

使用xFasterTransformer(xFT)加速英特尔® 至强®平台上的大语言模型 (LLM)推理

缪金成

英特尔数据中心与人工智能事业部 高级软件工程师

01

第四代英特尔® 至强® 可扩展处理器简介

- 至强®算力篇：从向量到矩阵
- 至强®内存篇：HBM

02

至强®平台LLM推理加速引擎 xFT

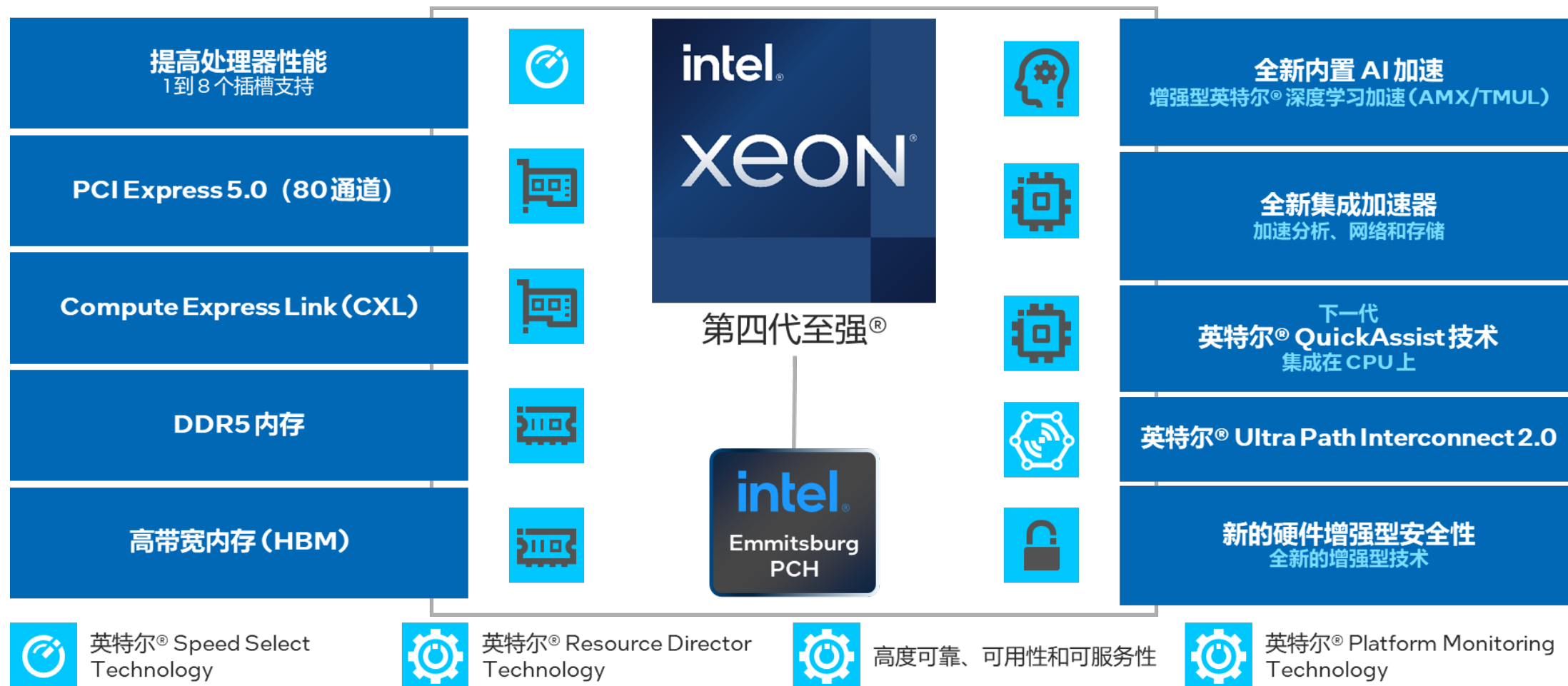
- xFasterTransformer开源项目
- xFT软件架构及特点
- 性能数据

03

Intel® Extension for PyTorch*

- Intel® Extension for PyTorch*软件架构及特点
- IPEX-LLM 优化与性能数据
- Pytorch社区贡献

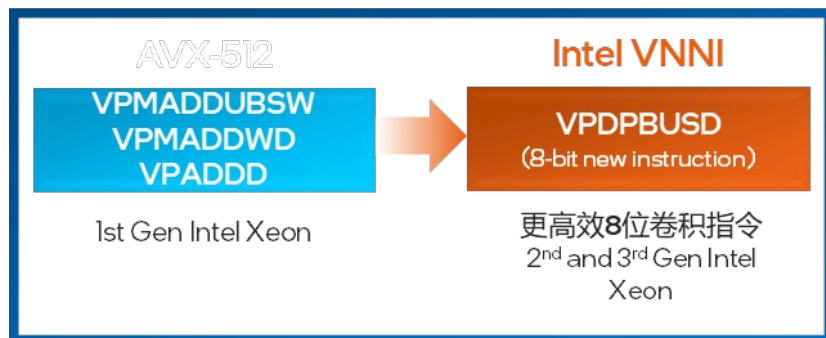
英特尔® 第四代至强® 可扩展处理器



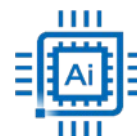
至强® 算力篇：从向量到矩阵



AVX-512
倍增的寄存器宽度512bit



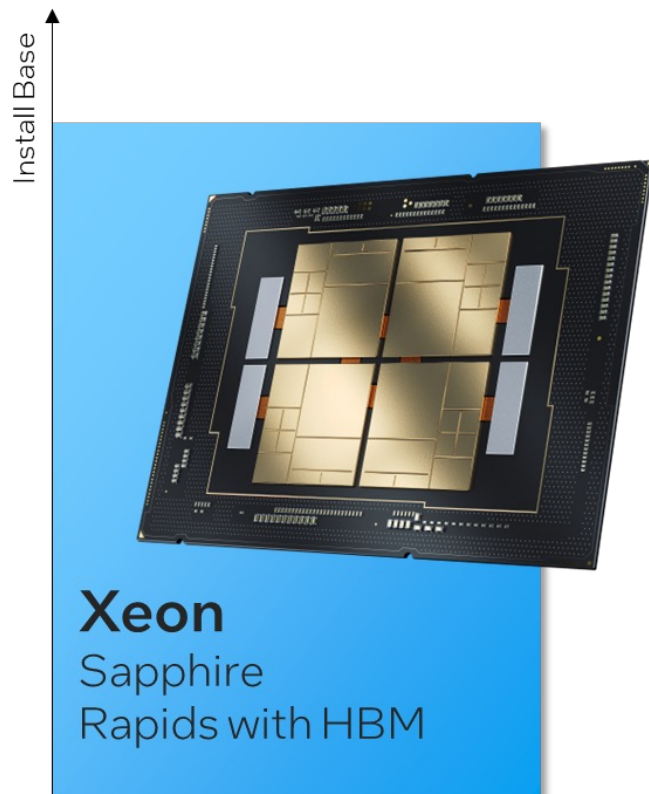
VNNI-INT8/BF16
支持VNNI卷积指令集和16bitBfloat浮点



英特尔® AMX

AMX
CPU 片上 AI 加速器




至强® 内存篇：HBM



4x

Memory
Bandwidth

Up to
64 GB
HBM2e

 Performance Cores	Advanced Matrix Extensions	Integrated Acceleration Engines	Compute
HBM2e	>100MB Shared LLC	Optane™	Memory
 CXL Compute Express Link 1.1	PCIe Gen 5	UPI 2.0 Improved Multi- socket Scaling	I/O
 EMIB	Multi-Tile	Logically Monolithic	Technology

¹Based on pre-production measurements and memory bound workloads

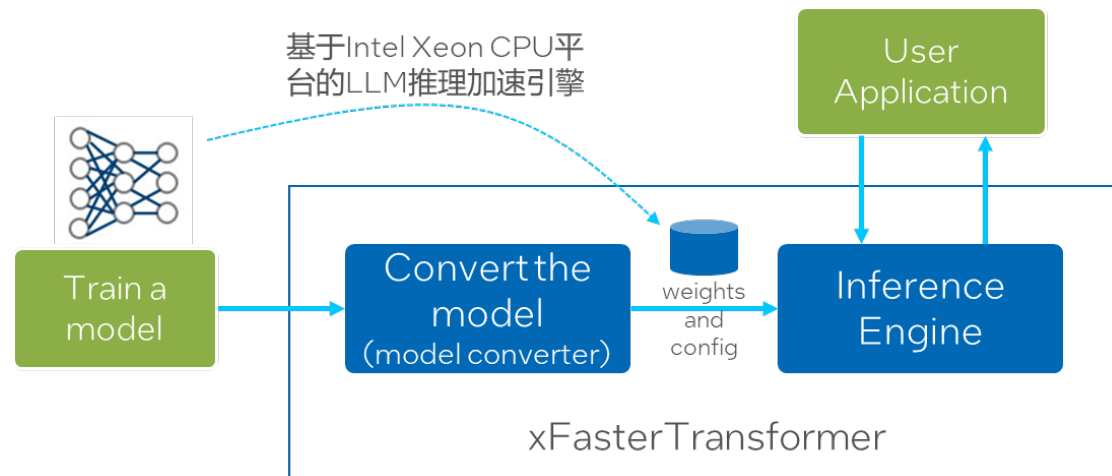
Performance

xFasterTransformer开源项目

<https://github.com/intel/xFasterTransformer>

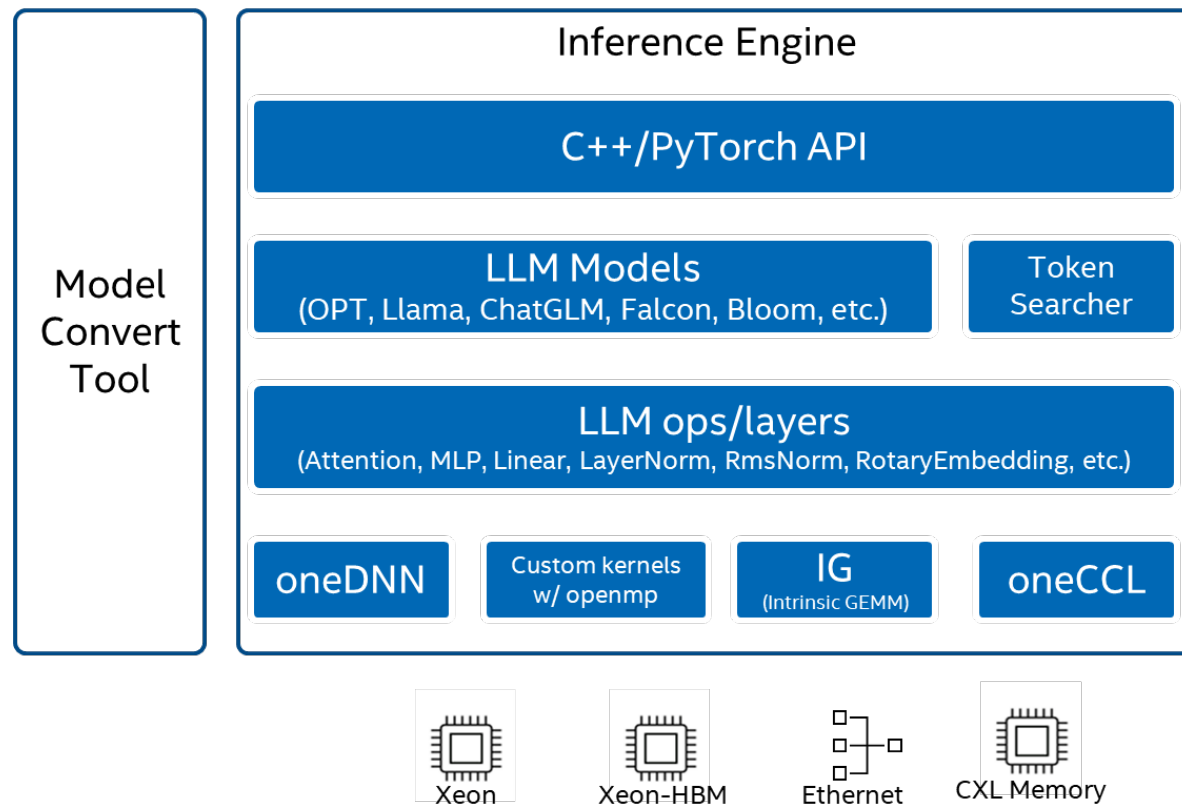


- Apache license
- 英特尔® 至强平台特别优化
- 英特尔® 硬件特性加速
- 支持FP16/BF16/INT8权重格式和计算
- 支持兼容Huggingface等主流模型格式：OPT, Llama, Llama2, ChatGLM, ChatGLM2, Falcon等
- 具有良好的分布式可扩展性



xFT软件架构

- IG: LLM专用Intrinsic GEMM 库
- oneCCL: 英特尔®高性能异构通信库
- oneMKL: 英特尔®高性能计算库，可用于批量矩阵运算
- LLM ops/layers: 支持分布式多机计算的LLM算子
- LLM Models: 支持学术界及工业界流行模型，并且有算子融合优化。



xFT之特点: 高性能LLM推理平台

高性能

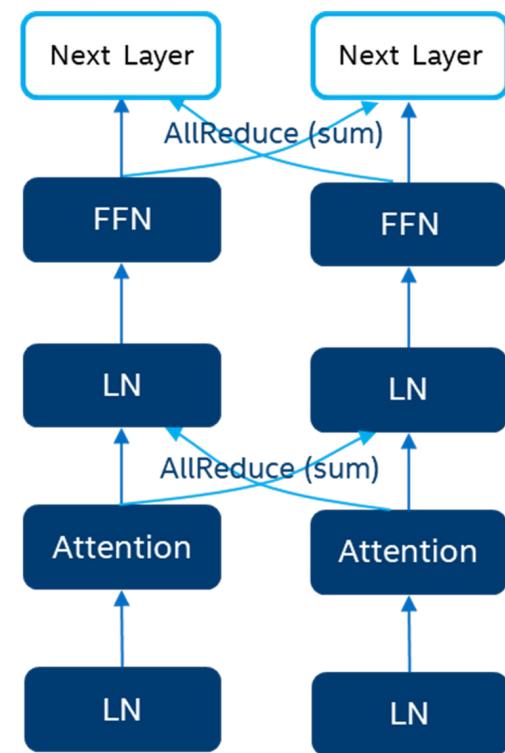
- Flash Attention 优化实现
- 数据类型转换/数据排布算子融合
- 充分利用AMX/AVX512 硬件特性加速计算
- 指令级GEMV/GEMM kernel优化
- 统一内存分配管理

并行可扩展性

- 支持Tensor Parallelism并行
- 使用SharedMem/oneCCL异构集合通信库

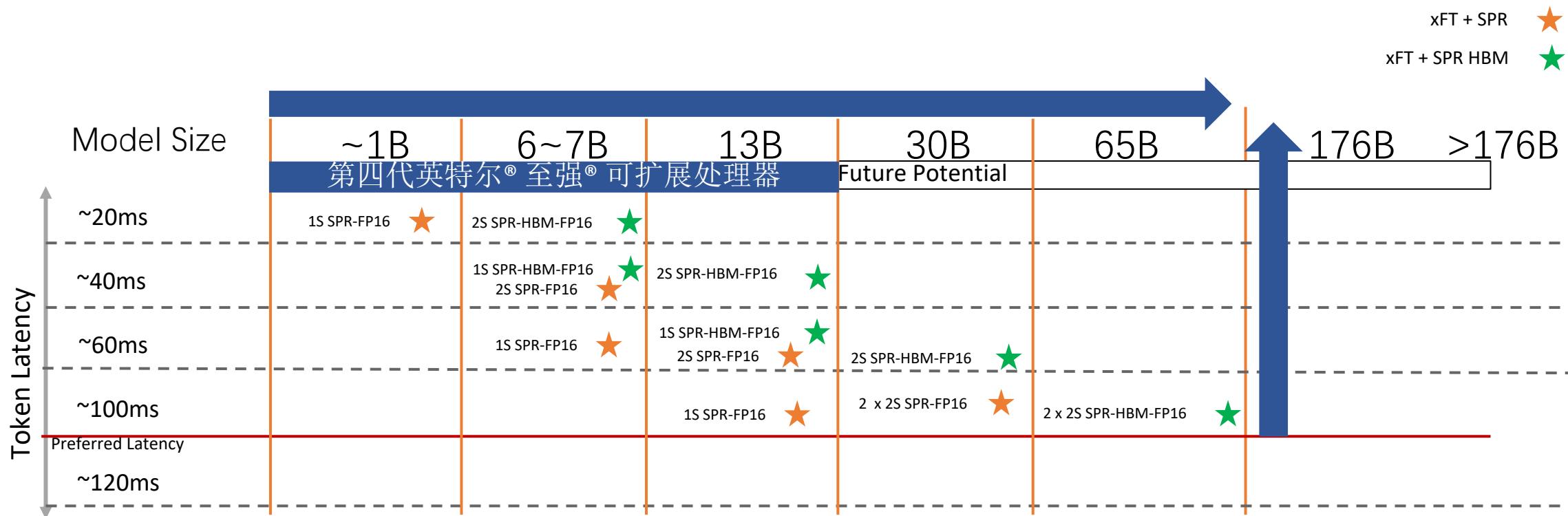
良好的兼容性

- C++ 高性能推理库，作为后端引擎易于集成进框架



One Layer Structure in
Transformer Encoder/Decoder

至强处理器加速大模型推理性能



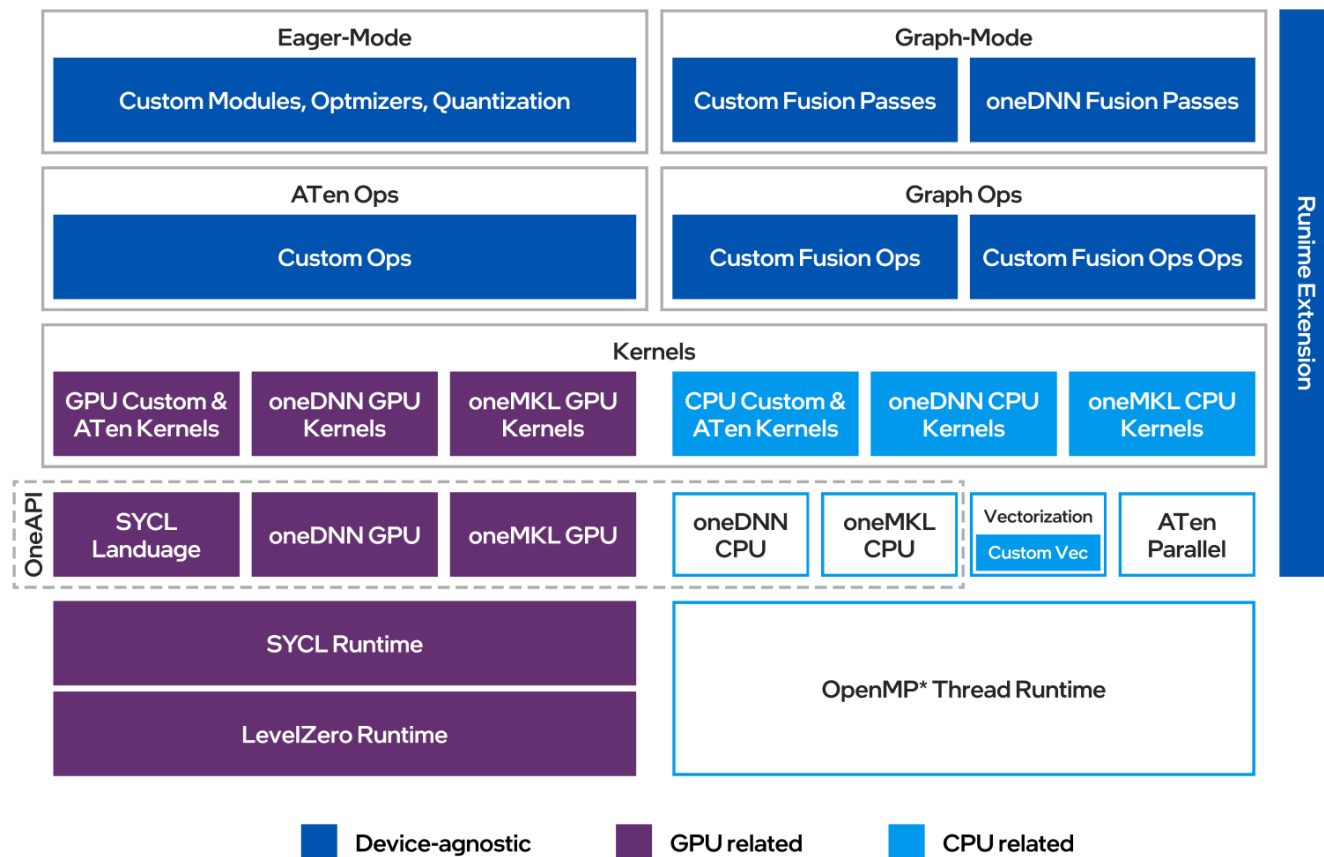
第四代英特尔® 至强® 可扩展处理器很好的支持1B ~ 13B的大模型推理

Intel® Extension for PyTorch*

张练钢

英特尔数据中心与人工智能事业部 人工智能框架工程师

Intel® Extension for PyTorch*



- Staging area for out-of-tree Intel optimizations
- Support both Intel CPU and GPU
- Extra performance boost gained with ease-of-use Python API
- Support both Python and C++ based deployment
- Same major release cadence as PyTorch

Major Optimization Methodologies

- General performance optimization and Intel new feature enabling in PyTorch upstream
- Additional performance boost and early adoption of aggressive optimizations through Intel® Extension for PyTorch*
- Multiple dimensions parallel for low latency and high throughput

Operator Optimization

- Vectorization
- Parallelization
- Memory Layout
- Low Precision

Graph Optimization

- Operator fusion
- Constant folding

Low Precisions

- Mixed-Precision
- Post Training Quantization
 - Static quantization
 - Dynamic quantization

Graph Optimization

Graph Generation

- ❑ TorchScript
- ❑ Torch.compile(PyTorch-V2.0+)
 - 'ipex' as an optimization backend.

Graph Fusion

- ❑ Automatically Adoption
- ❑ Rich Fusion Patterns for OOB Performance
 - Compute Bound + Memory Bound
 - ❖ Linear/Conv/Matmul/Einsum + Post-ops(e.g., activation, binary operation)
 - Customized Fusion Pattern
 - ❖ FlashAttention, Add+LayerNorm,.....

Inference using BF16 on IPEX

Resnet50

```
import torch
import torchvision.models as models

model = models.resnet50(weights='ResNet50_Weights.DEFAULT')
model.eval()
data = torch.rand(1, 3, 224, 224)

##### code changes #####
import intel_extension_for_pytorch as ipex
model = ipex.optimize(model, dtype=torch.bfloat16)
#####

with torch.no_grad(), torch.cpu.amp.autocast():
    model = torch.jit.trace(model, torch.rand(1, 3, 224, 224))
    model = torch.jit.freeze(model)

    model(data)
```

BERT

```
import torch
from transformers import BertModel

model = BertModel.from_pretrained("bert-base-uncased")
model.eval()

vocab_size = model.config.vocab_size
batch_size = 1
seq_length = 512
data = torch.randint(vocab_size, size=[batch_size, seq_length])

##### code changes #####
import intel_extension_for_pytorch as ipex
model = ipex.optimize(model, dtype=torch.bfloat16)
#####

with torch.no_grad(), torch.cpu.amp.autocast():
    d = torch.randint(vocab_size, size=[batch_size, seq_length])
    model = torch.jit.trace(model, (d,), check_trace=False, strict=False)
    model = torch.jit.freeze(model)

    model(data)
```

Key LLM Optimizations in IPEX

General GEMM/GEMV Kernel Optimizations

- Systolic array support from HW: AVX2/AVX512/AMX on SPR, DPAS on PVC
- Improve data locality: Register/cache blocking, weight prepacking

Multiple Low-precision Solutions

- BF16 auto-mixed precision
- INT8 static and smooth quantization
- INT4/INT8 weight-only quantization (WOQ)
 - GPTQ/Block-wise Quantization
- BF16/FP32/FP16/INT8 compute

High Performance MHA

- Concat Q/K/V Linear and Post ops fusion
- Indirect access KV cache (a simplified PagedAttention algorithm)
 - Zero-Copy for Next tokens
 - Prompt sharing for better locality for Scale-dot product.
- Flash attention, ROPE/RmsNorm fusion etc.

Distributed Inference

- Tensor Parallel
 - Auto Tensor Parallel for most of LLM models
 - SHM/oneCCL based all-reduce with low latency)

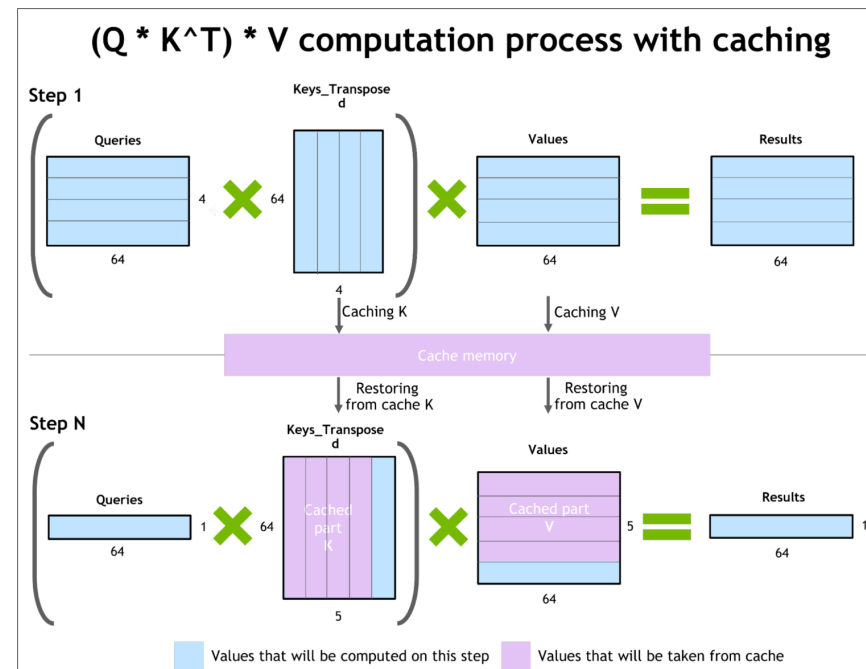
Indirect-access KV cache optimization

Motivation

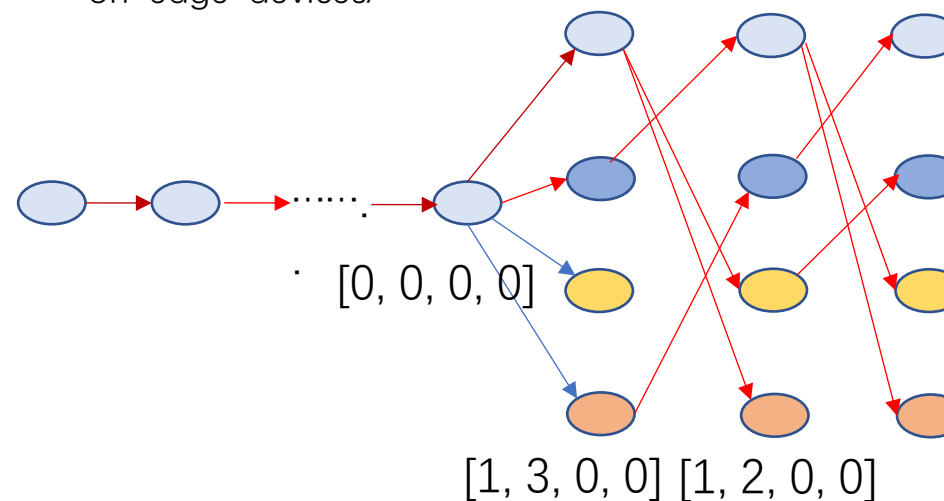
- cat and reorder_cache caused by kv_cache are the hotspot for long sequence (Even >50%)

Token	beam*batch_size	hidden state
t0	0	
	1	
	2	
	3	
t1	0	
	1	
	2	
	3	
...	...	
	...	
	...	
	...	
tn	0	
	1	
	2	
	3	

Cache format in pre-allocated buffer, indexing tokens with beam indices



KV cache illustrated: <https://www.axelera.ai/decoding-transformers-on-edge-devices/>



IPEX-LLM: LLM Optimization with IPEX

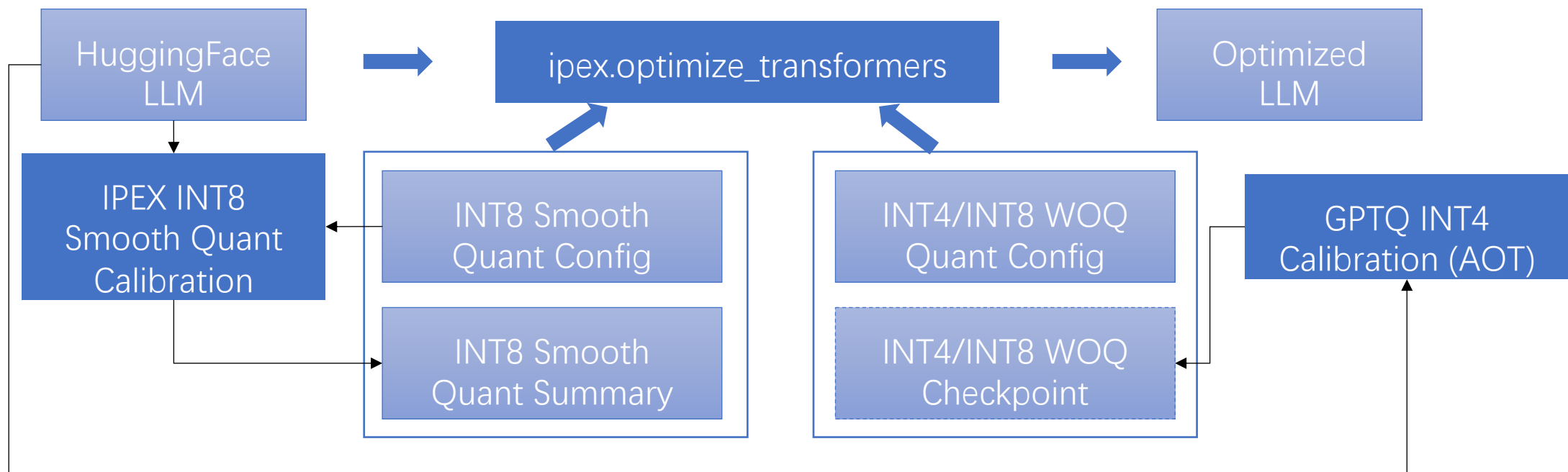
- One-liner API “`ipex.optimize_transformers`” to apply all optimizations
- [Get Started](#)

```
import torch
import intel_extension_for_pytorch as ipex
import transformers

model = transformers.AutoModelForCausalLM(model_name_or_path).eval()

dtype = torch.float # or torch.bfloat16
model = ipex.optimize_transformers(model, dtype=dtype)

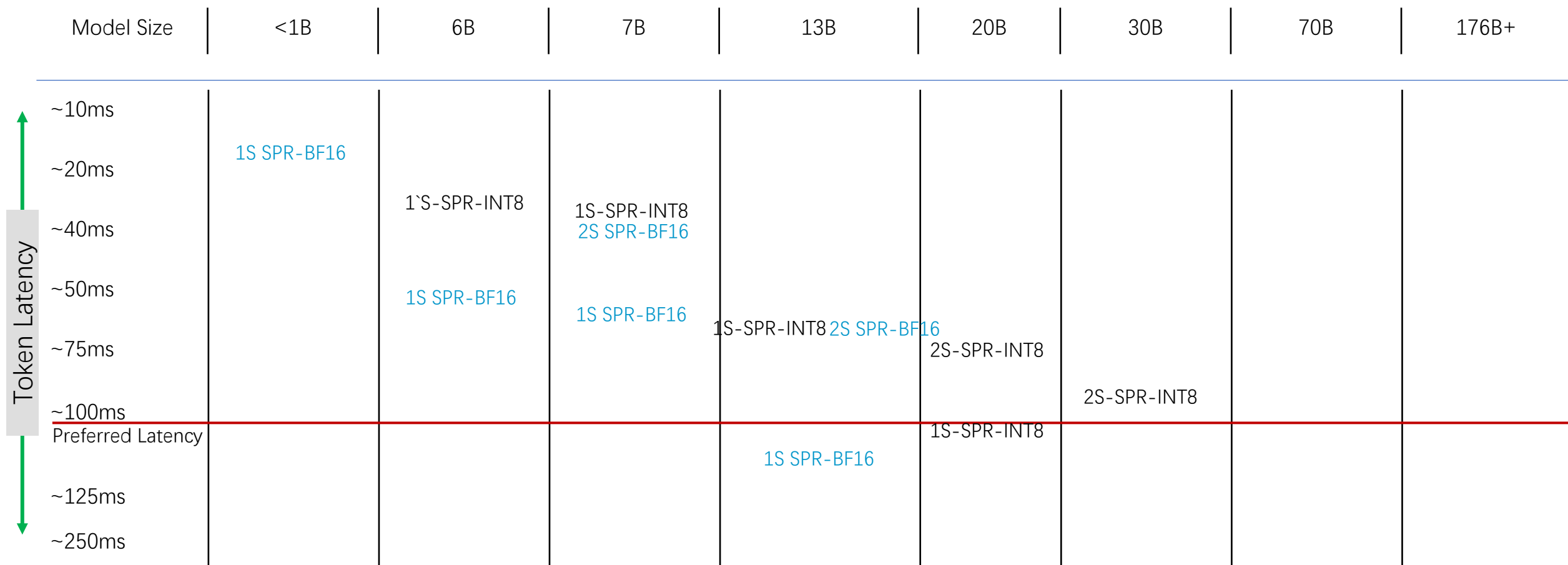
# inference with model.generate()
...
```



Optimized Model Architectures

MODEL FAMILY	Verified < MODEL ID > (Huggingface hub)	FP32/BF16	Weight only quantization INT8	Weight only quantization INT4	Static quantization INT8
LLAMA	"meta-llama/Llama-2-7b-hf", "meta-llama/Llama-2-13b-hf", "meta-llama/Llama-2-70b-hf"	✓	✓	✓	✓
GPT-J	"EleutherAI/gpt-j-6b"	✓	✓	✓	✓
GPT-NEOX	"EleutherAI/gpt-neox-20b"	✓	✓	✓	✗ **
FALCON*	"tiiuae/falcon-40b"	✓	✓	✓	✗ **
OPT	"facebook/opt-30b", "facebook/opt-1.3b"	✓	✓	✓	✗ **
Bloom	"bigscience/bloom", "bigscience/bloom-1b7"	✓	✓	✓	✗ **
CodeGen	"Salesforce/codegen-2B-multi"	✓	✓	✓	✗ **
Baichuan	"baichuan-inc/Baichuan2-13B-Chat", "baichuan-inc/Baichuan2-7B-Chat", "Baichuan-inc/Baichuan-13B-Chat"	✓	✓	✓	✗ **
ChatGLM	"THUDM/chatglm3-6b", "THUDM/chatglm2-6b"	✓	✓	✓	✗ **
GPTBigCode	"bigcode/starcoder"	✓	✓	✓	✗ **
T5	"google/flan-t5-xl"	✓	✓	✓	✗ **
Mistral	"mistralai/Mistral-7B-v0.1"	✓	✓	✓	✗ **

LLM AIGC Performance with IPEX



SPR: The 4th Gen Xeon 8480

LLM AIGC Performance with IPEX

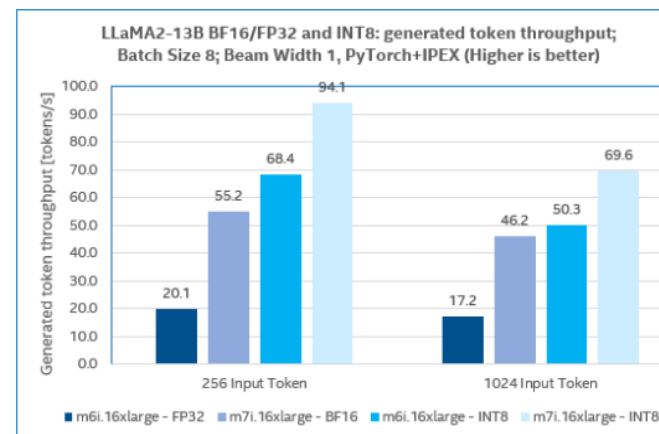
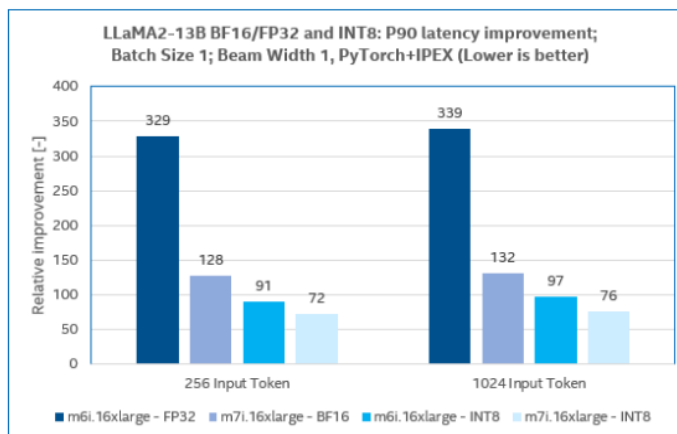
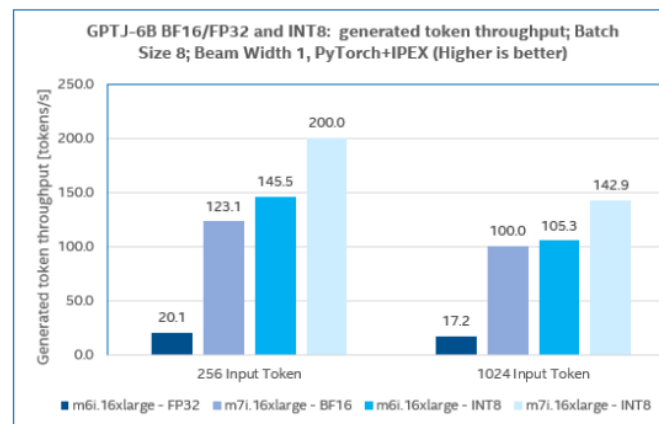
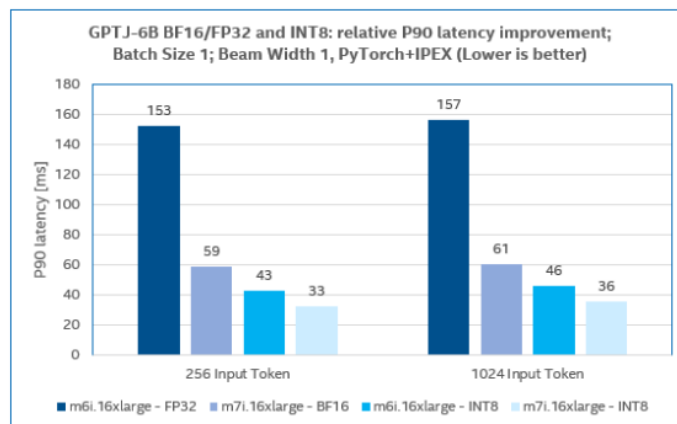
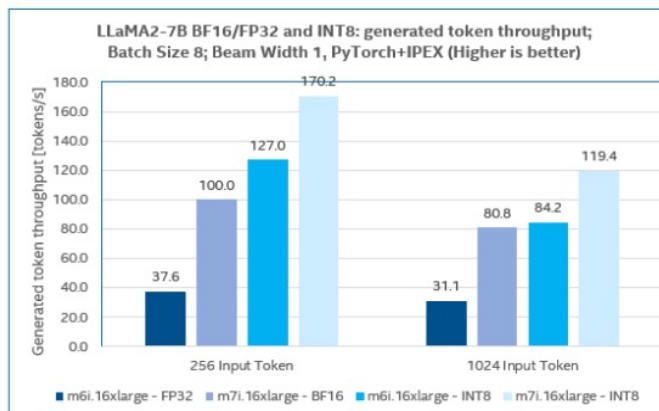
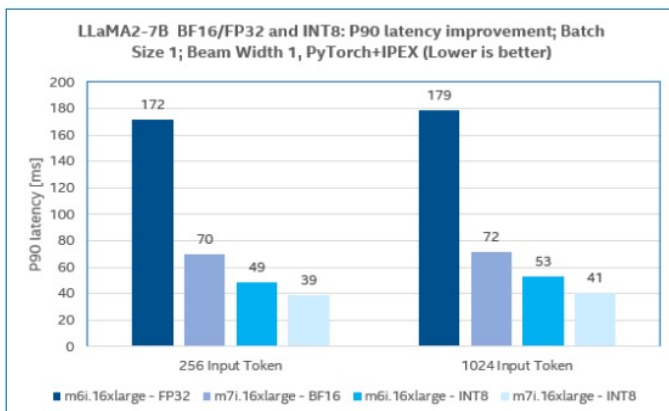
- The LLM inference performances on M7i and M6i instances are compared based on the above results. M7i, with the 4th Gen Xeon® processors, has a remarkable performance advantage over M6i with the 3rd Gen Xeon® processors.
- With a larger batch size the capacity of the model service can be improved at the cost of longer response latency for the individual sessions.

M7i performance boost ratio over M6i for non-quantized (BF16 or FP32) models:

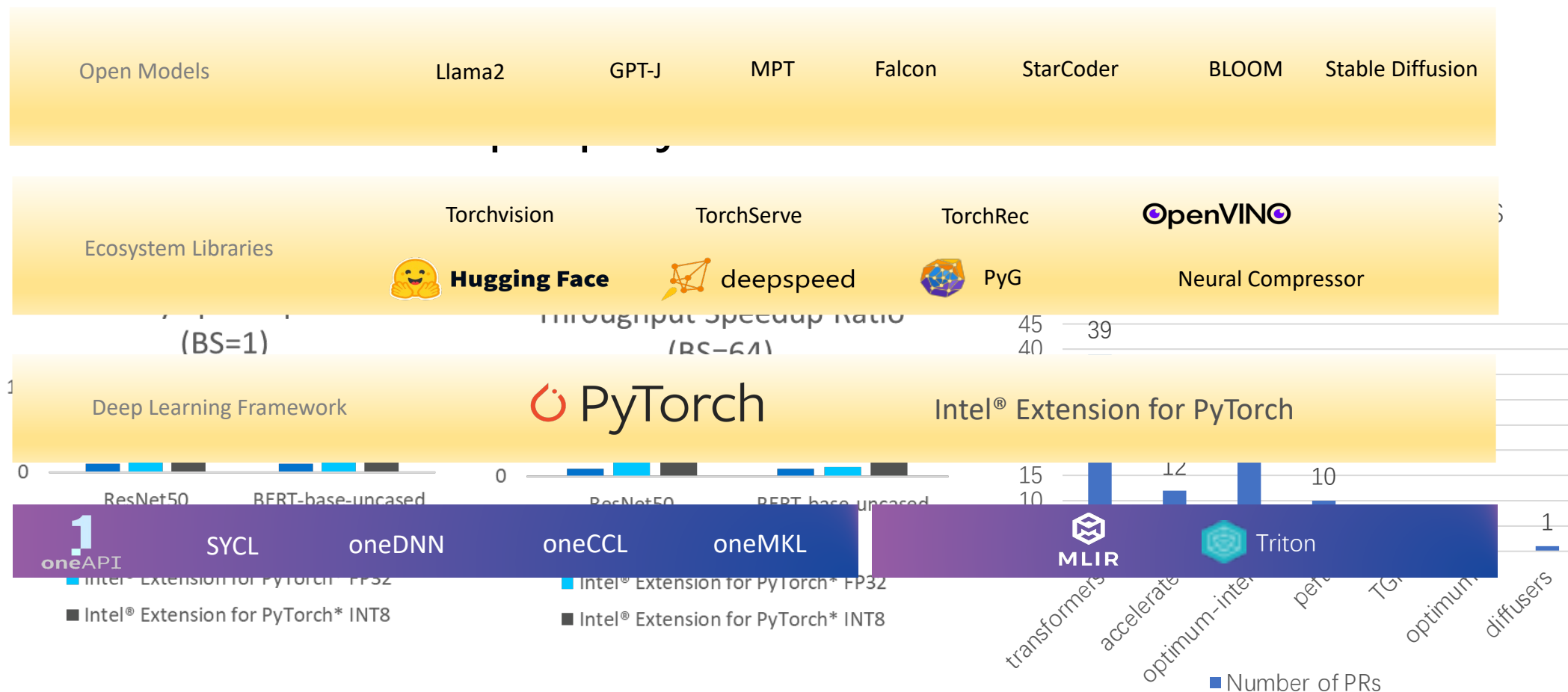
	Speedup	Throughput
LLaMA2 7B	2.47x	2.62x
LLaMA2 13B	2.57x	2.62x
GPT-J 6B	2.58x	2.85x

M7i performance boost ratio over M6i for INT8 quantized models:

	Speedup	Throughput
LLaMA2 7B	1.27x	1.38x
LLaMA2 13B	1.27x	1.27x
GPT-J 6B	1.29x	1.36x



Intel Contributions to PyTorch Ecosystem



* Other names and brands may be claimed as the property of others

Resources

IPEX-LLM: <https://github.com/intel/intel-extension-for-pytorch/tree/main/examples/cpu/inference/python/llm>

<https://github.com/intel/intel-extension-for-pytorch>

<https://intel.github.io/intel-extension-for-pytorch/>

<https://intel.github.io/intel-extension-for-pytorch/latest/tutorials/contribution.html>



Welcome to Feedbacks and Contributions!

THANKS

THANKS

THANKS

THANKS