Efficient LLM Inference on CPUs

Haihao Shen   Hanwen Chang   Bo Dong   Yu Luo   Hengyu Meng
{haihao.shen, hanwen.chang, bo1.dong, yu.luo, hengyu.meng}@intel.com

Abstract

Large language models (LLMs) have demonstrated remarkable performance and tremendous potential across a wide range of tasks. However, deploying these models has been challenging due to the astronomical amount of model parameters, which requires a demand for large memory capacity and high memory bandwidth. In this paper, we propose an effective approach that can make the deployment of LLMs more efficiently. We support an automatic INT4 weight-only quantization flow and design a special LLM runtime with highly-optimized kernels to accelerate the LLM inference on CPUs. We demonstrate the general applicability of our approach on popular LLMs including Llama2, Llama, GPT-NeoX, and showcase the extreme inference efficiency on CPUs. The code is publicly available at: https://github.com/intel/intel-extension-for-transformers.

1   Introduction

Large language models (LLMs) have shown remarkable performance and tremendous potential across a wide range of tasks Rozière et al. [2023], Touvron et al. [2023b,a], Zhang et al. [2022], Brown et al. [2020], Li et al. [2023]. However, deploying these models has been challenging due to the astronomical amount of model parameters, which necessitates significant memory capacity and high memory bandwidth.

Quantization is a technique to reduce the numeric precision of weights and activations of a neural network to lower the computation costs of inference. INT8 quantization Vanhoucke et al. [2011], Han et al. [2015], Jacob et al. [2018] is the most widely-used approach today given the trade-off between high inference performance and reasonable model accuracy. However, outliers in activations have been observed and those outlier values are limiting the general adoption of INT8 quantization, though there are some related work that has been proposed to address the issues Xiao et al. [2023], Wei et al. [2023], Dettmers et al. [2022]. FP8 is a newly introduced data type that has attracted lots of attentions Micikevicius et al. [2022], Kuzmin et al. [2022], Sun et al. [2019], Shen et al. [2023] while it has little adoptions due to the hardware unavailability. On the other hand, weight-only quantization becomes popular as it applies the low precision (e.g., 4-bit) to weights only, while keeping higher precision (e.g., 16-bit floating point) for activations, therefore maintaining the model accuracy. There are many excellent work on 4-bit weight-only quantization Dettmers et al. [2023], Frantar et al. [2022], Cheng et al. [2023b], Lin et al. [2023], Kim et al. [2023], Wu et al. [2023], Cheng et al. [2023a] that have demonstrated the effectiveness in LLM inference. Meanwhile, the open-source community is embracing such low-bit weight-only quantization and offers the CPP-based implementations such as llama.cpp and starcoder.cpp based on ggml library. These implementations are typically optimized for CUDA and may not work on CPUs. Therefore, it is important to address the challenge of making LLM inference efficient on CPU.

In this paper, we propose an effective approach for LLM inference on CPUs including an automatic INT4 quantization flow and an efficient LLM runtime. We leverage Intel Neural Compressor that provides the support of INT4 quantization such as GPTQ Frantar et al. [2022], AWQ Lin et al. [2023], TEQ Cheng et al. [2023a], SignRound Cheng et al. [2023b] and generate the INT4 model automatically. Inspired from the ggml library, we develop a tensor library for CPU, supporting all

the mainstream instruction sets such as AVX2, AVX512, AVX512_VNNI Rodriguez et al. [2018], and AMX (Advanced Matrix Extensions). Our results show the average latency of generation tokens from 20ms to 80ms on LLMs with 6B to 20B parameters using just a single socket of 4th Generation Intel® Xeon® Scalable Processors, while preserving the high accuracy within only 1% loss from FP32 baseline. Our main contributions are as follows:

- We propose an automatic INT4 quantization flow and generate the high-quality INT4 models with negligible accuracy loss within <1% from FP32 baseline.
- We design a tensor library that supports general CPU instruction sets and latest instruction sets for deep learning acceleration. With CPU tensor library, we develop an efficient LLM runtime to accelerate the inference.
- We apply our inference solution to popular LLM models covering 3B to 20B and demonstrate the promising per-token generation latency from 20ms to 80ms, much faster than the average human reading speed about 200ms per token.

The rest of this paper is organized as follows. Section 2 introduces the approach including INT4 quantization and inference. Section 3 outlines the experimental setup, presents accuracy & performance results, and offers discussion on performance tuning. Section 4 presents the conclusions and future work.

2 Approach

In this section, we introduce the approach which consists of two major components: an automatic INT4 quantization flow and an efficient LLM runtime, as shown in Figure 1. More details are described in the following sections.
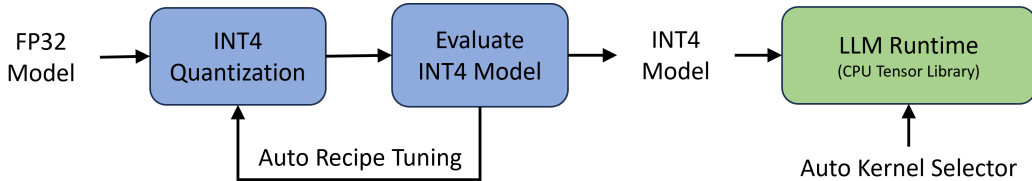

Figure 1: The left part is the automatic INT4 quantization flow: given a FP32 model, the flow takes the default INT4 quantization recipes and evaluates the accuracy of INT4 model; the recipe tuning loop is optional, if INT4 model can meet the accuracy target. The right part is a simplified runtime for efficient LLM inference built on top of a CPU tensor library with automatic kernel selector.

2.1 Automatic INT4 Quantization Flow

INT4 quantization flow is developed based on Intel Neural Compressor, a popular quantization tool for deep learning frameworks. Since the tool has already supported the mainstream INT4 quantization recipes such as GPTQ, SignRound, AWQ, TEQ, and RTN (round-to-nearest), our automatic quantization flow allows the recipe tuning on different quantization recipes, different granularities (channel-wise or group-wise), different group size (32, 64, 128 ... 1024). Each recipe generates an INT4 model that is evaluated in the flow. Once the INT4 model meets the accuracy target, the model will be passed to LLM Runtime for performance evaluation.

2.2 Efficient LLM Runtime

LLM runtime is designed to provide the efficient inference of LLMs on CPUs. Figure 2 describes the key components in LLM runtime, where the components (CPU tensor library and LLM optimizations) in green are specialized for LLM inference, while the other components (memory management, thread scheduler, operator optimization and fusion) in blue are required for a general runtime. More details about CPU tensor library and LLM optimizations are described in the following paragraphs, while the general components are omitted due to the space limitations. Note that the design is flexibly extensible with hardware abstraction layer (CPU only for now), while how to support other hardware is out of scope in this paper.
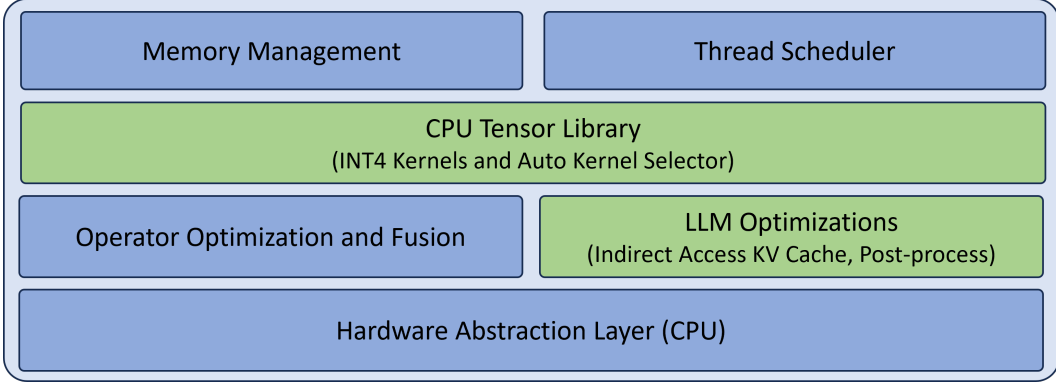
Figure 2: Key components in LLM runtime: general and LLM specialized.

CPU Tensor Library.   We develop CPU tensor library for linear algebra subroutines, inspired from the template design of cutlass. The tensor library offers a comprehensive support of INT4 kernels for x86 CPUs as shown in Table 1, where AMX is available in the latest Intel Xeon Scalable Processors and VNNI is available in both Intel and AMD CPUs.

Table 1: Support matrix by CPU tensor library: input/output data type, compute data type, and ISA (instruction set architecture). The library supports dynamic quantization for input along with batch or input channel per group, and weight quantization in both symmetric and asymmetric scheme.

Input Data Type	Output Data Type	Compute Data Type	Compute ISA
FP32	FP32	FP32	AVX2
FP32	FP32	FP32	AVX512F
FP32	FP32	INT8	AVX_VNNI
FP32	FP32	INT8	AVX512_VNNI
FP32	FP32	INT8	AMX_INT8
FP32/FP16	FP32/FP16	FP16	AVX512_FP16
FP32/BF16	FP32/BF16	BF16	AMX_BF16

LLM Optimizations.   Most recent LLMs are typically decoder-only Transformer-based models Vaswani et al. [2017]. Given the unique characteristics of next token generation, KV cache becomes performance critical for LLM inference. We describe the optimizations in Figure 3.
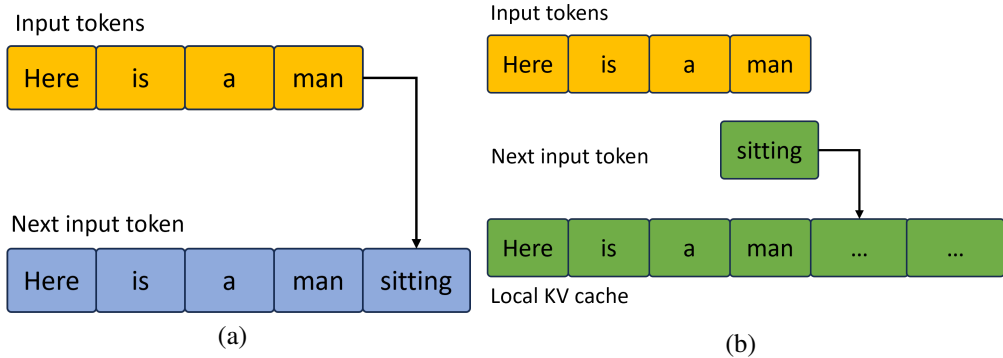

Figure 3: KV cache optimization. Left (a) shows the default KV cache, where new token generation requires memory reallocation for all the tokens (5 in this example); right (b) shows the optimized KV cache with pre-allocated KV memory and only new token updated each time.

3

3 Results

3.1 Experimental Setup

To demonstrate the generality, we select the popular LLMs across a wide range of architectures with the model parameter size from 7B to 20B. We evaluate the accuracy of both FP32 and INT4 models using open-source datasets from lm-evaluation-harness including lambada Paperno et al. [2016] openai, hellaswag Zellers et al. [2019], winogrande Sakaguchi et al. [2021], piqa Bisk et al. [2020], and wikitext. To demonstrate the performance, we measure the latency of next token generation on the 4th Generation Intel® Xeon® Scalable Processors, available on the public clouds such as AWS.

3.2 Accuracy

We evaluate the accuracy on the aforementioned datasets and show the average accuracy in Table 2. We can see from the table that the accuracy of INT4 model is nearly on par with that of FP32 model within 1% relative loss from FP32 baseline.

Table 2: INT4 and FP32 model accuracy. INT4 model has two configurations: group size=32 and 128.

LLM	FP32	INT4 (Group size=32)	INT4 (Group size=128)
EleutherAI/gpt-j-6B	0.643	0.644	0.64
meta-llama/Llama-2-7b-hf	0.69	0.69	0.685
decapoda-research/llama-7b-hf	0.689	0.682	0.68
EleutherAI/gpt-neox-20b	0.674	0.672	0.669
tiiuae/falcon-7b	0.698	0.694	0.693

3.3 Performance

We measure the latency of next token generation using LLM runtime and the popular open-source ggml-based implementation. Table 3 presents the latency under a proxy configuration with 32 as both input and output tokens. Note that ggml-based solution only supports group size 32 when testing.

Table 3: INT4 performance using LLM runtime and ggml-based solution. LLM runtime outperforms ggml-based solution by up to 1.6x under group-size=128 and 1.3x under group size=32.

model	LLM Runtime (Group size=32)	LLM Runtime (Group size=128)	ggml-based (Group size=32)
EleutherAI/gpt-j-6B	22.99ms	19.98ms	31.62ms
meta-llama/Llama-2-7b-hf	23.4ms	21.96ms	27.71ms
decapoda-research/llama-7b-hf	23.88ms	22.04ms	27.2ms
EleutherAI/gpt-neox-20b	80.16ms	61.21ms	92.36ms
tiiuae/falcon-7b	31.23ms	22.26ms	36.22ms

3.4 Discussion

Though we demonstrate the performance advantage over ggml-based solution, there are still opportunities for LLM runtime to further improve the performance through additional performance tuning such as thread scheduler in LLM runtime, blocking strategy in CPU tensor library.

4 Summary and Future Work

We presented an end-to-end INT4 LLM inference including an automatic INT4 model quantization and efficient LLM runtime. We demonstrated the generality on a set of popular LLMs and the performance advantage over the open-source solution on CPUs. As our future works, we plan to further improve the CPU tensor library and extend Hugging Face transformer APIs to support INT4 LLM inference as part of the contributions to the open-source community. Moreover, we plan to exercise our approach on personal computers (PCs) given the broad accessibility of CPUs, to meet the growing demands of AI generated content and empower generative AI on PCs.

4.1 Appendix

Memory usage is a critical metric for the 4-bit solution. The results are presented below.

Table 4: INT4 and FP32 memory usage. INT4 model has two configurations: group size=128 and 32.

LLM	FP32	INT4 (Group size=128)	INT4 (Group size=32)
EleutherAI/gpt-j-6B	22481MB	3399MB	4017MB
meta-llama/Llama-2-7b-hf	22057MB	3772MB	4928MB
decapoda-research/llama-7b-hf	21750MB	3773MB	4874MB
EleutherAI/gpt-neox-20b	68829MB	11221MB	13458MB
tiiuae/falcon-7b	24624MB	5335MB	5610MB

References

Y. Bisk, R. Zellers, J. Gao, Y. Choi, et al. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 7432–7439, 2020.

T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

W. Cheng, Y. Cai, K. Lv, and H. Shen. Teq: Trainable equivalent transformation for quantization of llms. *arXiv preprint arXiv:2310.10944*, 2023a.

W. Cheng, W. Zhang, H. Shen, Y. Cai, X. He, and K. Lv. Optimize weight rounding via signed gradient descent for the quantization of llms. *arXiv preprint arXiv:2309.05516*, 2023b.

T. Dettmers, M. Lewis, Y. Belkada, and L. Zettlemoyer. Llm. int8 (): 8-bit matrix multiplication for transformers at scale. *arXiv preprint arXiv:2208.07339*, 2022.

T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer. Qlora: Efficient finetuning of quantized llms. *arXiv preprint arXiv:2305.14314*, 2023.

E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*, 2022.

S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding, 2015. URL `https://arxiv.org/abs/1510.00149`.

B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

S. Kim, C. Hooper, A. Gholami, Z. Dong, X. Li, S. Shen, M. W. Mahoney, and K. Keutzer. Squeezellm: Dense-and-sparse quantization. *arXiv preprint arXiv:2306.07629*, 2023.

A. Kuzmin, M. Van Baalen, Y. Ren, M. Nagel, J. Peters, and T. Blankevoort. Fp8 quantization: The power of the exponent. *Advances in Neural Information Processing Systems*, 35:14651–14662, 2022.

R. Li, L. B. Allal, Y. Zi, N. Muennighoff, D. Kocetkov, C. Mou, M. Marone, C. Akiki, J. Li, J. Chim, et al. Starcoder: may the source be with you! *arXiv preprint arXiv:2305.06161*, 2023.

J. Lin, J. Tang, H. Tang, S. Yang, X. Dang, and S. Han. Awq: Activation-aware weight quantization for llm compression and acceleration. *arXiv preprint arXiv:2306.00978*, 2023.

P. Micikevicius, D. Stosic, N. Burgess, M. Cornea, P. Dubey, R. Grisenthwaite, S. Ha, A. Heinecke, P. Judd, J. Kamalu, et al. Fp8 formats for deep learning. *arXiv preprint arXiv:2209.05433*, 2022.

D. Paperno, G. Kruszewski, A. Lazaridou, Q. N. Pham, R. Bernardi, S. Pezzelle, M. Baroni, G. Boleda, and R. Fernández. The lambada dataset: Word prediction requiring a broad discourse context. *arXiv preprint arXiv:1606.06031*, 2016.

A. Rodriguez, E. Segal, E. Meiri, E. Fomenko, Y. J. Kim, H. Shen, and B. Ziv. Lower numerical precision deep learning inference and training. *Intel White Paper*, 3(1):19, 2018.

B. Rozière, J. Gehring, F. Gloeckle, S. Sootla, I. Gat, X. E. Tan, Y. Adi, J. Liu, T. Remez, J. Rapin, et al. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023.

K. Sakaguchi, R. L. Bras, C. Bhagavatula, and Y. Choi. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

H. Shen, N. Mellempudi, X. He, Q. Gao, C. Wang, and M. Wang. Efficient post-training quantization with fp8 formats, 2023.

X. Sun, J. Choi, C.-Y. Chen, N. Wang, S. Venkataramani, V. V. Srinivasan, X. Cui, W. Zhang, and K. Gopalakrishnan. Hybrid 8-bit floating point (hfp8) training and inference for deep neural networks. *Advances in neural information processing systems*, 32, 2019.

H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023a.

H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023b.

V. Vanhoucke, A. Senior, and M. Z. Mao. Improving the speed of neural networks on cpus. In *Deep Learning and Unsupervised Feature Learning Workshop, NIPS 2011*, 2011.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

X. Wei, Y. Zhang, Y. Li, X. Zhang, R. Gong, J. Guo, and X. Liu. Outlier suppression+: Accurate quantization of large language models by equivalent and optimal shifting and scaling. *arXiv preprint arXiv:2304.09145*, 2023.

X. Wu, Z. Yao, and Y. He. Zeroquant-fp: A leap forward in llms post-training w4a8 quantization using floating-point formats. *arXiv preprint arXiv:2307.09782*, 2023.

G. Xiao, J. Lin, M. Seznec, H. Wu, J. Demouth, and S. Han. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning*, pages 38087–38099. PMLR, 2023.

R. Zellers, A. Holtzman, Y. Bisk, A. Farhadi, and Y. Choi. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*, 2019.

S. Zhang, S. Roller, N. Goyal, M. Artetxe, M. Chen, S. Chen, C. Dewan, M. Diab, X. Li, X. V. Lin, et al. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022.