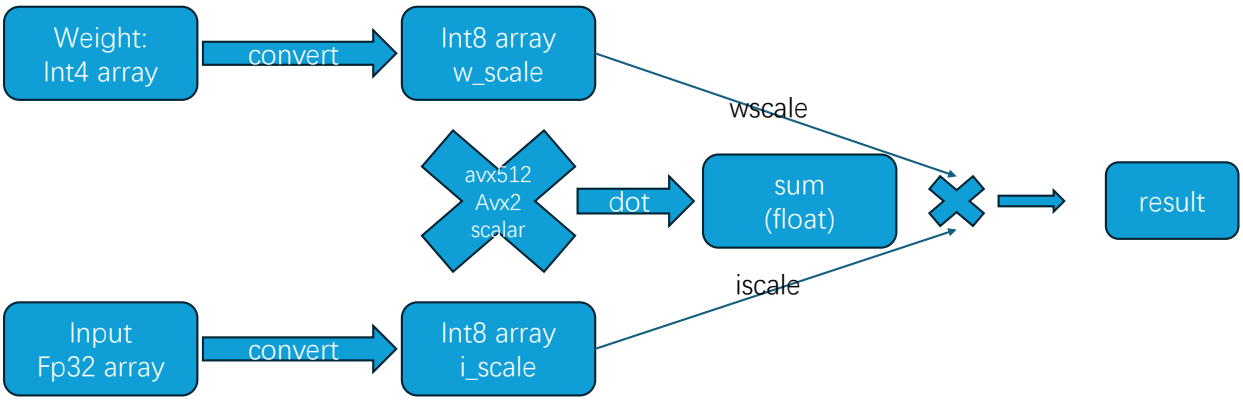


图 3-7 实际点乘过程

3.2.3 量化粒度选择

在量化的过程中，选择不同的范围会计算出不同缩放因子和偏移零点，即量化的粒度。粒度从大到小可以分为逐层量化、逐通道量化，以及逐组量化。

(1) 逐层量化（Per-Tensor Quantization）：这是最简单且粒度最大的一种量化方式。它

将整个网络层视为一个单元，对整层使用一组量化参数。这种方法的实现简单，但可能在精度上有所牺牲，因为它没有针对层内不同部分的特异性进行细致的量化调整。

(2) 逐通道量化（Per-Channel Quantization）：在计算机视觉中，通道通常指的是输入或输出特征图的独立数据集，如输入图形的 RGB 通道，而针对于大语言模型，通道可以理解为多头注意力机制的“头”，这些“头”可以视为并行通道。逐通道量化即以通道为单位进行量化，每个通道独立计算量化参数。因为其粒度更细，往往能够获得更高的精度，但也增加了计算量。具体来说：Per-Token：针对激活数据而言，每行使用一个量化系数。Per-Channel：针对权重而言，每列使用一个量化系数。

(3) 逐组量化（Group-Wise Quantization）：逐组量化的粒度介于逐层量化和逐通道量化之间。逐组量化需要对权重进行分组，以组为单位进行量化。因此，在这种方法中，需要确定组的大小以获得最佳的效果。

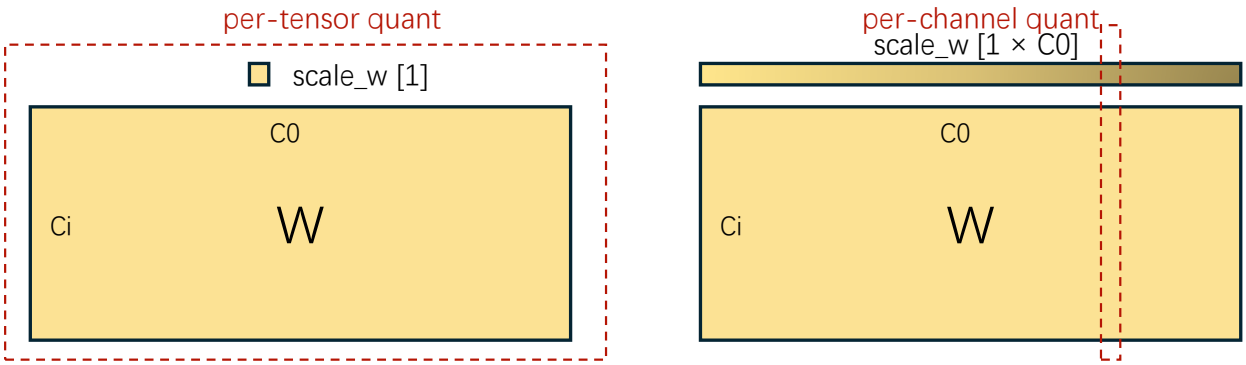


图 3-8 逐层量化和逐通道量化

图3-8给出了逐层量化和逐通道量化的示意，可以看出，逐通道量化相较于逐层量化更加细致，但是计算量也更大。因此，在实际计算中，采用逐组量化的方式，对模型的权重进行分组量化，兼顾计算效率和精度损失。具体过程如伪代码1中所示。由于硬件中并不存在存储 int4 数据的物理单元，实际量化过程中会采用 int8 数据类型的高位和地位来存储两个 int4 数据，这也是整数量化的灵活性，可以通过指针访问到量化后的数据。

在伪代码中有关于 $group\_size$ 的定义，假设其为 k，但实际实现时需要确定它的大小。因为该量化方案基于 Intel Xeon 处理器平台，该处理器支持 Intel AVX-512（高级矢量扩展 512）指令集。AVX-512 指令集包含丰富多样的指令，这些指令能够显著提升计算密集型任务的执行速度。其核心在于将同时处理的向量长度扩展到了 512 位，这意味着每个 AVX-512 寄存器可以同时处理多个数据点。在考虑权重量化时，如果可以将 $group\_size$ 配置为与 AVX-512 寄存器宽度相适应，那么每次操作可以最大化处理的数据量，从而提高

整体计算效率。在使用 INT8 来存储两个 INT4 数据，那么每个 AVX-512 寄存器可以并行计算 64 个 group，因此将 $group\_size$ 设置为 64 可以最大化每次 SIMD (Single Instruction, Multiple Data) 指令的数据处理量。这种设置允许每个 AVX-512 指令加载和存储 64 个 INT4 值对，从而实现高效的数据处理。此外，64 的选择也意味着可以减少因对齐和填充导致的内存浪费。

算法 1: Function to quantize a tensor

Input: tensor, qtype, in_dim (optional)

Output: Quantized tensor in a low bit format

Function `quant_block_weight`(*tensor, qtype, in_dim*)**:**

    group_size ← k
    block_size_in_bytes ← GetTypeSize($qtype$)
    if *GetDataType(tensor)* $\neq$ *float* **then**
        error ”Input tensor must be float32”
    end
    src ← GetPointer($tensor$)
    sum ← GetSumElement($tensor$)
    if *sum_column* mod *group_size* $\neq$ *0* **then**
        error ”Last dim of input tensor must be multipe of group_size”
    end
    dst_size ← $\left(\frac{sum}{group\_size}\right) \times block\_size\_in\_bytes$
    dst_tensor ← createTensor(dst_size)
    dst_tensor ← SetDataType($unsigned\_int8$)
    if *qtype = ”SYM_INT4”* **then**
        return SYMQuantize(src, dst_tensor)
    end
    else
        return ASYMQuantize(src, dst_ensor)
    end

3.2.4 BigDL-LLM 优化

在对模型完成量化的基础上，将使用 BigDL-LLM 库对 Llama2 模型进行进一步优化。因为在量化过程中已经完成了对模型 Attention 层的线性层做了替换，减少了计算时的内存消耗，但模型的其他层如 Decoder 层和 Attention 层，仍有优化的空间。BigDL-LLM 对 Llma2 模型的优化就是通过替换这些层的 forward 函数来实现优化，对于 Attention 层中的 kv-cache 进行优化，减少了内存消耗并通过底层指令集加快了运算速度。

Attention 层的 kv-cache 主要是由为了加速语言模型推理速度所设计的。Decoder only 的语言模型的推理过程共分为两阶段，prompt 输入和逐 token 预测。假设模型的地 i 层 Attention 的权重矩阵为 $W_Q^i, W_K^i, W_V^i, W_O^i$。在 prompt 输入时，假设第 i 层的输入为 $x^i \in \mathbb{R}^{b\times s\times h}$，其中 b 为 batch_size，s 为输入长度，h 为模型维度，依据 transfomers[5] 架构中注意力分数的计算，这一层的输入可以表示为：

$$x_{out}^i = softmax(\frac{x_Q^i {x_K^i}^T}{\sqrt{h}}) \cdot x_V^i \cdot W_O^i + x^i \in \mathbb{R}^{b\times s\times h} \tag{3.6}$$

其中 x_Q^i, x_K^i, x_V^i 由 x^i 与对应权重矩阵相乘得到，即

$$x_K^i = x^i \cdot W_K^i \in \mathbb{R}^{b\times s\times h} \tag{3.7}$$

$$x_V^i = x^i \cdot W_V^i \in \mathbb{R}^{b\times s\times h} \tag{3.8}$$

$$x_Q^i = x^i \cdot W_Q^i \in \mathbb{R}^{b\times s\times h} \tag{3.9}$$

而在第二阶段逐 token 预测时，当前词在第 i 层的向量表示为 $t^i \in \mathbb{R}^{b\times 1\times h}$。该层的输出可以表示为：

$$t_{out}^i = softmax(\frac{t_Q^i {x_K^i}^T}{\sqrt{h}}) \cdot x_V^i \cdot W_O^i + t^i \in \mathbb{R}^{b\times 1\times h} \tag{3.10}$$

从公式3.10中可以看到，x_K^i 和 x_V^i 均需要从之前的 attention 层中传播从而避免大量重复计算。因此，公式3.7和3.8需要被缓存下来，称之为 kv-cache，即:

$$x_K^i \leftarrow Concat(X_K^i, t^i \cdot W_K^i) \in \mathbb{R}^{b\times(s+1)\times h} \tag{3.11}$$

$$x_V^i \leftarrow Concat(X_V^i, t^i \cdot W_V^i) \in \mathbb{R}^{b\times(s+1)\times h} \tag{3.12}$$

kv-cache 的峰值内存消耗可以表示为 $memory_{max} = batch\_size(input_{len} + output_{len}) \times layer\_nums \times h\_models$，其中 $layer\_num$ 为模型层数，h_models 为维度，在某些情况下，kv-cache 的内存占用将超过模型本身。而 BigDL-LLM 中的 Segment kv-cache 策略解决了这一问题。Segment kv-cache 主要分为：预填充阶段和解码阶段。

预填充阶段：在此阶段中，提前计算并存储的 prompt 阶段的 x_K^i 和 x_V^i，这些 key/value 会存储在当前计算设备，避免因数据移动造成的开销。

解码阶段：在解码阶段会预先分配两个缓冲区，一个用于存储预测中产生的 key，一个用于存储 value。这些缓存区会记录索引信息。当使用索引来访问缓冲区信息时，无需再进行重排序或者合并，减少了计算开销。

3.3 容器化及云端部署方案

本小节将介绍如何将系统进行容器化及云端部署，同时集成 Intel TDX 硬件加密技术进行端到端隐私保护。

3.3.1 Docker 容器化实现

容器化是云端部署的基础。当谈及容器化时，Docker 是一个非常流行的解决方案。Docker 是一个开源平台，专门用于应用程序的容器化开发、交付和运行。通过将应用程序及其依赖项打包成一个容器，开发者可以轻松地在任何支持 Docker 的环境中部署这些应用程序。与虚拟机相比更加的轻量化，多个 Docker 容器可以共享宿主机的内核，不同容器中可以添加独立的依赖以支持应用程序的运行。对于本系统来说，由于其在本地部署时涉及到三个组件，即控制器、工作节点和用户接口层, 因此在设计 Docker 容器化方案时，需要考虑各个组件之间的通信。此外，因为模型文件的体积普遍比较大，因此需要考虑模型文件的存储和容器的加载方式。

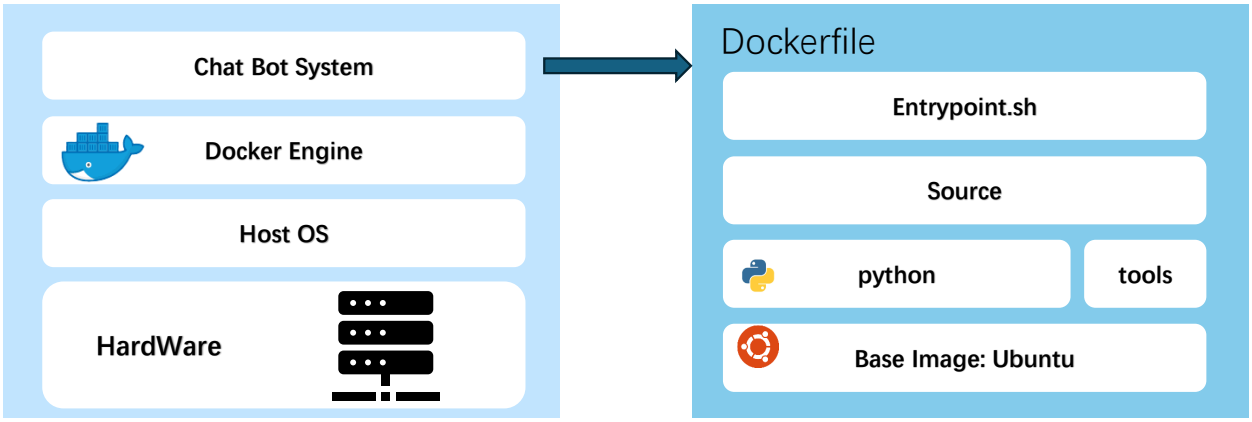


图 3-9 Docker 容器设计架构

在构建 Docker 容器时，需要编写 Dockerfile，即制定容器的基础镜像、相关依赖以及入口程序等。因为该系统依赖于 numactl 等系统工具，因此采用 Ubuntu 作为基础镜像，在安装相关依赖后打包系统源码。入口程序指定了容器在开始运行时的行为，因此可以通过入口程序来控制组件的运行和通信，所以在设计 Docker 容器时只设计一个镜像即可，通

过入口程序来控制容器的行为。图3-9为该系统的容器设计架构。

对于容器间的通信，可以在入口程序中为不同组件分配默认的 Ip 地址和端口，同时在启动时通过参数指定组件来实现。从系统的可扩展性和容器体积最小化的角度考虑，容器可以通过共享文件的方式访问模型文件。在 Docker 中有两种机制，以实现容器间的模型文件访问，即 Volume 和 Bind Mount。Volume 是一种由 Docker 管理的持久化数据方式，将宿主机上的特定目录映射到容器内部，可以实现持久化存储，并且能够被多个容器共享。相比之下，Bind Mount 直接将宿主机上的文件或目录挂载到容器中，赋予容器对宿主机文件系统的直接访问权限。综合考虑可扩展性和容器体积最小化的角度，使用 Volume 来共享模型文件，因为它更加便于管理和维护，提供了持久化存储，并且支持多种文件系统，为云端部署提供了更多的选择。图3-10展示了入口程序的处理过程。

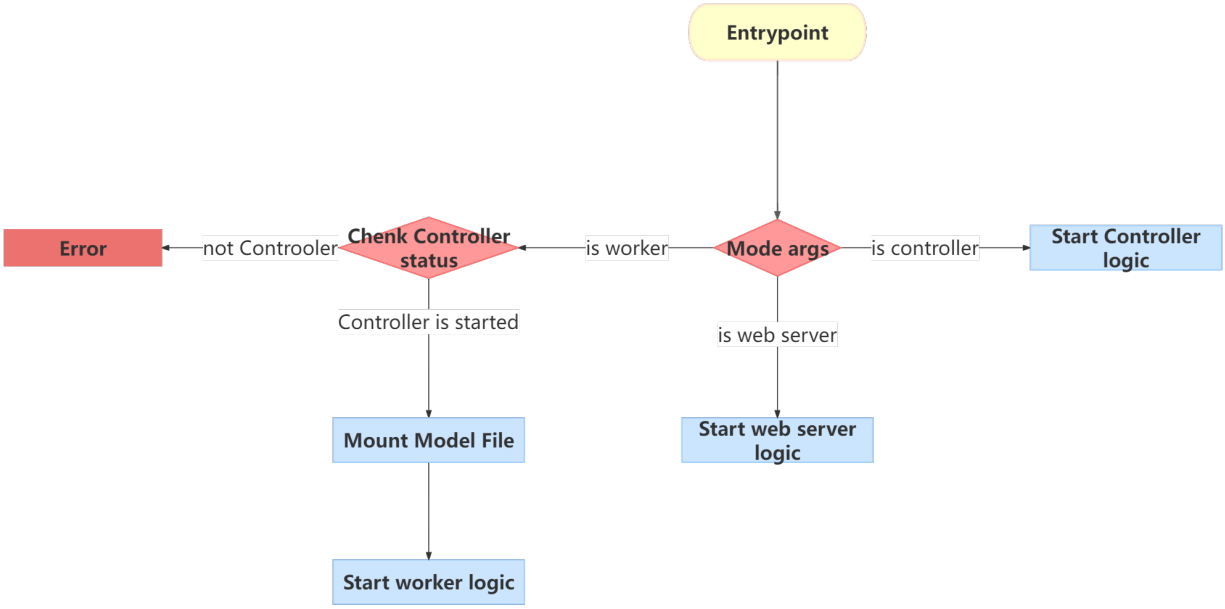


图 3-10 入口程序运行流程

3.3.2 Intel TDX 集成

Intel® Trust Domain Extensions（Intel® TDX）是英特尔推出的最新保密计算技术。它是一种基于硬件的可信执行环境（TEE），目的是通过创建硬件隔离的虚拟机（VM），以保护敏感数据和应用程序免受未经授权的访问。Intel TDX 依赖于 CPU 中的 TDX 模块来实现，这个软件模块运行在新的 CPU 安全仲裁模式（SEAM）[27]下，作为一个对等的虚拟机管理器（VMM），支持使用现有的虚拟化基础设施进入和退出信任域。Intel TDX 使用 SEAM、客户物理地址（GPA）中的共享位、安全扩展页表（EPT）、物理地址元数据表、Intel® Total

Memory Encryption - Multi-Key（Intel® TME-MK）[40]和远程证明等架构元素。Intel TDX 确保数据的完整性、保密性和真实性，从而使工程师和技术专业人员能够创建和维护安全系统，增强了对虚拟化环境的信任。

机密容器（CoCo）则是云原生计算基金会（CNCF）的一个沙盒项目，旨在通过利用各种硬件平台和软件技术实现云原生的保密计算。Intel® TDX 机密容器解决方案基于 CoCo，并在两个方面实现了许多功能。第一个方面是易于使用。对于应用程序开发人员，可以在 TDX 机密容器中运行未经修改的容器镜像。对于 Kubernetes 运维人员/管理员，可以使用标准的 Kubernetes 工具安装整个解决方案并管理 Kubernetes 集群。第二个方面是提供了强大的安全性，通过为敏感和高价值的应用程序、数据和模型提供端到端的保护。

因此对于完成容器化的应用程序，选择 Intel TDX CoCo（Confidential Containers）作为解决方案是最简单易用的，同时也符合聊天机器人系统隐私保护的需求。传统的容器技术使用的运行时环境是 runc，即通过 Linux 内核中的 namespaces 和 cgroups 实现进程隔离，但是进程之间是共享系统内核的，所以也存在较大被攻击的可能性。而 Kata 容器则是基于硬件虚拟化技术（如 Intel VT-x) 的方案，基于硬件虚拟化技术为不同的容器创建隔离的环境，这样保证即使在某一容器受到攻击的情况下，其他容器还能正常运行。Intel TDX CoCo 则利用了 TDX 硬件加密技术，进一步保证了在云环境中敏感数据和工作负载的安全性。图3-11展示了 TDX CoCo 的架构。

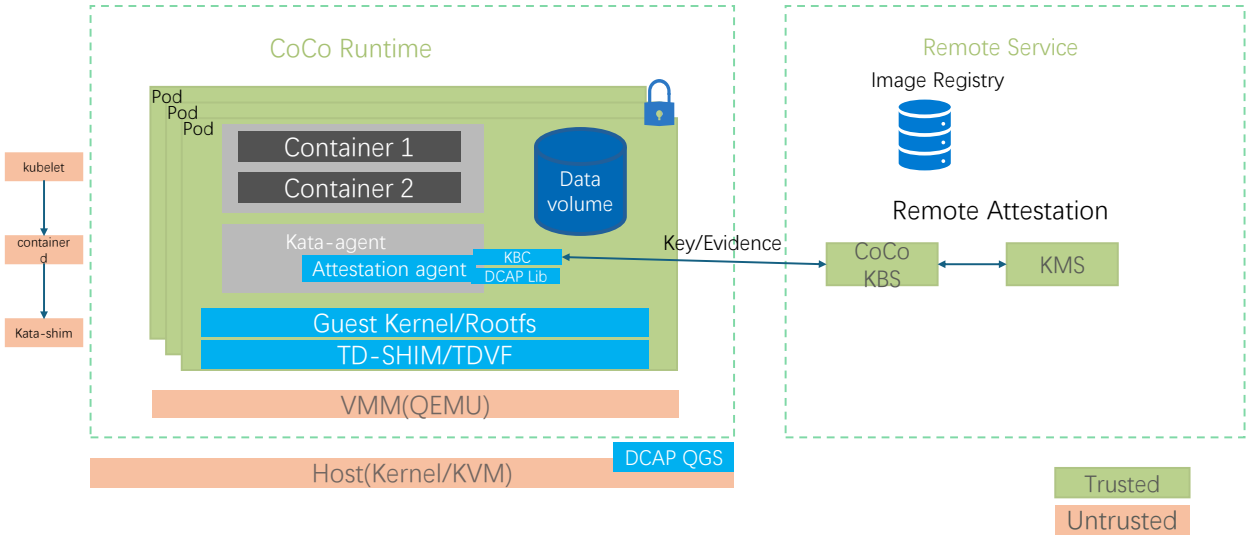


图 3-11 Confidential Container 架构图

其中绿色属于可信域。加密的容器镜像将会从远程拉取并在 TDX 执行环境中解密，所有需要的信息包括容器解密密钥、运行时策略等都只能在远程证明（Attestation）完成后获

得，其中镜像的加密通过 skopeo 等开源工具实现。而远程证明架构则依赖于 Intel TDX 及相关的底层技术，包括 DCAP（Data Center Attestation Primitives）、QGS（Quote Generation Service）[41] 等工具，在创建 Trust Domain 时，TDX 会将所有关键代码和数据适当测量并记录，之后生成 TDREPORT；在获得 TDREPORT 后，会发送给 QGS 并生成 Quote，Quote 包含了 TDREPORT 和签名，用来向外部验证者证明 TD 的安全状态和完整性；CoCo KeyBroker Service 则是在信任方（Relying Party）部署的远程服务，在对 Quote 进行验证后，将从 KMS 密钥管理服务中释放容器密钥、API 访问密钥等敏感数据。在实际系统运行时，只需要修改 kubernetes 的 runtimeclass 为 TDX CoCo 环境，即可实现将镜像安全地拉取到可信域中运行，从而实现无感集成。用户使用过程中，会启动 kata container、Attestation agent 等组件，自动进行远程认证以及解密容器、开启功能接口等流程。

3.3.3 云端部署方案

kubernetes 作为最流行的容器编排工具，在云端部署时是最常见的选择，在完成容器化的基础上，只需借助 k8s 即可完成云端部署。借助 kubernetes 所提供的管理能力，可以实现本系统在云端部署的同时保证可扩展性以及负载均衡。根据 §3.3.2提到的技术原理，需要根据系统内核安装特定版本的 TDX CoCo 相关工具及依赖。在完成安装后，只需要指定 Pod（k8s 中的基本运行单元）的运行时环境为 CoCo 环境，即可完成云端环境中的隐私保护。架构如图3-12所示。

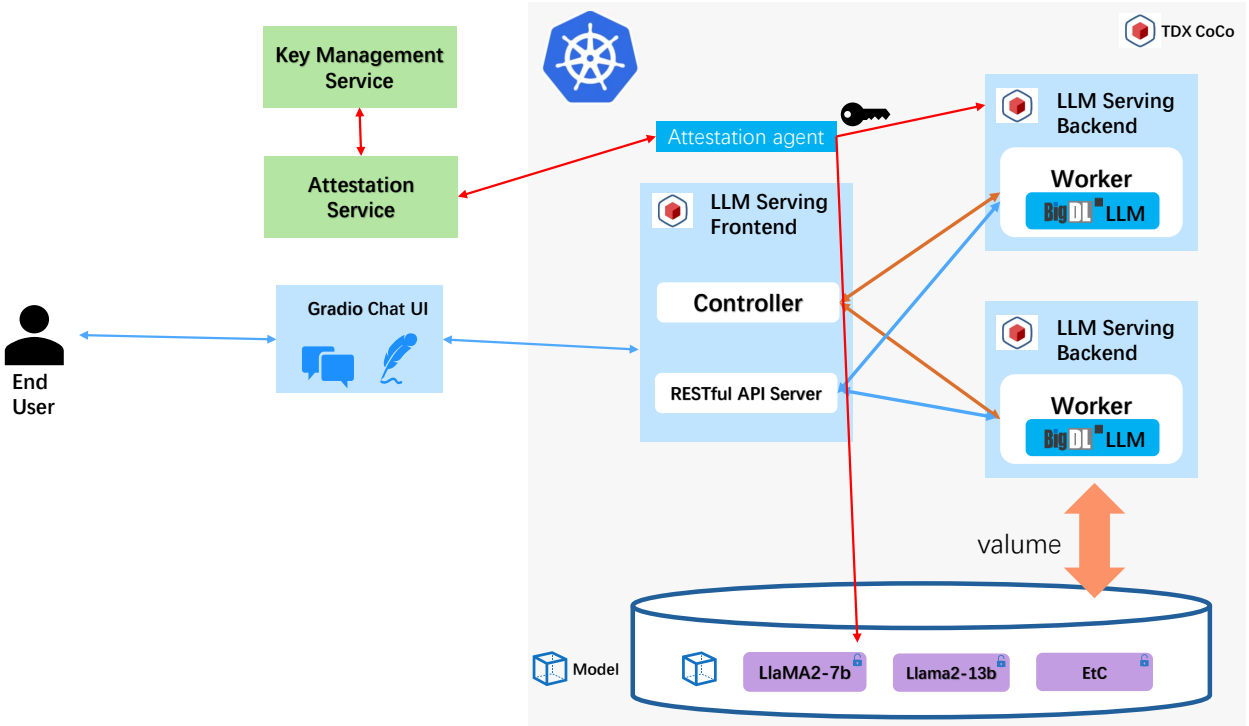


图 3-12 云端部署架构图

根据 §3.3.2中提到的相关技术，当 kubernetes 从镜像仓库中拉取容器后，Confidential Container 会自动创建可信任容器的运行时环境，Attestation Agent 会搜集相应证据并最终通过 QGS 生成 Quote，Quote 包含了当前环境中验证完整性和一致性的信息。用户需要使用对话系统之前，需要发起 Attestation 请求后，此时 attestation agent 会将 quote 发送给远程的 attestation 服务进行验证，成功验证后将释放用于解密容器镜像及 data value 中的模型文件等的密钥。在解密之后，系统开始运行。笔者在 §4.3中给出了关于 TDX CoCo 环境中所带来的额外性能开销。

第四章 实验分析

本章中基于搭载 Intel SPR（Sapphire Rapids）架构 Xeon 处理器的服务器对聊天机器人系统进行评估。笔者首先搜集了 Llama2 家族模型的各项基础指标，包括模型的体积，内存占用、加载时间等指标，来评估量化方法; 之后对模型进行了能力测试，使用 C-Eval[42]数据集，搜集了模型的能力指标；最后在 k8s 集群中部署系统，进行一系列的性能测试，对比了裸机和 TDX 环境下的性能差异。

4.1 模型基础指标评估

首先，笔者针对系统中所实现的两种 int4 量化方法和 transformers[37]库中所实现的 int8 量化对 Llama-2-7b 模型和 Llama-2-13b 模型的内存占用做了对比，如表4.1所示，从中可以看到，在对 Llama-2-7b 和 Llama-2-13b 模型应用不同的量化技术后，内存占用有显著变化。在量化前，Llama-2-7b 和 Llama-2-13b 模型的内存占用分别为 25833.02MB 和 49731.59MB。当采用对称 INT4 量化技术时，这两个模型的内存占用分别下降到 3976.43MB 和 7217.72MB，约为原内存占用的 15.4% 和 14.5%。而在采用非对称 INT4 量化时，内存占用稍有增加，分别是 4173.34MB 和 7600.73MB，仍然远低于未量化的内存占用。此外，使用 INT8 量化技术后，内存占用也得到了大幅度减少，分别降至 7323.84MB 和 13728.85MB，约占未量化时的 28.3% 和 27.6%。总体来看，INT4 量化在减少内存占用方面表现更为优秀，尤其是 SYM_INT4 量化，显示出了最佳的内存节省效果，表明经过量化后的模型将极大程度的减少内存占用。

表 4.1 Llama2 内存占用对比

模型类型	内存占用 (MB)	
	Llama-2-7b	Llama-2-13b
Standard	25833.02	49731.59
Symmetric INT4	3976.43	7217.72
Asymmetric INT4	4173.34	7600.73
INT8	7323.84	13728.85

此外，模型的加载时间也是一个重要指标，笔者同样对 Llama-2-7b 和 Llama-2-13b 进行统计，不同量化方法的加载时间如图4-1所示。从图中可以看到，不同量化方法对 Llama-2-7b 和 Llama-2-13b 模型的影响也颇为显著。未量化的 Llama-2-7b 和 Llama-2-13b 模型的加载时间分别为 30.1276 秒和 64.7866 秒。对比之下，采用 SYM_INT4 量化后，Llama-2-7b 的

加载时间显著降低至 1.68288 秒，减少了近 95%；而 Llama-2-13b 的加载时间也降至 9.57686 秒，减少了约 85%。这一显著的加载时间减少表明 SYM_INT4 量化不仅大幅降低了内存占用，而且极大地提高了模型的加载效率，而 ASYM_INT4 量化则略慢于 SYM_INT4，说明 ASYM_INT4 的量化过程计算量较大。相比之下，使用 INT8 量化技术后，Llama-2-7b 和 Llama-2-13b 的加载时间有所增加，分别为 34.3865 秒和 73.7418 秒，已经超出未量化模型的两倍，这表明 INT8 量化过程增加的计算量已经远远超出原有模型的加载消耗。综合考虑内存占用和加载时间两个指标，SYM_INT4 量化技术在效率和资源消耗上均表现最佳，为模型部署提供了一种高效的优化路径。

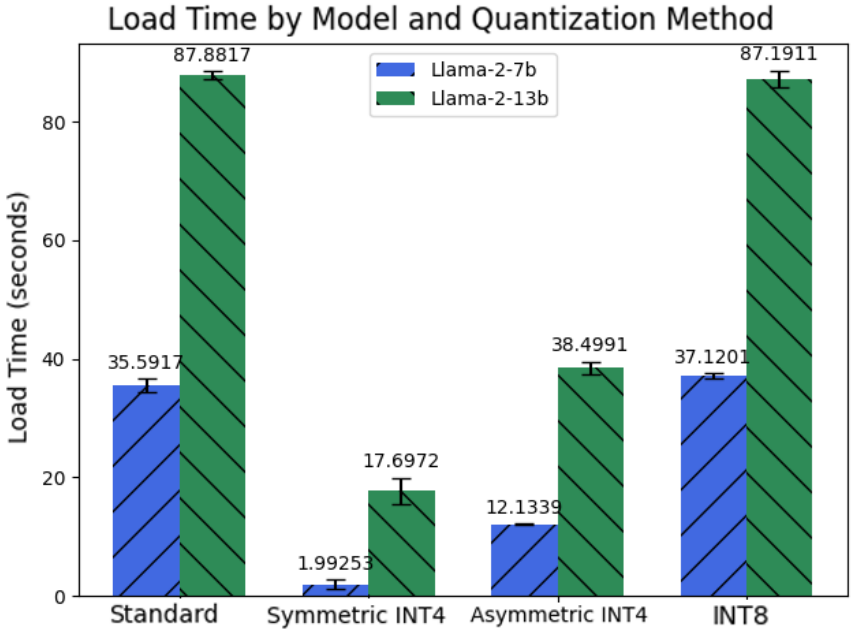


图 4-1 模型加载时间

最后，因为大语言模型的目的是提供对话、补全功能，因此需要评价其输出的效果，笔者选择使用困惑（perplexity）作为评价指标。困惑度是一种常用的语言模型评估指标，它量化了模型在预测序列中下一个词汇时的不确定性程度。困惑度的基本推导为：

语言模型预测的是给定前文之后单词出现的概率。假设有一个词序列 $W = w_1, w_2, \ldots, w_N$，语言模型会给出该序列的概率是 $P(W)$。利用链式规则，这个概率可以表示为各个单词在给定前文情况下出现的概率的乘积：

$$P(W) = P(w_1) \times P(w_2 \mid w_1) \times \ldots \times P(w_N \mid w_1, w_2, \ldots, w_{N-1}) \tag{4.1}$$

而困惑度是语言模型预测整个测试集词序列的概率的倒数的几何平均。对于整个词序列，

困惑度 $PP(W)$ 定义为

$$PP(W) = P(w_1, w_2, \ldots, W_N)^{-\frac{1}{N}} \tag{4.2}$$

可以将概率取对数来简化计算，通常使用自然对数：

$$\log P(W) = \log P(w_1) + \log P(w_2 \mid w_1) + \ldots + \log P(w_N \mid w_1, w_2, \ldots, w_{N-1}) \tag{4.3}$$

因此，困惑度可以表示为：

$$PP(W) = \exp\left(-\frac{1}{N}\sum_{i=1}^{N} \log P(w_i \mid w_1, \ldots, w_{i-1})\right) \tag{4.4}$$

在这种比较中，通常假设分数越低越好。在实际计算困惑度时，笔者选用开源数据集 LongBench[43]，其涵盖了单文档 QA、多文档 QA、摘要、合成任务和代码补全等关键的长文本应用场景。因为 Llama-2 是以英语训练的大预言模型，故选择 LongBench 中的英文任务进行测试。实际计算时，笔者通过将十四个英文数据集进行结合，对所有数据迭代更新后得出困惑度。Llama-2-7b 的困惑度如表4.2所示。从数据中可以观察到，Llama 模型在应用不同量化方法时，困惑度有所变化。例如，Llama-2-7b 在未量化时的困惑度为 5.2709，而在使用 sym_int4 位量化时增至 5.5676，这表明困惑度上升了约 5.63%。尽管性能有所下降，但鉴于量化可以显著减小模型大小并提高计算效率，这种困惑度的增加在许多应用场景中是可以接受的。同样，在 Llama-2-13b 模型中，从未量化的 4.8999 增至 sym_int4 的 4.9445，困惑度增加了约 0.91%，再次显示出在保持相对较低性能损失的同时，实现效率的提升是可行的。而且 int4 量化和 int8 量化的困惑度差异并不明显，证明在使用 int4 量化后模型仍能维持在一个合理的性能范围内，支持效率与效果之间的平衡。

表 4.2 Perplexity of Llama Models with Different Precision Settings

Model	**Precision**	**Perplexity**
Llama-2-7b	standard	5.2709
Llama-2-7b	sym_int8	5.3681
Llama-2-7b	sym_int4	5.5676
Llama-2-7b	asym_int4	5.5297
Llama-2-13b	standard	4.8999
Llama-2-13b	sym_int8	4.9168
Llama-2-13b	sym_int4	4.9445
Llama-2-13b	asym_int4	5.0289

4.2 模型能力测试

§4.1对模型的基础指标进行了测试，验证了本系统的量化方法能在显著压缩模型大小的同时保持其准确性，证明即使在资源受限的环境下，模型的效能也可以得到保证。然而，模型的量化可能影响其处理复杂计算任务的能力，因此有必要对模型进行更深入的能力评估以确定量化的实际影响。

首先，为了观察模型的对话是否合理，笔者设置了两个中英文的简单输入，观察到sym_int4 量化模型的生成结果如表4.3所示，给出的回答是正常的。

表 4.3 Example of SYM_INT4 model output

中文	英文
输入: 讲一个笑话 输出: Why don’t scientists trust atoms? Because they make up everything!	Input: What is AI? Output: Ah, a great question! AI, or Artificial Intelligence, refers to the development of computer systems that can perform tasks that typically require human intelligence.

之后，为了进一步评估量化模型的能力，本研究引入了 C-Eval 评估方法[42]，一个设计用来全面评估模型综合能力的高级工具。C-Eval 覆盖从自然科学到社会科学等广泛领域，确保了评估的全面性和挑战性。例如，在自然科学领域，可以通过评估模型在物理学、化学、生物学等学科中的回答准确性来衡量其对科学知识的掌握情况。通过使用 C-Eval，本研究不仅可以评估量化模型在一般知识领域内的表现，还可以深入分析模型在特定学科领域的细微差别。使用 C-Eval 进行评估的过程中，对量化前后的模型在各个学科领域的表现进行了对比分析。这种对比不仅理解量化对模型性能的具体影响，还揭示了量化可能引入的特定领域的偏差或弱点。通过这种全面的评估，研究者可以发现量化对模型性能的具体影响。例如，某些模型在未量化时可能在某个特定领域表现出色，但在量化后表现可能会有所下降。

图4-2是对 Llama-2-7b 模型的测试结果，测试结果覆盖了六个核心类别：科学技术（STEM）、社会科学、人文学科、其他、困难和平均得分。通过对比分析，可以观察到, 未经过量化的模型在多数类别中都保持稳定领先，sym_int8 的表现和未量化的模型非常接近。相比之下，asym_int4 和 sym_int4 的得分在大多数类别中稍显不足，考虑到量化过程所带来的精度损失，这一结果是符合预期的，总体而言 sym_int4 的表现要优于 asym_int4。这

表明 int4 量化对模型能力造成的影响在可接受范围内。

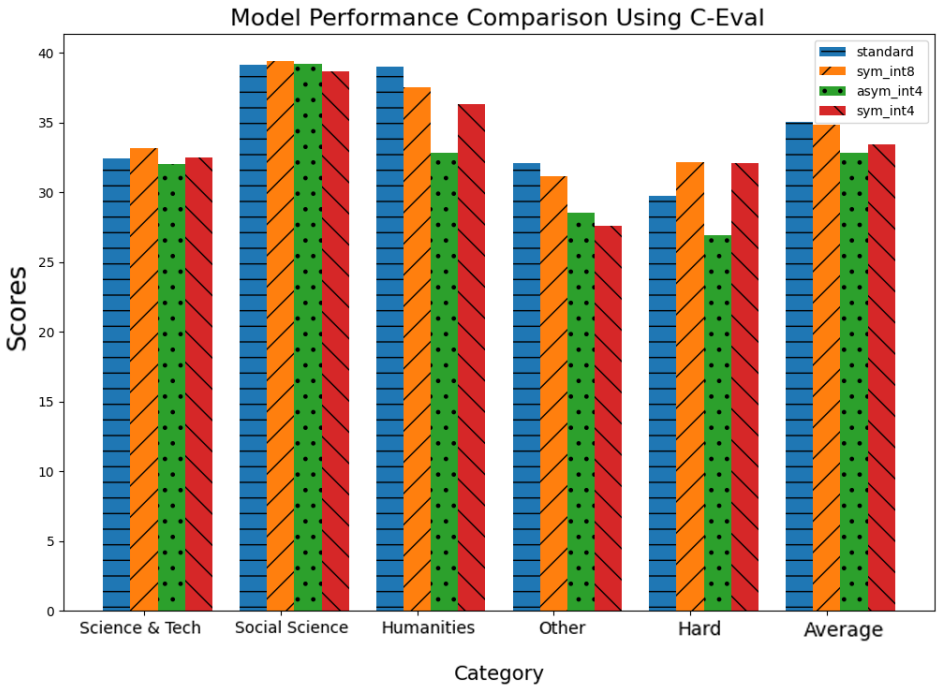


图 4-2 C-Eval 测评分数

4.3 系统性能评估

本节主要对聊天机器人系统在不同输入输出条件下的性能进行了系统评估，通过测量系统的响应时间等关键性能指标，全面反映了系统的效率。此外，本节还根据 §3.3.2 所述，对比了在裸机环境与采用 TDX 隐私防护技术的环境中的系统性能，以评估这种安全技术在保护数据隐私时引入的性能开销。通过这种方法，本研究旨在明确隐私防护措施对聊天机器人系统性能的具体影响，为未来系统设计和优化提供实证基础。

笔者将系统部署在一个有 4 个节点、8 个 pod 规模的 Kubernetes 集群上，集群中的每个节点均为 96 核心的 Intel Xeon Platinum 8468 处理器的高性能服务器。pod 是执行计算的基本单元，其中有两个 pod 分别负责控制器和 API 服务器的运行，其他 pod 将根据具体负载启动相应数量的工作节点来负责模型的推理。此外，该系统已经部署了 TDX CoCo 可执行环境，在指定 pod 的 runtimeclass 为 TDX 之后，系统将运行在隐私保护环境中。模型文件存储于另一台 ftp 服务器中，通过 mount 的方式挂载到工作节点所在的 pod。有一台信任方（Relying Party）远程服务器部署了 CoCo KBS 服务，负责远程证明和密钥发放。

首先，针对单工作节点的不同长度输入输出的响应时间进行分析，使用 Llama-2-7b 模型并选择 sym_int4 量化方法与原始模型对比，针对不同长度的输入输出搜集系统给出的

响应时间，如图4-3所示, 可以看到在不同输入输出序列下量化模型的响应速度远高于原始模型，响应时间约为原始模型的 50%。通过使用 sym_int4 量化方法，我们观察到模型在处理相同长度的输入输出序列时，响应时间显著减少。这种优势主要体现在计算效率的提升上，量化模型由于使用更低比特的表示方式，减少了计算和内存传输的负担，从而加快了模型的推理速度。

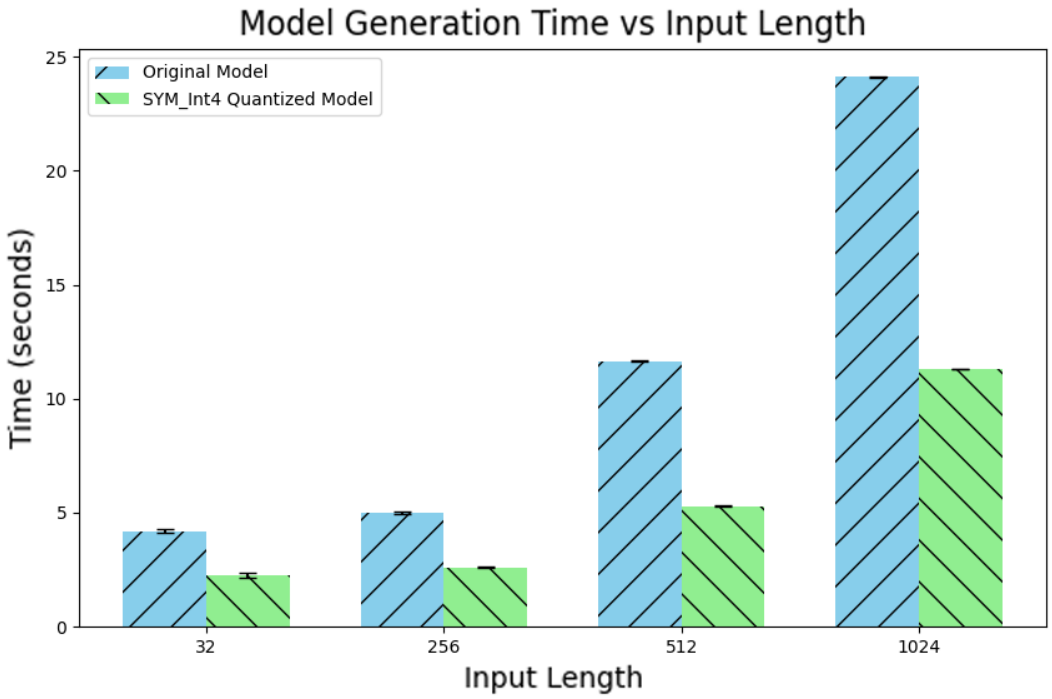


图 4-3 不同输入长度下的系统响应时间

之后，由于模型的输出实际是逐 token 输出的，笔者对系统中的流式接口进行测试，以 1024、512 长度输入为例，统计生成每个 token 的时间，截取前 11 个 token 得到的结果如图4-4、4-5所示，模型产生第一个 token 的时间远大于后续 token 耗时，且后续 token 都趋于稳定，这一现象表明，量化后的模型在初次启动时需要进行“预热”（warm_up），导致第一个 token 的生成时间较长。这是因为系统在首次加载模型数据时，需要更多的时间来预热计算资源，以准备进行高效的计算。一旦模型完成了 warm_up 过程，后续 token 的生成速度就会显著加快并趋于稳定。

这种现象在许多实时处理系统中都非常常见，尤其是在使用深度学习模型进行实时预测或生成时。为了确保系统在部署后的稳定性能，增加 warm_up 步骤是必要的。这不仅可以显著缩短实际使用过程中每个 token 的生成时间，还可以提升整体系统的响应速度和用户体验。

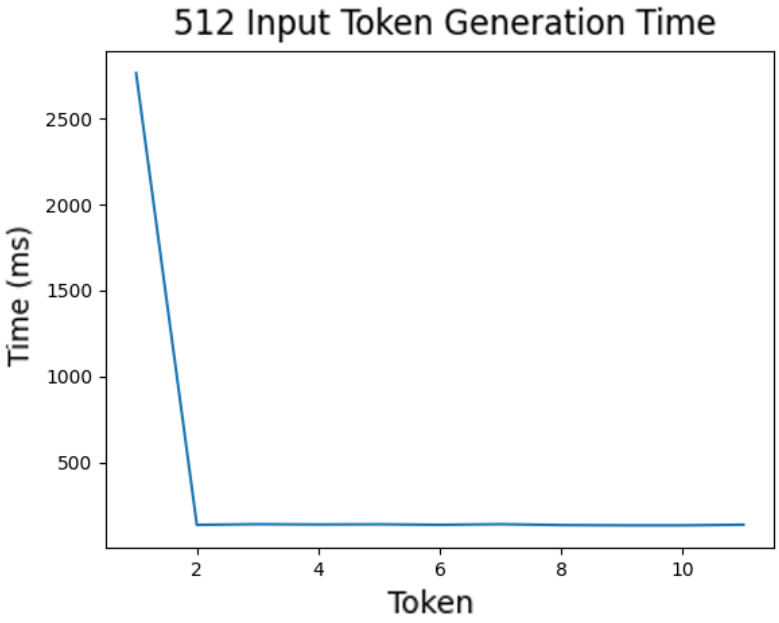


图 4-4 输入长度为 512 时 token 延迟时间

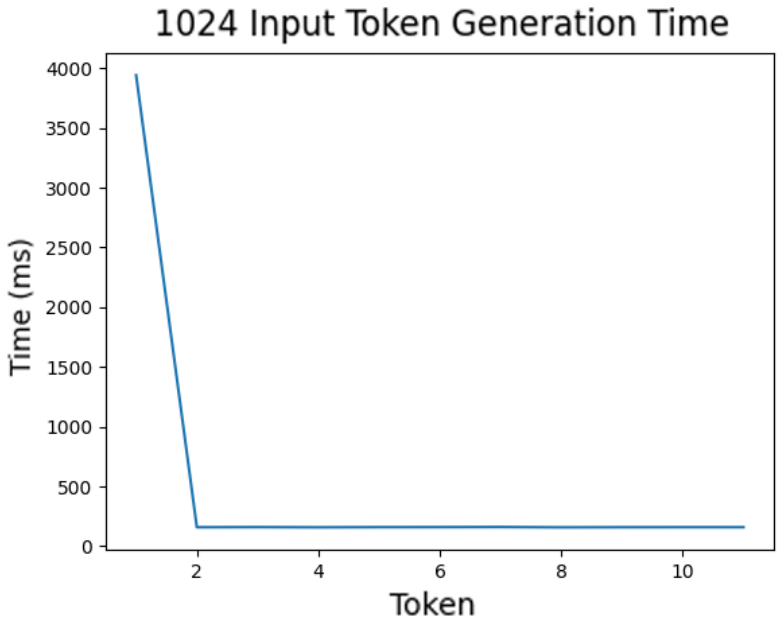


图 4-5 输入长度为 1024 时 token 延迟时间

根据上文得出两个评价响应速度的指标，first token time 以及 next token time，分别代表系统在模型预加载时产生第一个 token 的时间和产生后续 token 时间的平均值。现在结合硬件特性进行性能调优。由于现代的多核服务器均采用 NUMA（Non-Uniform Memory Access）结构，每个 socket 通过内存控制器连接本地内存（local memory），通过 socket 间的高速总线访问属于其他 socket 的远端内存（remote memory）。直接连接的 CPU 核心与内存以及其他外设（如网卡、GPU）组成一个 NUMA 域（或 NUMA 节点）。在同一个域内（域内）的内存访问性能（包括带宽和延迟）通常显著优于跨 NUMA 域（域间）的访问性

能，这种现象被称为 NUMA 效应。当前环境中的 ntel Xeon Platinum 8468 处理器的 numa 拓扑如表4.4所示，共有两个节点，分别对应于编号 0-47、96-143 和 48-95、144-191 的 CPU 核心。

表 4.4 服务器 NUMA 配置信息

NUMA 节点	CPU 范围
0	0-47, 96-143
1	48-95, 144-191

由于服务器启用了超线程，故每个 numa 节点的 48 个物理 cpu 被划分成了 96 个，实际 numa 节点的物理核心数为 48。在一个多 NUMA 系统中，如果进程被迁移到了与创建时不同的 NUMA domain，就可能影响性能。此外，进程迁移时必须暂停，在新的核心上也会不可避免地遇到 cache、分支预测器等组件的冷启动开销，产生性能波动。因此，对于大语言模型这也的计算密集型程序，将线程、进程和 CPU 核心绑定往往能够提升性能。笔者对不同的邦核方式进行了实验，并以系统的 first token time 和 next token time 为指标进行比较。图4-6展示了三种不同邦核方式的性能。

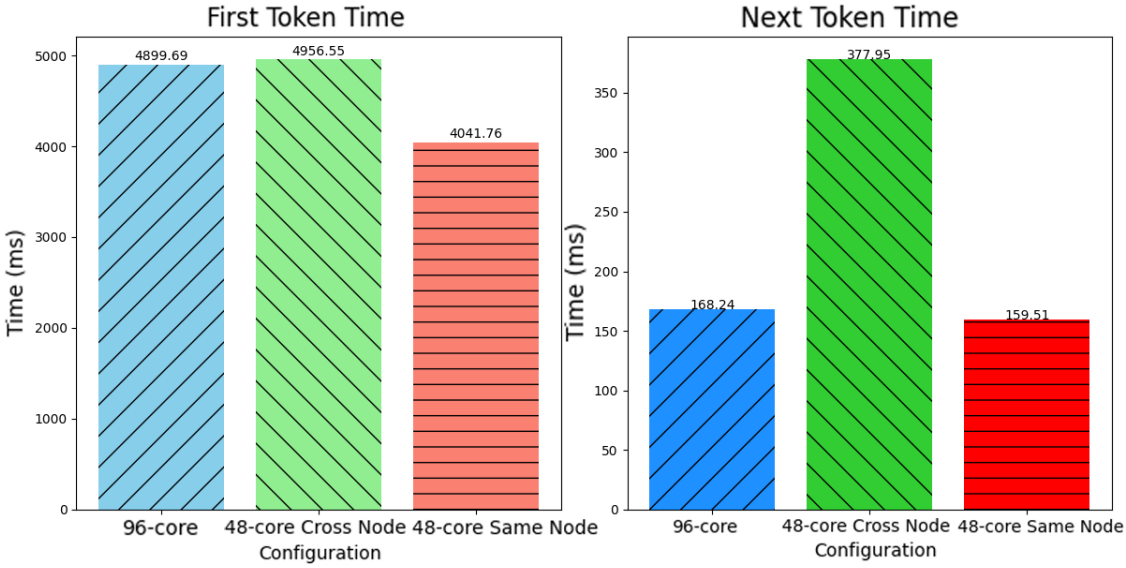


图 4-6 不同绑核方式性能对比

从中可以看到，跨 numa 节点访问内存的绑核方式性能最差，next token time 几乎为其他两种邦核方式的两倍，说明在计算过程中，进程迁移的开销远大于计算本身。对使用全部 96 核心的绑核方式，虽然核心数为同节点绑核的两倍，但是性能依然略差于同节点绑核，这也说明两个 numa 节点同时参与计算并不是最优解。上述实验给出了最优的 cpu 核

心分配策略，即对于每个工作节点所在的 pod，将 cpu 设置为 48 核心能够发挥服务器的最佳性能。

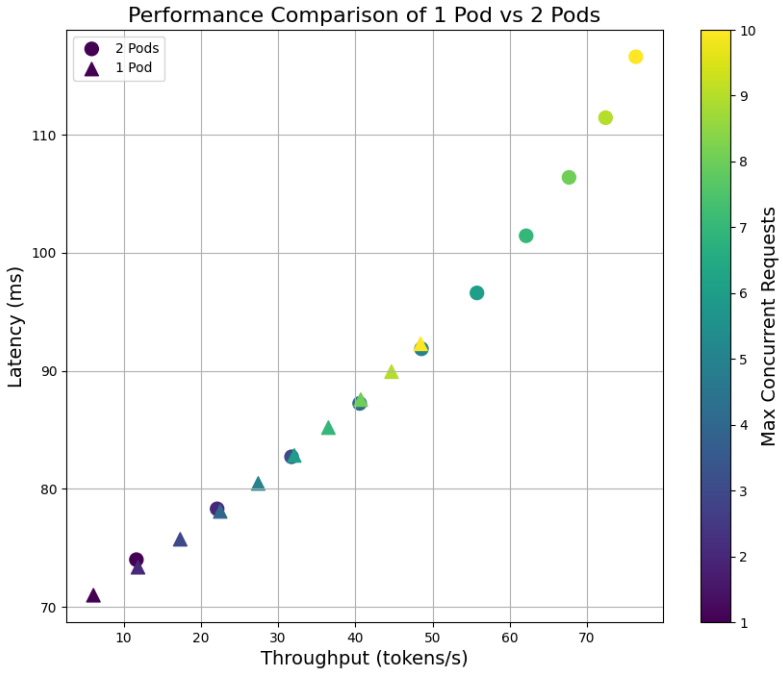


图 4-7 并发场景中系统的性能数据

为了验证和展示系统在 Kubernetes 环境下的负载均衡能力，笔者通过并发测试来评估系统的性能表现。测试将采用多线程并发请求的方法，这可以模拟真实世界中多用户同时访问系统的情况。通过这种方式，能够测量系统在不同负载条件下的响应时间，并进一步分析其扩展和负载均衡的效能。具体实验方法为：针对固定长度的输入样例，设计不同并发度的请求，统计一段时间内系统吞吐量和模型的 next token time 作为性能指标。图4-7统计了在不同并发场景下，系统响应 1000 次请求的平均值，同时对比了工作节点总数为 1 和 2 的情况下系统的吞吐量和 next token time。分析图片可以发现，在单个工作节点的配置下，系统在高并发情况下的表现受限于计算资源，而两个工作节点的配置不仅提高了系统的处理能力，还实现了负载均衡，从而提高了整体的响应速度和稳定性。这说明控制器对工作节点的调度策略能够合理分配计算任务，增加工作节点的方式也可以显著提高系统的吞吐量，减少系统的总体负载，改善用户体验。

最后，笔者将 pod 运行时环境设置为 TDX CoCo，并测量了其性能，表4.5展示了系统在裸机和 TDX 环境中的模型加载时间和响应时间。其输入为 1024 长度的文本序列，输出最大长度限制在 256，共测试了 1000 个请求的平均延迟，工作节点数量均为 1。从表格数据可以看出，虽然 TDX 环境下的加载时间和响应延迟略高于裸机环境，但这种性能下降

在可接受的范围内。尽管在 TDX 环境中，系统的首次延迟和后续延迟都有所增加，但增幅相对较小。例如，首次响应延迟从裸机的 4039.43 毫秒增加到 TDX 的 4422.78 毫秒，增加了约 9.5%；而后续响应延迟仅从 161.46 毫秒增加到 172.65 毫秒，增幅约 6.9%。这表明 TDX CoCo 环境虽然牺牲了一部分性能，但仍然能够在大多数应用场景下维持高效的运行。更重要的是，TDX 环境提供了更加严格的安全保障，能够有效地隔离和保护运行环境，防止数据泄露和其他安全威胁，适用于在不可信的云平台环境中部署系统的场景。

表 4.5 裸机与 TDX 环境性能对比

Category	**Native**	**TDX**
Load Time (s)	1.77435	4.55689
First Latency (ms)	4039.43	4422.78
Next Latency (ms)	161.46	172.65

第五章 总结与展望

5.1 工作总结

本文深入探讨了大语言模型的发展动态，重点研究了其在硬件资源受限的环境中的优化与部署问题并设计与实现了一种基于低比特量化的高性能聊天机器人系统：通过对现有的大语言模型应用场景和商用聊天机器人系统进行全面研究后，分析出对于个人用户和学习者来说，痛点在于资源受限的硬件设备无法支持大语言模型所造成的内存占用和计算资源消耗，进而根据近年来对模型量化技术的研究，提出并实现了基于 Llama2 模型的 int4 低比特量化，显著降低了模型在个人设备上的内存消耗，同时应用了 BigDL-LLM 库中的优化技术，进一步提升了模型在资源受限环境下的执行效率；对于大语言模型应用过程中的隐私泄露问题，本文研究了最新的 Intel TDX 硬件加密技术及基于其的一种实现：TDX Confidential Container，设计了系统的云端部署方案，并应用了隐私保护技术，为数据处理和存储提供了安全的执行环境，有效的防止了隐私泄露和未授权访问，提升了系统的安全性。

本文实现了该系统并通过实验证明上述系统设计的有效性；结果表明，本文所实现的低比特量化技术可以将 Llama2 模型的原始内存占用缩减至原来的 28% 左右，将其加载时间缩减至 3% 至 15%，同时保证其模型能力的损失不超过 20%，具有正常的对话功能；系统对于用户的响应时间也大幅提升，约为量化前的 50%，并且在高并发的场景下，系统能够实现负载均衡，能够有效地调度多个工作节点，并且该系统在实现完整的隐私保护的情况下，只引入了约 6.03% 至 7.15% 额外延迟。值得注意的是，该系统的量化方法虽然以 Llama2 模型为基础实现，但所涉及到的量化方法可以进一步适配到其他开源大语言模型，可以通过进一步开发形成一种普适性的方法。

5.2 工作展望

未来笔者将基于此系统继续改进，以支持更多开源大语言模型的优化。在模型量化的同时，进一步考虑模型剪枝、动态量化等技术，提升性能；除了基于 CPU 的优化外，研究 GPU 底层所存在的高效计算的指令集，尝试在 GPU 上实现模型量化的优化方法，进一步利用 DeepSpeed 等框架，实现模型在多卡环境中的分布式推理；对于系统的部署和优化，未来探索扩展到多模态 LLMs 以提供更丰富的交互体验；对于隐私保护机制，研究在集成 Intel TDX 技术的基础上，是否存在其他潜在的安全漏洞，尝试进一步加强隐私保护。

参考文献

[1] JOUPPI N P, YOUNG C, PATIL N, et al. In-datacenter performance analysis of a tensor processing unit [C]//Proceedings of the 44th annual international symposium on computer architecture. 2017: 1-12.

[2] ABADI M, BARHAM P, CHEN J, et al. {TensorFlow}: a system for {Large-Scale} machine learning[C]// 12th USENIX symposium on operating systems design and implementation (OSDI 16). 2016: 265-283.

[3] PASZKE A, GROSS S, MASSA F, et al. Pytorch: An imperative style, high-performance deep learning library[J]. Advances in neural information processing systems, 2019, 32.

[4] RADFORD A, NARASIMHAN K, SALIMANS T, et al. Improving language understanding by generative pre-training[J]. 2018.

[5] VASWANI A, SHAZEER N, PARMAR N, et al. Attention is all you need[J]. Advances in neural information processing systems, 2017, 30.

[6] DEVLIN J, CHANG M W, LEE K, et al. Bert: Pre-training of deep bidirectional transformers for language understanding[J]. arXiv preprint arXiv:1810.04805, 2018.

[7] BROWN T, MANN B, RYDER N, et al. Language models are few-shot learners[J]. Advances in neural information processing systems, 2020, 33: 1877-1901.

[8] LIU Y, OTT M, GOYAL N, et al. Roberta: A robustly optimized bert pretraining approach[J]. arXiv preprint arXiv:1907.11692, 2019.

[9] LAN Z, CHEN M, GOODMAN S, et al. Albert: A lite bert for self-supervised learning of language representations[J]. arXiv preprint arXiv:1909.11942, 2019.

[10] SANH V, DEBUT L, CHAUMOND J, et al. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter[J]. arXiv preprint arXiv:1910.01108, 2019.

[11] TOUVRON H, MARTIN L, STONE K, et al. Llama 2: Open foundation and fine-tuned chat models[J]. arXiv preprint arXiv:2307.09288, 2023.

[12] WEI J, TAY Y, BOMMASANI R, et al. Emergent abilities of large language models[J]. arXiv preprint arXiv:2206.07682, 2022.

[13] SMITH S, PATWARY M, NORICK B, et al. Using deepspeed and megatron to train megatron-turing nlg 530b, a large-scale generative language model[J]. arXiv preprint arXiv:2201.11990, 2022.

[14] HAN S. Efficient methods and hardware for deep learning[D]. Stanford University, 2017.

[15] DETTMERS T, LEWIS M, BELKADA Y, et al. Gpt3. int8 (): 8-bit matrix multiplication for transformers at scale[J]. Advances in Neural Information Processing Systems, 2022, 35: 30318-30332.

[16] YAO Z, YAZDANI AMINABADI R, ZHANG M, et al. Zeroquant: Efficient and affordable post-training quantization for large-scale transformers[J]. Advances in Neural Information Processing Systems, 2022, 35: 27168-27183.

[17] LIN J, TANG J, TANG H, et al. Awq: Activation-aware weight quantization for llm compression and acceleration[J]. arXiv preprint arXiv:2306.00978, 2023.

[18] XIAO G, LIN J, SEZNEC M, et al. Smoothquant: Accurate and efficient post-training quantization for large language models[C]//International Conference on Machine Learning. PMLR, 2023: 38087-38099.

[19] HU E J, SHEN Y, WALLIS P, et al. Lora: Low-rank adaptation of large language models[J]. arXiv preprint arXiv:2106.09685, 2021.

[20] DETTMERS T, PAGNONI A, HOLTZMAN A, et al. Qlora: Efficient finetuning of quantized llms[J]. Advances in Neural Information Processing Systems, 2024, 36.

[21] STAAB R, VERO M, BALUNOVIĆ M, et al. Beyond memorization: Violating privacy via inference with large language models[J]. arXiv preprint arXiv:2310.07298, 2023.

[22] VU M N, NGUYEN T, JETER T, et al. Analysis of privacy leakage in federated large language models[J]. arXiv preprint arXiv:2403.04784, 2024.

[23] RAFAILOV R, SHARMA A, MITCHELL E, et al. Direct preference optimization: Your language model is secretly a reward model[J]. Advances in Neural Information Processing Systems, 2024, 36.

[24] DAI J, PAN X, SUN R, et al. Safe rlhf: Safe reinforcement learning from human feedback[J]. arXiv preprint arXiv:2310.12773, 2023.

[25] PATIL V, HASE P, BANSAL M. Can sensitive information be deleted from llms? objectives for defending against extraction attacks[J]. arXiv preprint arXiv:2309.17410, 2023.

[26] RUAN Y, DONG H, WANG A, et al. Identifying the risks of lm agents with an lm-emulated sandbox[J]. arXiv preprint arXiv:2309.15817, 2023.

[27] CHENG P C, OZGA W, VALDEZ E, et al. Intel tdx demystified: A top-down approach[J]. ACM Computing Surveys, 2023.

[28] SEV-SNP A. Strengthening vm isolation with integrity protection and more[J]. White Paper, January, 2020, 53: 1450-1465.

[29] GAO J, GALLEY M, LI L. Neural approaches to conversational ai: Question answering, task-oriented dialogues and social chatbots[M]. Now Foundations and Trends, 2019.

[30] BAO S, HE H, WANG F, et al. Know more about each other: Evolving dialogue strategy via compound assessment[J]. arXiv preprint arXiv:1906.00549, 2019.

[31] ZHOU L, GAO J, LI D, et al. The design and implementation of xiaoice, an empathetic social chatbot[J]. Computational Linguistics, 2020, 46(1): 53-93.

[32] KWON W, LI Z, ZHUANG S, et al. Efficient memory management for large language model serving with pagedattention[C]//Proceedings of the 29th Symposium on Operating Systems Principles. 2023: 611-626.

[33] FRANTAR E, ASHKBOOS S, HOEFLER T, et al. Gptq: Accurate post-training quantization for generative pre-trained transformers[J]. arXiv preprint arXiv:2210.17323, 2022.

[34] MAO J, RABINOVICH M, SCHOMP K. Assessing support for dns-over-tcp in the wild[C]//International Conference on Passive and Active Network Measurement. Springer, 2022: 487-517.

[35] CHITRALEKHA G. A survey on different scheduling algorithms in operating system[C]//Computer Networks and Inventive Communication Technologies: Proceedings of Third ICCNCT 2020. Springer, 2021: 637-651.

[36] GONG Y, CHEN M, SONG L, et al. Study on the classification model of lock mechanism in operating system[C]//2022 IEEE 2nd International Conference on Power, Electronics and Computer Applications (ICPECA). IEEE, 2022: 857-861.

[37] WOLF T, DEBUT L, SANH V, et al. Transformers: State-of-the-art natural language processing[C/OL]// Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations. Online: Association for Computational Linguistics, 2020: 38-45. https://www.aclweb.org/anthology/2020.emnlp-demos.6.

[38] GORMLEY I C, FRÜHWIRTH-SCHNATTER S. Mixture of experts models[M]//Handbook of mixture analysis. Chapman and Hall/CRC, 2019: 271-307.

[39] LI Y, GONG R, TAN X, et al. Brecq: Pushing the limit of post-training quantization by block reconstruction [J]. arXiv preprint arXiv:2102.05426, 2021.

[40] SCHRAMMEL D, SULTANA S, GREWAL K, et al. Memes: Memory encryption-based memory safety on commodity hardware[C/OL]//De Capitani di Vimercati S, SAMARATI P. SECRYPT 2023 - Proceedings of the 20th International Conference on Security and Cryptography: volume 1. Portugal: SciTePress, 2023:

25-36. DOI: 10.5220/0012050300003555.

[41] SCARLATA V, JOHNSON S, BEANEY J, et al. Supporting third party attestation for intel sgx with intel data center attestation primitives[J]. White paper, 2018, 12.

[42] HUANG Y, BAI Y, ZHU Z, et al. C-eval: A multi-level multi-discipline chinese evaluation suite for foundation models[C]//Advances in Neural Information Processing Systems. 2023.

[43] BAI Y, LV X, ZHANG J, et al. Longbench: A bilingual, multitask benchmark for long context understanding[J]. arXiv preprint arXiv:2308.14508, 2023.

[44] KASNECI E, SESSLER K, KÜCHEMANN S, et al. Chatgpt for good? on opportunities and challenges of large language models for education[J]. Learning and individual differences, 2023, 103: 102274.

[45] LIU S Y, LIU Z, HUANG X, et al. Llm-fp4: 4-bit floating-point quantized transformers[J]. arXiv preprint arXiv:2310.16836, 2023.

附录 A 正文符号定义

以下是文中使用的主要符号及其定义：

符号	定义
	在 §3.1.1 节中用到的符号
N	工作节点集合
n	集合 N 中的工作节点
S_n	工作节点 n 的处理速速
$\underset{x\in A}{argmin} f(x)$	x$\in A : f(x) \leq f(y)$对所有$y \in A$
R_n	通过计算得出的节点 n 的负载情况
	在 §3.1.3 节中用到的符号
T	生成的 token 的总数
t	当前时间对应的 token
M	模型生成 token 的过程
Δt	模型 M 生成 t 时间对应 token 的耗时
$S(x_t)$	发送 token x_t的独立线程
	在 §3.2 节中用到的符号
X_{f32}	float32 数据类型的矩阵 X
X_{i4}	int4 数据类型的矩阵 X
$\underset{ij}{max}(X)$	矩阵 X 中的最大值
$\underset{ij}{min}(X)$	矩阵 X 中的最小值
$scale_{x_{f32}}$	绝对值量化中的缩放因子
ndx_{f32}	零点量化中的缩放因子
zp_{xi32}	零点量化中的偏移量
$group\_size$	逐组量化时矩阵的大小
$W_Q^i, W_K^i, W_V^i, W_O^i$	第 i 层的 Q、K、V、O 权重矩阵
x_K^i, X_V^i, x_Q^i	传播至第 i 层时 K、V、Q 对应的矩阵运算结果
x^i	传播至第 i 层时的注意力分数
b	输入的批次大小

s	输入序列长度
h	模型维度
$memory_{max}$	模型向前传播过程中占用的峰值内存
在 §4.1 节中用到的符号	
W	词序列 W = $w_1, w_2, \ldots, w_N$
$P(W)$	模型预测输出词序列 W 的概率
$P(w_n)$	模型预测输出词 w_n的概率
$PP(W)$	模型困惑度，代表模型预测出序列 W 的不确定性

附录 B 量化算法核心代码

本系统的核心量化算法引用 llama.cpp 库，这些库通过 Python 的 ctypes 模块动态加载以实现接口的调用。

在 sym_int4 量化方法中，首先确定每个块中的最大绝对值和相应的原始最大值。接着，以最大值的负八分之一为基准，计算一个缩放因子（d），用来将块中的每个值归一化并缩放到一个定点数的范围内。最终，每个块中的值被量化为 int4 数值，并存放在 uint8 数据的低位和高位上。

```
1  void quantize_row_q4_0_reference(const float * restrict x,
     block_q4_0 * restrict y, int64_t k) {
2      static const int qk = QK4_0;
3      assert(k % qk == 0);
4      const int nb = k / qk;
5
6      for (int i = 0; i < nb; i++) {
7          float amax = 0.0f; // absolute max
8          float max  = 0.0f;
9
10         for (int j = 0; j < qk; j++) {
11             const float v = x[i*qk + j];
12             if (amax < fabsf(v)) {
13                 amax = fabsf(v);
14                 max  = v;
15             }
16         }
17
18         const float d  = max / -8;
19         const float id = d ? 1.0f/d : 0.0f;
20
```

```
        y[i].d = GGML_FP32_TO_FP16(d);

        for (int j = 0; j < qk/2; ++j) {
            const float x0 = x[i*qk + 0    + j]*id;
            const float x1 = x[i*qk + qk/2 + j]*id;

            const uint8_t xi0 = MIN(15, (int8_t)(x0 + 8.5f));
            const uint8_t xi1 = MIN(15, (int8_t)(x1 + 8.5f));

            y[i].qs[j]  = xi0;
            y[i].qs[j] |= xi1 << 4;
        }
    }
}
```

在 asym_int4 方法中，首先计算每个块中的最小值和最大值，然后用这两个值的差来确定一个缩放因子。这个缩放因子用于将块内的每个值线性转换到 int4 的范围内，同时保留了块内数值的相对大小和差异。转换公式涉及减去最小值和乘以逆缩放因子，使得量化后的数据可以更精确地表示原始数据的动态范围。

```
void quantize_row_q4_1_reference(const float * restrict x,
   block_q4_1 * restrict y, int64_t k) {
    const int qk = QK4_1;

    assert(k % qk == 0);

    const int nb = k / qk;

    for (int i = 0; i < nb; i++) {
        float min = FLT_MAX;
```

```
10          float max = -FLT_MAX;
11
12          for (int j = 0; j < qk; j++) {
13              const float v = x[i*qk + j];
14
15              if (v < min) min = v;
16              if (v > max) max = v;
17          }
18
19          const float d  = (max - min) / ((1 << 4) - 1);
20          const float id = d ? 1.0f/d : 0.0f;
21
22          y[i].d = GGML_FP32_TO_FP16(d);
23          y[i].m = GGML_FP32_TO_FP16(min);
24          for (int j = 0; j < qk/2; ++j) {
25              const float x0 = (x[i*qk + 0    + j] - min)*id;
26              const float x1 = (x[i*qk + qk/2 + j] - min)*id;
27
28              const uint8_t xi0 = MIN(15, (int8_t)(x0 + 0.5f));
29              const uint8_t xi1 = MIN(15, (int8_t)(x1 + 0.5f));
30
31              y[i].qs[j]  = xi0;
32              y[i].qs[j] |= xi1 << 4;
33          }
34      }
35  }
```

附录 C 实验部分涉及数据集

模型能力测试所使用到的 LongBench 数据集：

表 3.1 不同任务类型的评价指标和样本数量

任务	任务类型	评价指标	平均长度	语言	Sample 数量
HotpotQA	多文档 QA	F1	9,151	英文	200
2WikiMultihopQA	多文档 QA	F1	4,887	英文	200
MuSiQue	多文档 QA	F1	11,214	英文	200
DuReader	多文档 QA	Rouge-L	15,768	中文	200
MultiFieldQA-en	单文档 QA	F1	4,559	英文	150
MultiFieldQA-zh	单文档 QA	F1	6,701	中文	200
NarrativeQA	单文档 QA	F1	18,409	英文	200
Qasper	单文档 QA	F1	3,619	英文	200
GovReport	摘要	Rouge-L	8,734	英文	200
QMSum	摘要	Rouge-L	10,614	英文	200
MultiNews	摘要	Rouge-L	2,113	英文	200
VCSUM	摘要	Rouge-L	15,380	中文	200
TriviaQA	Few shot	F1	8,209	英文	200
SAMSum	Few shot	Rouge-L	6,258	英文	200
TREC	Few shot	Accuracy	5,177	英文	200
LSHT	Few shot	Accuracy	22,337	中文	200
PassageRetrieval-en	合成任务	Accuracy	9,289	英文	200
PassageCount	合成任务	Accuracy	11,141	英文	200
PassageRetrieval-zh	合成任务	Accuracy	6,745	中文	200
LCC	代码	Edit Sim	1,235	Python/C#/Java	500
RepoBench-P	代码	Edit Sim	4,206	Python/Java	500

系统性能评估时使用到的不同长度输入：

表 3.2 模型输入示例

输入长度	**内容**
32	Once upon a time, there existed a little girl who liked to have adventures...
256	Once upon a time, there was a young girl named Samantha who lived with her parents in a small town. Samantha...
1024	In the year 2048, the world was a very different place from what it had been just two decades before. The pace of technological progress had quickened to an...

致　　谢

人生并非二进制，无法用 0 和 1 来描述。

不知不觉间已经走到本科生涯的末尾，四年时光，有喜悦，有挫败，有孤独，也有欢聚。我记得自习室里为考试奋战的夜晚，记得大草坪傍晚时的点点星光，记得教学楼旁熙熙攘攘的人群，也记得宿舍楼下盛满月影的酒杯。一路走来，恍若昨日才始，又似已岁月悠长。

感谢我的指导教师凌振教授对本论文的严格审查与悉心指导，其专业的指引使我逐渐深入了解科研论文写作的完整流程。同时，我也要感谢英特尔亚太研发有限公司的史栋杰和龚奇源在系统开发过程中对我的答疑解惑。在面对难解的源码问题时，您们的及时指导总能启发我找到解决之道。感谢在英特尔实习期间给予我理解与支持的 Jason Dai、田翔宇、刘芍君、孙赫阳、张子腾、邓槟槟和 Qiu Xin 等同事，与您们的合作使我能够在大型项目组环境中熟练掌握并提升协同开发技能。

感谢我的女朋友霍雨佳对我的鼓励，感谢一直陪伴在我身边的孙英华、鲁洋洋、周俊、王一强、陈瑞星、王子航、张浩然等同学，在我的学术和生活旅程中提供了宝贵的支持和友谊。无论是学习上的互助还是生活中的相互鼓励，他们的存在让我的大学生活更加丰富多彩。感谢和我一起追逐风和自由的张凯迪、陈睿、李嘉靖同学，有幸能够一起体验骑在发动机上看世界的感觉。

感谢辅导员宋欣楠老师和董烨老师在我大学生涯中所提供的指导与支持。在我遇到困难和挑战时总能给予我及时的帮助和鼓励，让我能够以更积极的态度面对问题，并找到解决问题的方法。同时，感谢黄晓菁老师在教务管理和课程规划方面的专业指导，使我能够更有效地安排学习计划，确保学业进展顺利。

此刻亦是结束也是开始，再见东南大学，再见南京。我们，后会有期。

直接基于本文工作所形成的科研项目：

[1] IPEX-LLM. https://github.com/intel-analytics/ipex-llm. 在 Intel 硬件上加速绝大多数开源大语言模型的推理和微调. 本文作者系该项目设计与开发者.