Implementing AI Agents without GPUs: High-Performance Inference on Intel Xeon 6

Planning / Implementation

AI Agents have emerged as the next evolution for large language models (LLM), enabling models to make their own function calls independent of user input. Additionally, by augmenting the models with external knowledge retrieval in the form of Retrieval Augmented Generation (RAG) these agents can improve their flexibility and reduce hallucinations. RAG systems are increasingly used in applications such as enterprise search, customer support automation, technical documentation assistants, compliance analysis, and internal knowledge management. Traditionally, deploying performant RAG systems at scale has relied heavily on GPU-based infrastructure, introducing challenges related to cost, power consumption, resource availability, and operational complexity.

This paper demonstrates how efficient, production-ready RAG inference can be achieved without GPUs, using Intel Xeon processors. Leveraging Advanced Matrix Extensions (AMX) Intel Xeon processors deliver high-throughput, low-latency inference for both embedding generation and LLM-based text generation.

Through a practical reference architecture built with vLLM and LangChain, this paper illustrates how organizations can deploy scalable, cost-efficient RAG systems on Intel Xeon processors while maintaining strong performance and enterprise-grade reliability. Performance insights, architectural considerations, and deployment guidance are provided to demonstrate how CPU-optimized RAG inference reduces total cost of ownership, simplifies infrastructure, and enables broader adoption of generative AI across on-premises and hybrid environments, proving that high-quality RAG inference is no longer reliant on GPUs.

Environment Setup

This section walks through how to set up the necessary environment configuration required to deploy the AI agent.

Create Virtual Environment

It is recommended to install the uv package to create a virtual python environment to avoid dependency issues and quickly download needed python libraries.

Run the following in your Linux terminal:

```
curl -LsSf https://astral.sh/uv/install.sh | sh
uv venv agent-env
source agent-env/bin/activate
```

Setup Docker

Follow these steps to install Docker and build a CPU-optimized vLLM image.

1. Install Docker by following the instructions here and verifying it works by running the hello-world image: https://docs.docker.com/engine/install/
2. Clone the vLLM github repo.

```
git https://github.com/vllm-project/vllm.git
```

3. Build CPU-optimized vLLM image and set flags to take advantage of Intel's AMX and AVX acceleration. vLLM will use OneDNN on the backend automatically to best optimize performance.

```
cd vllm
sudo docker buildx build --platform=linux/amd64 --build-arg VLLM_CPU_AMXBF16=1 --build-arg V
LLM_CPU_AVX512=1 --build-arg VLLM_CPU_AVX512BF16=1 --build-arg VLLM_CPU_AVX512VNNI=1 --targe
t vllm-openai -t vllm-cpu-amx -f docker/Dockerfile.cpu --load .
```

4. Start the vLLM Docker container and download LLM model from HuggingFace, add a HuggingFace token if your desired model is gated such as in the case of Llama3 models.

```
docker run -it --name vllm-amx --rm -p 8000:8000 \
  -v ~/.cache/huggingface:/root/.cache/huggingface \
  -e HF_TOKEN="your_hf_token_here" \
  vllm-cpu-amx meta-llama/Llama-3.1-8B-Instruct \
  -- \
  --enable-auto-tool-choice \
  --tool-call-parser llama3_json
```

5. Check that the vLLM container is running and available for inferencing calls after allowing it time to startup.

```
curl http://localhost:8000/v1/models
```

Agentic AI & RAG Setup

With the vLLM server running, the following steps build the full RAG-enabled agent pipeline using LangChain.

1. Download the dependencies
   Install the necessary python libraries.

```
uv pip install -q langchain langchain-openai langchain-huggingface langchain-core langchain-
community langchain-text-splitters beautifulsoup4
```

2. Import the packages
   Inside of a python script we can import the needed packages.

```
from langchain_openai import ChatOpenAI, OpenAIEmbeddings
import os
from langchain.chat_models import init_chat_model
```

```
from langchain_huggingface import HuggingFaceEmbeddings
from langchain_core.vectorstores import InMemoryVectorStore
import bs4
from langchain_community.document_loaders import WebBaseLoader
from langchain_text_splitters import RecursiveCharacterTextSplitter
from langchain.tools import tool
from langchain.agents import create_agent
```

3. Initialize the Embedding Model and Vector Store
   The embedding model converts text into 768-dimensional vectors used for semantic similarity search. It runs entirely on CPU alongside the LLM server, with no resource contention between the two processes since embedding generation is fast and infrequent relative to LLM decoding.

```
embeddings = HuggingFaceEmbeddings(model_name="sentence-transformers/all-mpnet-base-v2")
vector_store = InMemoryVectorStore(embeddings)
```

4. Load and Chunk the Source Documents
   Source documents are loaded, parsed, and split into overlapping chunks before being embedded and stored. The chunk overlap preserves sentence-level context across split boundaries, improving retrieval quality for documents with dense information. For this example, we use the Beautiful Soup library to extract the main text of a simple blog post concerning LLMs.

```
# Load a web page, extracting only the main post content
bs4_strainer = bs4.SoupStrainer(class_=("post-title", "post-header", "post-content"))
loader = WebBaseLoader(
    web_paths=("https://lilianweng.github.io/posts/2023-06-23-agent/",),
    bs_kwargs={"parse_only": bs4_strainer},
)
docs = loader.load()

# Split into 1000 character chunks with 200 character overlap
text_splitter = RecursiveCharacterTextSplitter(
    chunk_size=1000,
    chunk_overlap=200,
    add_start_index=True,
)
all_splits = text_splitter.split_documents(docs)

# Embed all chunks and index them in a vector store
document_ids = vector_store.add_documents(documents=all_splits)
```

   In an enterprise deployment, the WebBaseLoader would be replaced with loaders suited to the document corpus: PyPDFLoader for internal PDFs, DirectoryLoader for a file system, or a custom loader targeting a CMS, SharePoint, or database.

5. Configure the LLM Client
   The LangChain LLM client connects to the local vLLM server. Because vLLM exposes a standard OpenAI REST API, no custom integration code is needed and the vLLM server can be easily updated and scaled independently of the agent code.

```
VLLM_BASE_URL = "http://localhost:8000/v1"
MODEL_NAME = "meta-llama/Llama-3.1-8B-Instruct"

llm = ChatOpenAI(
    openai_api_base=VLLM_BASE_URL,
    openai_api_key="EMPTY",  # vLLM doesn't require a real key
    model_name=MODEL_NAME,
    temperature=0.1,
    max_tokens=512,
)
```

6. Define the Retrieval Tool
   The retrieval tool is the bridge between the agent's reasoning loop and the vector store. When the LLM determines that external context is needed, it emits a structured tool call that LangChain executes automatically, returning the top matching chunks as context for the next generation step.

```
@tool(response_format="content_and_artifact")
def retrieve_context(query: str):
    """Retrieve information to help answer a query."""
    retrieved_docs = vector_store.similarity_search(query, k=2)
    serialized = "\n\n".join(
        (f"Source: {doc.metadata}\nContent: {doc.page_content}")
        for doc in retrieved_docs
    )
    return serialized, retrieved_docs
```

   The k=2 parameter returns the two most semantically similar chunks. Increasing k broadens coverage at the cost of a longer prompt and higher per-token latency. More sophisticated tools such as a database query function, calculator, or API client can be added to the tools list to extend the agent's capabilities without modifying any other part of the pipeline.

7. Construct the Agent and Run Inference
   With the LLM client and retrieval tool defined, the agent is assembled with a single call. LangChain's create_agent factory builds a ReAct-style agent (Reasoning + Acting) that manages the think-act-observe loop automatically.

```
tools = [retrieve_context]
prompt = (
    "You have access to a tool that retrieves context from a blog post. "
    "Use the tool to help answer user queries."
)
agent = create_agent(llm, tools, system_prompt=prompt)

query = (
    "What is the standard method for Task Decomposition?\n\n"
    "Once you get the answer, look up common extensions of that method."
)
```

```
# stream() emits each reasoning step as it is produced,
# enabling real-time display of tool calls and intermediate responses.
for event in agent.stream(
    {"messages": [{"role": "user", "content": query}]},
    stream_mode="values",
):
    event["messages"][-1].pretty_print()
```

Xeon 6 vLLM Inferencing Performance

Benchmarking was performed across three 7-8B parameter models using the Lenovo ThinkSystem SR650 V4 powered by Intel's Xeon 6 processors:

- Llama-3.1-8B-Instruct
- Qwen3-8B
- Mistral-7B

Table 1. Output token generation rate 512 Input / 512 Output

Users	**Llama-8B**			**Qwen3-8B**			**Mistral-7B**		
	Tok/s	Tok/s/user	TTFT (ms)	Tok/s	Tok/s/user	TTFT (ms)	Tok/s	Tok/s/user	TTFT (ms)
1	10.16	10.16	197.02	9.57	9.57	195.27	11.63	11.63	187.35
2	20.08	10.04	276.59	18.78	9.39	282.47	22.93	11.46	277.62
4	39.31	9.82	484.56	36.85	9.21	475.27	44.94	11.23	467.69
8	72.90	9.11	772.23	70.41	8.80	815.79	82.74	10.34	769.82
16	138.25	8.64	1425.55	128.71	8.04	1482.89	156.41	9.77	1413.50
32	200.39	6.26	2130.28	191.87	5.99	2216.58	222.34	6.94	2147.51
64	295.69	4.62	3515.06	276.29	4.31	3664.53	338.21	5.28	3449.08
128	405.50	3.16	6238.52	385.30	3.01	6508.17	464.16	3.62	6206.06

Table 2. Output token generation rate 512 Input / 1024 Output

Users	**Llama-8B**			**Qwen3-8B**			**Mistral-7B**		
	Tok/s	Tok/s/user	TTFT (ms)	Tok/s	Tok/s/user	TTFT (ms)	Tok/s	Tok/s/user	TTFT (ms)
1	10.15	10.15	199.74	9.57	9.57	195.63	11.63	11.63	184.90
2	20.00	10	298.26	18.73	9.36	301.45	22.90	11.45	277.03
4	39.17	9.79	488.66	36.67	9.16	475.66	44.76	11.19	469.86
8	72.42	9.05	776.91	69.81	8.72	819.00	81.98	10.24	772.95
16	136.36	8.52	1431.88	126.65	7.91	1480.38	153.90	9.61	1415.63
32	195.61	6.11	2129.49	186.57	5.83	2240.58	216.23	6.75	2135.45
64	286.28	4.47	3481.94	266.47	4.16	3624.84	325.14	5.08	3446.06
128	386.00	3.01	6321.50	364.87	2.85	6512.12	439.44	3.43	6227.17

At a single concurrent user, all three models achieved sub-200ms time-to-first-token (TTFT) regardless of input length, with Mistral-7B reaching 11.63 tok/s, Llama-3.1-8B at 10.16 tok/s, and Qwen3-8B at 9.57 tok/s on the 512-input/512-output benchmark. These latencies are well within the thresholds required for interactive agent use cases and still easily scale to 16-32 users without excessive speed degradation.

Throughput scales efficiently under concurrent load making it optimal for use for multiple concurrent users or a small agent swarm. In the 512/512 configuration, aggregate output token generation for Mistral-7B grew from 11.63 tok/s at one user to 464.16 tok/s at 128 concurrent users (a roughly 40x increase) demonstrating that the platform's Intel AMX-accelerated compute cores are effectively utilized as the request queue deepens. Llama and Qwen exhibited comparable scaling curves, reaching 405.50 tok/s and 385.30 tok/s respectively at 128 users.

Table 3. Xeon 6 vLLM Inferencing Performance

Users	**Llama-8B**			**Qwen3-8B**			**Mistral-7B**		
	Tok/s	Tok/s/user	TTFT (ms)	Tok/s	Tok/s/user	TTFT (ms)	Tok/s	Tok/s/user	TTFT (ms)
1	10.13	10.13	197.53	9.55	9.55	200.46	11.58	11.58	187.39
2	19.83	9.91	303.16	18.57	9.28	310.26	22.69	11.34	289.92
4	38.54	9.63	504.56	36.05	9.01	490.03	43.95	10.98	484
8	70.4	8.8	794.53	67.81	8.47	855.69	79.64	9.95	799.66
16	129.47	8.09	1498.18	120.05	7.50	1561.77	145.27	9.07	1494.71
32	181.22	5.66	2252.81	172.04	5.37	2364.43	198.39	6.19	2243.36
64	257.64	4.02	3644.35	240.05	3.75	3851.81	289.51	4.52	3622.54
128	335.25	2.61	6595.97	314.73	2.45	6881.8	373.42	2.91	6489.93

Conclusions

The results presented in this paper demonstrate that GPU infrastructure is no longer a prerequisite for deploying production-ready AI agents. By combining the Intel Xeon 6740P's Advanced Matrix Extensions with the vLLM inference server and LangChain's agent framework, the Lenovo ThinkSystem SR650 V4 delivers consistent, low-latency inference across a range of 7-8B parameter models all on CPU. The implementation walkthrough further illustrates that the architecture is straightforward to deploy, requiring no custom integration code and remaining fully compatible with the broader vLLM, LangChain, and HuggingFace ecosystems.

For enterprises evaluating their AI infrastructure strategy, the SR650 V4 represents a compelling path to adoption. CPU-only inference eliminates the supply chain constraints and high capital expenditure associated with GPU procurement, while Intel AMX acceleration ensures that performance scales predictably with concurrent load. Whether deployed as a standalone agent host or scaled horizontally across nodes to serve larger user populations, the Lenovo ThinkSystem SR650 V4 powered by Intel Xeon 6 offers a cost-effective, operationally simple, and enterprise-grade foundation for the next generation of AI-powered applications.

Hardware Details

The following table lists the configuration of the server we used in our tests.

Table 4. Server configuration

Feature	Description
Server model	Lenovo ThinkSystem SR650 V4
Processor	2x Intel Xeon 6740P 48C 270W 2.1GHz
Installed Memory	16x Samsung 64GB TruDDR5 6400MHz (2Rx4) 10x4 16Gbit RDIMM
Disk	4x ThinkSystem 2.5" U.2 PM9D3a 1.92TB Read Intensive NVMe PCIe 5.0 x4 HS SSD
OS	Ubuntu 22.04.5 LTS (GNU/Linux 6.8.0-60-generic x86_64)

Author

Eric Page is an AI Engineer at Lenovo. He has 6 years of practical experience developing Machine Learning solutions for various applications ranging from weather-forecasting to pose-estimation. He enjoys solving practical problems using data and AI/ML.

Related product families

Product families related to this document are the following:

- Artificial Intelligence
- Processors

Notices

Lenovo may not offer the products, services, or features discussed in this document in all countries. Consult your local Lenovo representative for information on the products and services currently available in your area. Any reference to a Lenovo product, program, or service is not intended to state or imply that only that Lenovo product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any Lenovo intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any other product, program, or service. Lenovo may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

Lenovo (United States), Inc.
8001 Development Drive
Morrisville, NC 27560
U.S.A.
Attention: Lenovo Director of Licensing

LENOVO PROVIDES THIS PUBLICATION ”AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. Lenovo may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

The products described in this document are not intended for use in implantation or other life support applications where malfunction may result in injury or death to persons. The information contained in this document does not affect or change Lenovo product specifications or warranties. Nothing in this document shall operate as an express or implied license or indemnity under the intellectual property rights of Lenovo or third parties. All information contained in this document was obtained in specific environments and is presented as an illustration. The result obtained in other operating environments may vary. Lenovo may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Any references in this publication to non-Lenovo Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this Lenovo product, and use of those Web sites is at your own risk. Any performance data contained herein was determined in a controlled environment. Therefore, the result obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

This document, LP2406, was created or updated on March 24, 2026.

Send us your comments in one of the following ways:

- Use the online Contact us review form found at:
  https://lenovopress.lenovo.com/LP2406
- Send your comments in an e-mail to:
  comments@lenovopress.com

This document is available online at https://lenovopress.lenovo.com/LP2406.

Trademarks

Lenovo and the Lenovo logo are trademarks or registered trademarks of Lenovo in the United States, other countries, or both. A current list of Lenovo trademarks is available on the Web at https://www.lenovo.com/us/en/legal/copytrade/.

The following terms are trademarks of Lenovo in the United States, other countries, or both:
Lenovo®
ThinkSystem®

The following terms are trademarks of other companies:

Intel®, the Intel logo and Xeon® are trademarks of Intel Corporation or its subsidiaries.

Linux® is the trademark of Linus Torvalds in the U.S. and other countries.

SharePoint® is a trademark of Microsoft Corporation in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.