Boosting AI Inferencing for LLM Models on Intel CPU-Powered Lenovo Servers, Part 2

Planning / Implementation

In Part 1 of this series of papers , we analyzed the performance of different LLM architectures—encoder-only, decoder-only, and encoder-decoder—on Intel CPU-powered Lenovo ThinkEdge SE450 server. The study evaluated latency, throughput, batch size, and CPU utilization, offering insights into how each architecture scales under various workloads.

In Part 1, we found that encoder-only models provided stable throughput and efficient CPU utilization, while decoder-only models demonstrated strong scaling capabilities but required higher memory. Encoder-decoder models exhibited the highest computational demand, impacting latency and overall performance. The findings helped businesses understand the trade-offs between performance and resource consumption for different LLM types.

Building upon these findings, this second paper in the series explores how Intel hardware techniques and Intel Extension for PyTorch (IPEX) accelerate AI inference, speeding up real-world AI applications. Our focus shifts from general benchmarking to hardware and software acceleration and optimization strategies that enhance large-scale AI deployments.

Let’s introduce what is AMX, AVX-512 and IPEX:

- AMX (Advanced Matrix Extensions): A hardware-level accelerator designed for deep learning, optimized for matrix multiplication, and supporting low-precision inference (BF16, INT8), available in Intel Xeon 4th Gen CPUs and newer.
- AVX-512 (Advanced Vector Extensions 512): An Intel CPU instruction set that enhances parallel processing with 512-bit wide vector operations, boosting performance for AI inference, cryptography, and scientific computing.
- IPEX (Intel Extension for PyTorch): A software-level optimization framework that enhances all types of PyTorch deep learning workloads by improving computation efficiency on Intel CPU.

In this paper, we conduct three different inference configurations:

- IPEX + AMX: IPEX optimization with Intel Advanced Matrix Extensions enabled.
- IPEX + AVX-512: IPEX optimization with Intel AVX-512 (AMX disabled).
- Baseline PyTorch: Standard PyTorch without IPEX or AMX optimizations.

Key takeaways for readers:

- Understand how Intel’s hardware and software optimizations enhance AI inference performance.
- Identify the most suitable LLM architecture for various real-world AI applications.
- Learn how to leverage Intel Extension for PyTorch for optimized AI workloads.

Background

Different LLM architectures are optimized for specific AI tasks. Understanding their strengths and best use cases is crucial for selecting the right model for AI inference.

Table 1. LLM Architectures and Real-World Use Cases

Architecture	Best for Tasks Like	Examples
Encoder-Only	Sentiment Analysis, Text Similarity, Named Entity Recognition	BERT, RoBERTa
Decoder-Only	Text generation, Creative Writing, Conversational AI	GPT-3, OPT, Llama
Encoder-Decoder	Translation, summarization, QA	T5, BART

The rest of the paper demonstrates how AI inference can achieve significant improvements in efficiency, speed, and scalability by leveraging Intel's hardware and software optimizations,

Methodology

To understand the impact of Intel's hardware and software optimizations, we evaluate three LLMs representing different architectures:

- **BERT-Base-NER (Encoder-Only)**: A Named Entity Recognition (NER) model based on BERT-Base, commonly used for text classification and entity extraction in documents.
- **OPT-350M (Decoder-Only)**: A text generation model optimized for generating coherent and contextually accurate text, widely used in chatbots and conversational AI.
- **m2m100_418M (Encoder-Decoder)**: A translation model designed to handle multilingual machine translation, enabling seamless language conversion in real-world applications.

These models were chosen to reflect diverse NLP tasks, allowing a comprehensive comparison of inference performance under various workloads.

To quantify the effectiveness of Intel's optimizations, we assess the following key performance metrics:

1. **Latency**: The time taken to process a single inference request.
2. **Throughput**: The number of inference requests handled per second.
3. **Concurrent Users**: The number of parallel inference requests supported efficiently.
4. **CPU Utilization (Profiler)**: Measures the compute allocation of different model layers during inference workloads.

Results and Analysis

In this section, we present the performance evaluation of our approach based on key metrics, including latency, throughput, concurrent users, and CPU utilization. The analysis is conducted using benchmark tests under controlled environments to ensure reliability and accuracy. We compare our results against plain PyTorch model as baseline to assess improvements and identify potential bottlenecks.

Latency

Latency measures the time taken to process an inference request, lower latency ensures faster AI responses for real-time applications. In our test, we used a fixed 128 input token length with batch sizes range from 1 to 32 across all three model architectures.

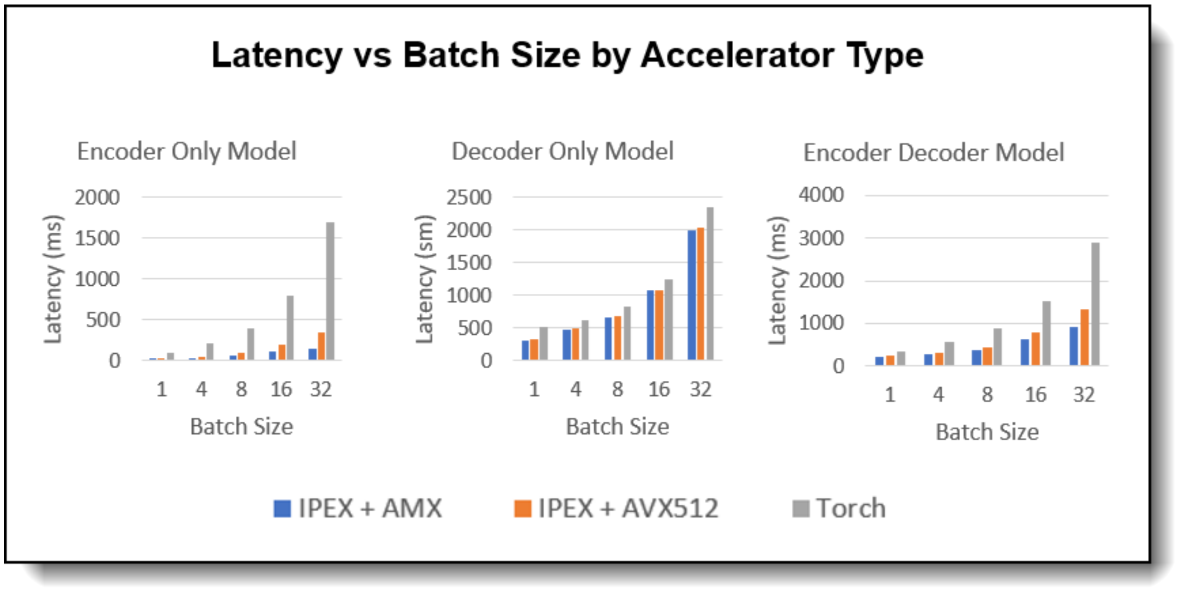

Figure 1. Latency vs Batch Size by Accelerator Type

Observations:

1. **Latency Improvement**: The latency improvement is most pronounced in bigger batch sizes but remains significant even as the batch size smaller for Encoder Only and Encoder Decoder Model. It shows the model benefits more from IPEX + AMX in increased workloads.
2. **Model Comparison**: Encoder-only model demonstrated better scaling efficiency with increased batch sizes and input token length, it can boost up to 10x latency decrease in AI inference

Throughput

Throughput defines the number of requests processed or token generated per second. Higher throughput indicates better scalability for large-scale AI deployments. we used a fixed 128 input token length with batch sizes range from 1 to 32 across all three model architectures.

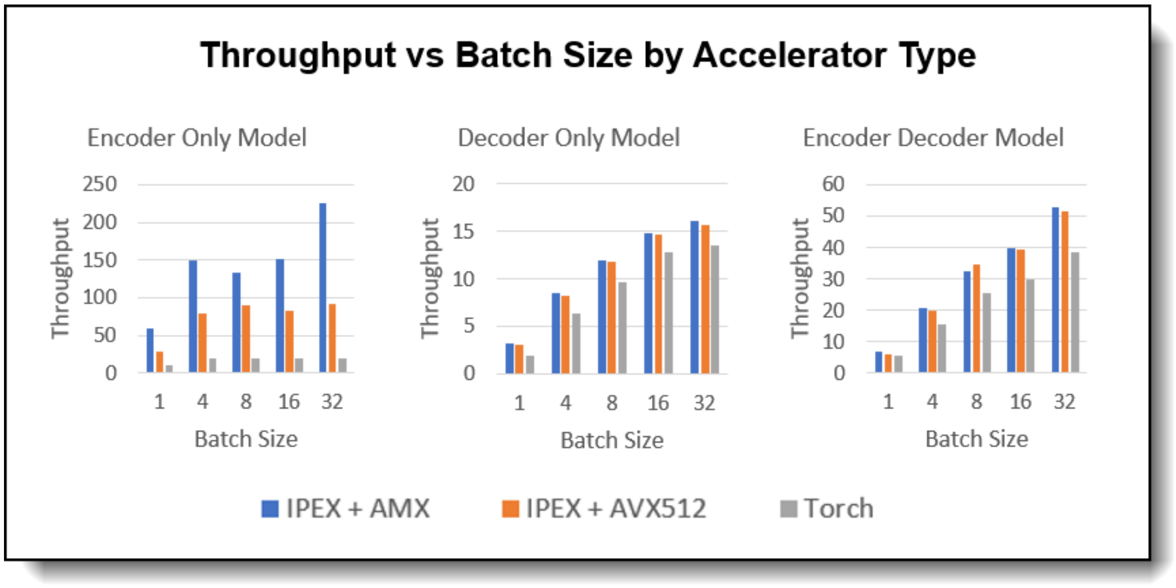

Figure 2. Throughput vs Batch Size by Accelerator Type

Observations:

1. **Throughput Improvement**: The throughput advantage of **IPEX + AMX** becomes more significant as the batch size increases, demonstrating better scalability. However, same as latency, throughput optimized at batch size = 16 for Decoder Only Model.
2. **Model Comparison**: Encoder-only model showed best scaling capability with increased batch sizes, it can boost up to 6x throughput increase in AI inference. Followed by Encoder Decoder models which can still increase 1.5x throughput.

Concurrent Users

Concurrent Users evaluates how well the system handles multiple simultaneous inference requests. Higher concurrent user capacity means improved multi-user efficiency. In our test, we used a fixed 128 input token length with batch sizes range from 1, 8, 32 and simulate the number of concurrent users range from 1, 4, 8 users across all three model architectures.

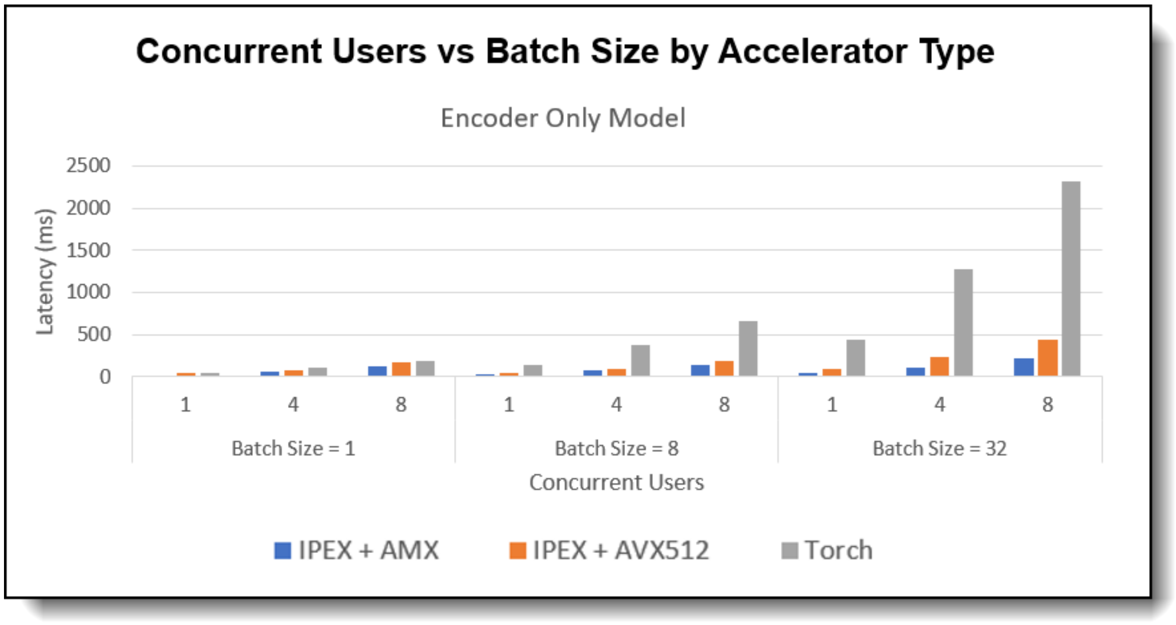

Figure 3. Concurrent Users vs Batch Size by Accelerator Type - Encoder Only Model

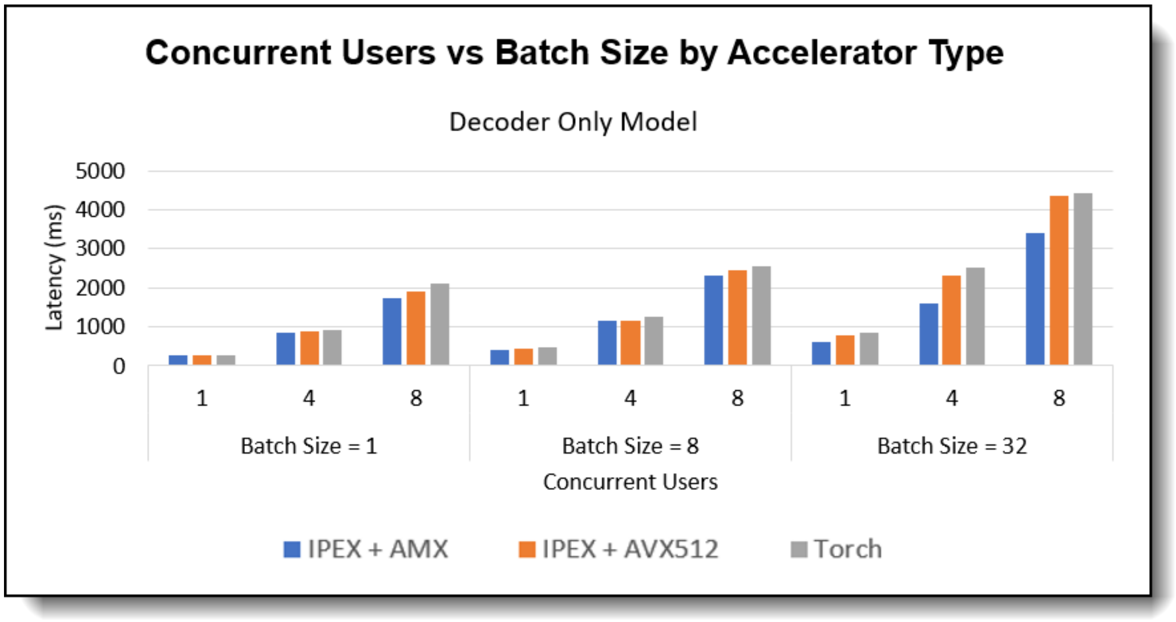

Figure 4. Concurrent Users vs Batch Size by Accelerator Type - Decoder Only Model

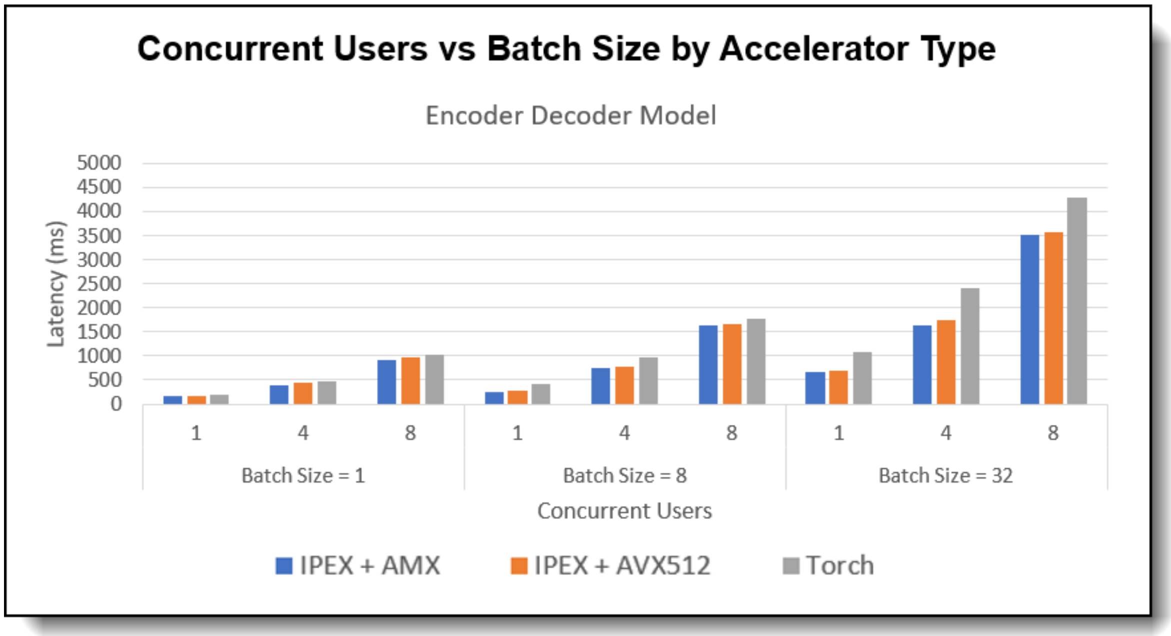

Figure 5. Concurrent Users vs Batch Size by Accelerator Type - Encoder Decoder Model

Observations:

1. **Concurrent User Improvement**: The chart demonstrated with higher batch size, as the number of concurrent users increases, IPEX + AMX consistently maintains lower latencies compared to IPEX + AVX-512 and Plain PyTorch, especially noticeable at higher concurrency level.
2. **Model Comparison**: Encoder-only model achieved best scaling capability with increased concurrent users and batch sizes. Decoder Only and Encoder-Decoder Model can also handle multiple concurrent users when choose a modest batch size.

CPU Utilization within Model

CPU Utilization within Model analyzes how efficiently 4th Gen Intel Xeon CPUs process AI workloads. We will dive deeper to visual how IPEX + AMX optimizes the model and quantify the benefit. In our test, we leveraged torch.profiler to visualize the each layer's CPU consumption within an model.

Firstly, let's recap what does AMX and IPEX optimize:

- AMX (Advanced Matrix Extensions): optimizes matrix multiplication and vector computations at the hardware level, enhancing AI, HPC, and data analytics workloads.
- IPEX (Intel Extension for PyTorch) optimizes PyTorch deep learning workloads at the software level by improving computation efficiency on Intel CPUs.

Secondly, we breakdown each of the 3 large language models and observe their architectures. From the following table, we discovered that Encoder only Modal benefits the most from AMX and IPEX due to its heavy structure on transformer layers. Roughly 87% of the parameters are influenced by the optimization.

Table 2. Parameter Distribution Across Model Architectures

Component	BERT-Base-NER	OPT-350M	m2m100_418M
Total Parameters	~110 million	~350 million	~418 million
Layers	12 Encoder layers	24 Decoder layers	12 Encoder + 12 Decoder layers
Embedding Layer	~23 million (~21%)	~35 million (~10%)	~100 million (~24%)
Transformer Layers	~96 million (~87%)	~288 million (~82%)	~336 million (~80%)
Output Layers	~2 million (~2%)	~27 million (~8%)	~17 million (~4%)
Optimizable Components	Transformer Layers (~87%)	Transformer Layers (~82%)	Transformer Layers (~80%)

Thirdly, we look one level down on the CPU work allocation for each LLM execution. We are most interested in the matrix multiplication and addition layer. Below chart is a summary from the outputs. I pick the top 1 CPU consumption subproject during the workload.

Table 3. CPU Consumption On Matrix Calculation Across Model Architectures

Architecture	Environment	Name	Self CPU %	Self CPU (s)
Encoder Only	IPEX AMX	torch_ipex::ipex_linear	55	3.36
	IPEX AVX-512	torch_ipex::ipex_linear	75	12.43
	PyTorch	aten::addmm	60	31
Decoder Only	IPEX AMX	torch_ipex::ipex_linear	29	2.09
	IPEX AVX-512	torch_ipex::ipex_linear	41	3.97
	PyTorch	aten::addmm	49	4.56
Encoder Decoder	IPEX AMX	torch_ipex::ipex_linear	25	5.97
	IPEX AVX-512	torch_ipex::ipex_linear	33	7.1
	PyTorch	aten::addmm	28	7.93

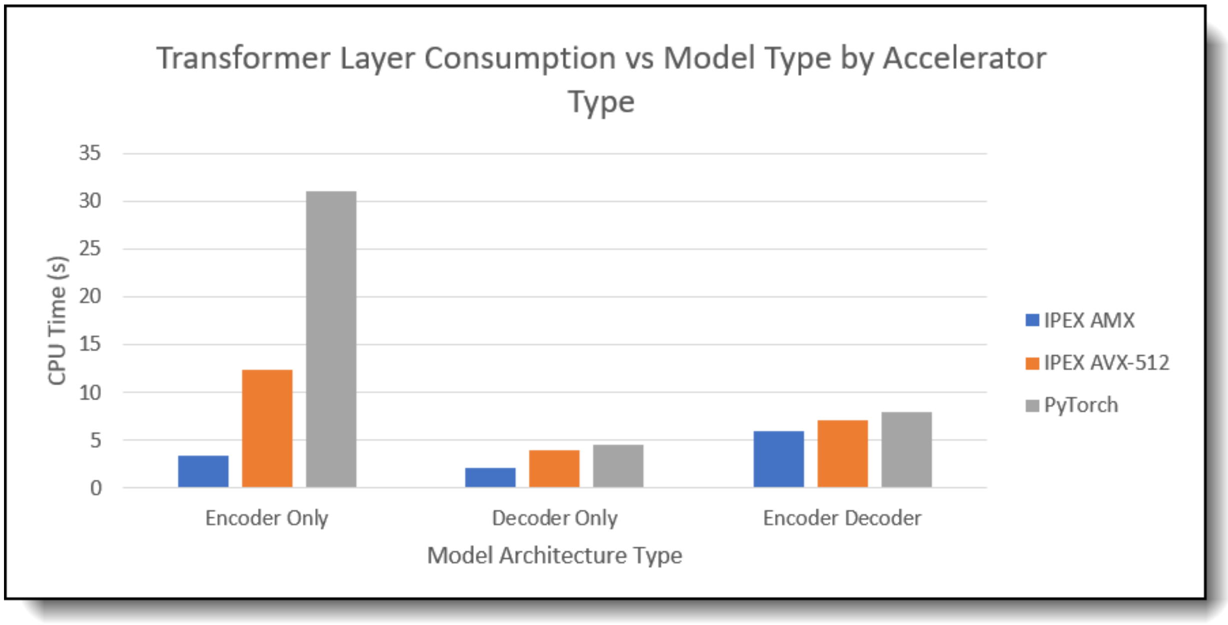

Figure 6. Transformer Layer Consumption VS Model Type by Accelerator Type

Observations:

- **CPU Consumption Optimization**: The chart and table prove IPEX + AMX consistently achieves the lowest execution time and CPU consumption during the workload across all model architectures.
- **Model Comparison**: Encoder-only Model benefits the most and speed up the matrix and vector computation up to 9x. Decoder-only Model comes next but still boost the performance by 2x.

Conclusion

This paper highlights the interplay between LLM architecture, hardware and software capability and limitation. By leveraging Intel's AMX and IPEX optimizations, businesses can achieve significant performance improvements. The insights provided here empower organizations to acknowledge the distinction among various LLMs and select the right AI solution.

The following table is a summary of all the tests we conducted through the research.

Table 4. Conclusion

Architecture	**Configuration**	**Avg Latency Lift***	**Throughput***	**Matrix Computation***
Encoder Only	IPEX AMX	7x - 10x faster	5x - 10x more	9x faster
	IPEX AVX-512	3x - 5x faster	2x - 5x more	3x faster
Decoder Only	IPEX AMX	1.3x -1.8x faster	1.2x - 2x more	2.5x faster
	IPEX AVX-512	1.1x - 1.3x faster	1.1x - 1.6x more	1.4x faster
Encoder Decoder	IPEX AMX	1.5x - 3x faster	up to 1.4x more	1.5x faster
	IPEX AVX-512	1.3x - 2x faster	up to 1.3x more	1.1x faster

* With Plain PyTorch Performance as Baseline

Final notes:

1. Encoder-dominant tasks (e.g., classification, text similarity) see the highest gains from AMX acceleration because these workloads involve large-scale batch matrix multiplications, which AMX's tile-based processing optimizes efficiently. The 7x-10x latency improvement and 9x faster matrix computation indicate that PyTorch's standard implementation struggles with memory-bound operations, while AMX significantly reduces data movement overhead.
2. Decoder-Only and Encoder-Decoder models benefit from optimizations but exhibit inherent scaling limitations due to sequential processing constraint, suggesting that autoregressive processing limit the benefits of AMX's tiling optimizations. The lower throughput gains further indicate that attention-based operations may still be memory-bandwidth constrained rather than pure compute-bound. Additional research required.
3. Intel AI optimizations significantly enhance concurrency handling, making large-scale AI inference deployments more feasible.

Future Work

As AI inference continues to evolve, optimizing hardware efficiency and cost-effectiveness remains a crucial area of exploration. Future research should focus on balancing performance with server capacity, identifying the best hardware configurations for specific business needs, and leveraging Intel-accelerated techniques to refine cost estimates. The following areas highlight key directions for further investigation

1. Test the model accuracy with lower data precision to increase server capacity and optimize computational efficiency.
2. Compare different hardware and pricing choices to determine the optimal infrastructure for various business use cases.
3. Refine hardware price estimates by incorporating Intel-accelerated techniques, such as Intel Neural Compressor and IPEX optimizations, to maximize price-to-performance efficiency.

Appendix - Test configuration

The following table lists the configuration of our server under test.

Table 5. Appendix - Test configuration

Component	Description
Server	Lenovo ThinkSystem SR650 V3
Processor	Intel Xeon Gold 6426Y processor
Sockets	2
Cores per Socket	32
Hyperthreading	Intel Hyper-Threading Technology Enabled
CPUs	64
Turbo	Intel Turbo Boost Technology Enabled
Base Frequency	1.8GHz
All-core Maximum Frequency	2.8GHz
Maximum Frequency	3.0GHz
NUMA Nodes	2
Installed Memory	512GB (16x32GB DDR5 5600 MT/s [5600 MT/s])
NIC	2x Ethernet Controller E810-XXV SFP 4x I350 Gigabit Network Connection 2x Ethernet Controller E810-C QSFP
Disk	16x 7TB Micron 7450 MTFDKCC7T6TFR 1x 894.3GB Micron 7450 MTFDKBA960TFR
BIOS	ESE128C-3.30
Microcode	0x2b000620
OS	Ubuntu 22.04.5 LTS
Kernel	5.15.0-131-generic

References

For more information, see these web sites:

- Liu, B. Comparative analysis of encoder-only, decoder-only, and encoder-decoder language models. College of Liberal Arts & Sciences, University of Illinois Urbana-Champaign, Champaign, IL. https://www.scitepress.org/Papers/2024/128298/128298.pdf
- Accelerating RAG Pipelines for Enterprise LLM Applications using OpenVINO on the Lenovo ThinkSystem SR650 V3 with 5th Gen Intel Xeon Scalable Processors. By Rodrigo Escobar, Abirami Prabhakaran, David Ellison, Ajay Dholakia, Mishali Naik. https://lenovopress.lenovo.com/lp2025-accelerating-rag-pipelines-for-llms-using-openvino-on-sr650-v3
- Intel Extension for PyTorch Github: https://github.com/intel/intel-extension-for-pytorch
- Intel Advanced Matrix Extensions (Intel AMX) Overview. https://www.intel.com/content/www/us/en/products/docs/accelerator-engines/advanced-matrix-extensions/overview.html

Authors

Kelvin He is a AI Data Scientist at Lenovo. He is a seasoned AI and data science professional specializing in building machine learning frameworks and AI-driven solutions. Kelvin is experienced in leading end-to-end model development, with a focus on turning business challenges into data-driven strategies. He is passionate about AI benchmarks, optimization techniques, and LLM applications, enabling businesses to make informed technology decisions.

David Ellison is the Chief Data Scientist for Lenovo ISG. Through Lenovo's US and European AI Discover Centers, he leads a team that uses cutting-edge AI techniques to deliver solutions for external customers while internally supporting the overall AI strategy for the World Wide Infrastructure Solutions Group. Before joining Lenovo, he ran an international scientific analysis and equipment company and worked as a Data Scientist for the US Postal Service. Previous to that, he received a PhD in Biomedical Engineering from Johns Hopkins University. He has numerous publications in top tier journals including two in the Proceedings of the National Academy of the Sciences.

Related product families

Product families related to this document are the following:

- Artificial Intelligence

Notices

This document, LP2167, was created or updated on March 5, 2025.

Send us your comments in one of the following ways:

- Use the online Contact us review form found at:
  https://lenovopress.lenovo.com/LP2167
- Send your comments in an e-mail to:
  comments@lenovopress.com

This document is available online at https://lenovopress.lenovo.com/LP2167.

Trademarks