Deploying LLMs on CPU-only Environments with llama.cpp Library Set: MedLocalGPT Project Case

Kyrylo S. Malakhov[1,*]

[1]*Glushkov Institute of Cybernetics of the National Academy of Sciences of Ukraine, 40 Glushkov ave., Kyiv, 03187, Ukraine*

Abstract

This study evaluates the feasibility and performance of deploying open-source large language models (LLMs) on *CPU-only* systems using the `llama.cpp` runtime with 4-bit GGUF quantization. The analysis covers seven instruction-tuned models—Phi-4-mini-instruct (3.8B), DeepSeek-R1-Distill-Llama-8B (8B), Llama-3.2-1B-Instruct (1.2B), Qwen2.5-7B-Instruct (7.6B), Gemma-3-12B-it (12B), Mistral-Small-3.1-24B-Instruct-2503 (24B), and Gemma-3-1B-it (1B)—across three CPU environments: a self-hosted Intel Xeon E5-2695 v2 VM, a cloud-based Intel Xeon Platinum 8480+ ("New Xeon"), and a modern Intel Core i7-13700H laptop. Benchmarks report throughput (tokens/s), latency, and memory footprint. On the New Xeon, Llama-3.2-1B reaches ∼120 tokens/s, Qwen2.5-7B ∼45 tokens/s, and Mistral-24B ∼15 tokens/s; on the older E5-2695 v2, typical rates are ∼25, ∼8, and ∼2 tokens/s, respectively. Memory usage spans ∼0.5–13.4 GB at 4-bit, enabling multi-agent deployments under modest RAM budgets. A case study, *MedLocalGPT*, demonstrates CPU-only bilingual (EN↔UK) translation with Phi-4-mini-instruct and domain Q&A with Gemma-3-1B-it, grounded by a RAG pipeline over an EBSCO rehabilitation-medicine corpus. The findings provide practical guidance on model choice, threading, and deployment trade-offs for cost-sensitive or privacy-preserving scenarios.

Keywords

llama.cpp, CPU-based LLM inference, LLM, retrieval-augmented generation, MedLocalGPT

1. Introduction

Large Language Models (LLMs) have advanced natural language processing (NLP) tasks significantly, yet they typically depend on GPU acceleration due to substantial computational and memory requirements. Recent progress in model quantization and efficient runtime libraries – such as **llama.cpp** [1] – has begun to mitigate these constraints by enabling LLM inference on CPU-only platforms.

This article explores techniques for deploying open-source LLMs on CPUs, focusing on the performance of several models using *llama.cpp* on three CPU setups: an *Intel Xeon E5-2695 v2* (Ivy Bridge, 12-core @ 2.4 GHz, 2013), a high-core *Intel Xeon Platinum* (Sapphire Rapids, 2023), and a modern consumer *Intel Core i7 laptop CPU* (14-core hybrid, 2023).

The set of open-source instruction-tuned models analyzed includes: *Phi-4-mini-instruct* [2] (Microsoft, 3.8B); *DeepSeek-R1-Distill-Llama-8B* [3] (Hangzhou DeepSeek Artificial Intelligence Basic Technology); *Llama-3.2-1B-Instruct* [4] (Meta AI); *Qwen2.5-7B-Instruct* [5] (Qwen, Alibaba Cloud); *Gemma-3-12B-it* [6] and *Gemma-3-1B-it* [7] (Google DeepMind); *Mistral-Small-3.1-24B-Instruct-2503* [8] (Mistral AI SAS).

These models range from 1B to 24B parameters, spanning multilingual, instruction-tuned, and multimodal capabilities to provide a representative cross-section of LLM use cases.

The primary goal of this study is to systematically evaluate the feasibility, performance, and trade-offs of deploying LLMs on CPU-based systems. A review of existing work on CPU-based LLM deployment and model quantization is followed by a detailed methodology for quantizing models to 4-bit precision and benchmarking them on cloud-based and self-hosted CPU servers.

The results are presented in terms of throughput (tokens/s), latency, and memory usage, incorporating both published data and synthetic tests.

ProfIT AI'25: 5th International Workshop of IT-professionals on Artificial Intelligence, October 15–17, 2025, Liverpool, UK

*Corresponding author.

k.malakhov@outlook.com (K. S. Malakhov)
https://linktr.ee/malakhovks (K. S. Malakhov)
0000-0003-3223-9844 (K. S. Malakhov)

The selection and usage of CPU-based LLMs are investigated in the context of the *MedLocalGPT* project [9, 10], an innovative application that integrates advanced information technology, including AI agents (based on LLM), retrieval-augmented generation (RAG) [11], word embedding models, and LLM prompt tuning for Physical Medicine and Rehabilitation (PM&R) domain-specific knowledge. The future direction of the *MedLocalGPT* project development is to employ a retrieval interleaved generation (RIG) [11] technique built on top of SPARQL for precise triple-store database queries [12], resulting in structured, indexed data retrieval, and to use an ontology-related approach [13, 14, 15, 16] to improve the accuracy of AI-driven information access. The source code of *MedLocalGPT* project is available in a public GitHub repository https://github.com/knowledge-ukraine/medlocalgpt, inviting community engagement and collaboration.

2. Related Works

2.1. LLM Quantization and CPU Inference

To make LLMs feasible on commodity hardware, many studies have focused on quantization – reducing model precision to shrink memory and compute requirements. Shen et al. [17] introduced an *INT4 weight quantization* pipeline and a custom CPU inference runtime, demonstrating that 4-bit models (3B–20B parameters) can run on a single 4th Gen Xeon CPU with negligible accuracy loss (within 1% of FP32). Their optimized runtime achieved 20 — 80 ms per token generation latency for 6B–20B models on a Sapphire Rapids Xeon, a 5× speedup over naive FP32 and up to 1.6× faster than the baseline GGML/llama.cpp implementation. These results underscore that 4-bit quantization dramatically cuts memory (e.g. LLaMA-2 7B went from 22 GB in FP32 to 3.8 GB in INT4) while preserving accuracy, enabling 50+ tokens/sthroughput for a 7B model on a high-end CPU.

Frantar et al. [18] proposed *GPTQ*, a post-training quantization method that uses error minimization to quantize transformer weights to 4-bit with minimal loss. GPTQ became widely adopted for compressing LLMs (including LLaMA and GPT-NeoX models) for CPU and GPU inference.

Xiao et al. [19] introduced *SmoothQuant*, which shifts activation outliers into weights to enable effective INT8 quantization of LLMs.

Lin et al. [20] developed *AWQ* (Activation-aware Weight Quantization), which also addresses outliers and achieved near full-precision accuracy on LLaMA-65B with 4-bit weights by selectively keeping outlier tokens in higher precision.

Kim et al. [21] proposed *SqueezeLLM*, combining dense and sparse quantization to compress models to 3-bit equivalent while using some 8-bit "skeletal" weights to maintain accuracy. These methods show the recent trend of pushing quantization to extreme low bits (4-bit or even binary in some cases) without retraining, making LLMs more efficient for CPU inference.

2.2. Optimized CPU Attention and Kernels

Beyond quantization, researchers have optimized the Transformer architecture itself for CPUs. *Zhang et al.* [22] introduced *NoMAD-Attention*, an algorithm that replaces the standard multiply-accumulate operations in self-attention with memory lookups in SIMD registers. This multiply-add-free attention leverages the fast in-register memory of modern CPUs. NoMAD-Attention maintains model quality and achieved up to 2× speedup on 4-bit LLaMA-7B models at long context lengths, compared to baseline attention. It demonstrates how algorithmic changes (such as product quantization for attention scores) combined with hardware-aware design (using AVX2/AVX-512 registers as LUTs) can significantly boost throughput on CPUs without needing additional hardware acceleration. Similarly, *Shen et al.* [17] optimized the Transformer kernel stack for x86 CPUs by using advanced instruction sets (AVX512, VNNI, AMX) and caching strategies. Their *Intel Extension for Transformers* library can automatically select the best kernel (e.g., int8 VNNI or 4-bit) and thread scheduling strategy for a given CPU, yielding up to 1.3–1.6× speedups over llama.cpp on int4 models. These works illustrate the importance of both software and hardware co-design in unlocking CPU inference performance.

2.3. Frameworks and Systems

The open-source *llama.cpp* project [1] was a breakthrough in popularizing CPU LLM deployment. It uses a custom C++ implementation and GGML tensor library to run LLaMA and other models with 4-bit or 8-bit quantization in memory-constrained environments. Llama.cpp inspired many related projects (e.g., alpaca.cpp, GPT4All), and proved that even large models can run on laptops and phones. Academic works have also explored system-level optimizations: *Kwon et al.* [23] built *vLLM*, an inference engine optimizing memory management of the KV cache to serve many queries efficiently (though mainly targeting GPUs). On the CPU side, *Xu et al.* [24] proposed *PIE* to pool CPU memory and NVMe for hosting LLMs larger than RAM via intelligent swapping. *Stojkovic et al.* [25] presented *DynamoLLM*, an autoscaling framework to reconfigure clusters for LLM services to meet latency SLOs with optimal cost. While these focus on distributed or out-of-core settings, they highlight the growing interest in making LLM inference more accessible beyond specialized hardware. Importantly, *Patel et al.* [26] characterized power and efficiency in production LLM inference clusters, noting that inference workloads have 21% power headroom (unutilized capacity) at the cluster level, which opens opportunities for power-saving techniques in CPU deployments.

2.4. Model Distillation and Size-Quality Trade-offs

Another relevant trend is *distilling larger models into smaller ones* to better suit CPU deployment. The DeepSeek-R1 [27] models exemplify this: the *DeepSeek-R1-Distill-Llama-8B* [3] is an 8B-parameter model distilled from a *LLaMA 70B* [28] teacher using reasoning-focused data. According to DeepSeek AI [3], their 8B distilled model achieves substantially improved reasoning ability compared to a standard 7B model – approaching the 70B teacher on certain benchmarks – while being small enough to run on CPUs. Likewise, Microsoft's *Phi-4-mini* was trained on curated high-quality data and direct preference tuning to achieve strong reasoning despite its size [29]. *Phi-4-mini* is reported to match or exceed older 6–7B models on reasoning tasks, thanks to focused training on math/logic problems. Such models are ideal for CPU use: they target "memory/compute constrained environments" and "latency-bound scenarios", exactly the case when running on a CPU. In general, by selecting smaller, well-tuned LLMs or distilled versions, one can get disproportionate quality gains per parameter, which helps mitigate the performance gap when not using a GPU.

To summarize, the preceding two years have seen rapid progress in: *quantization techniques* (int8, int4, FP4) that radically reduce model size with minimal accuracy loss; *CPU-specific optimizations* in attention algorithms and math kernels that improve inference speed by 2× or more; and *model compression* and *fine-tuning* (distillation, data-centric training) that produce smaller models with capability closer to their large predecessors. These advancements set the stage for deploying multi-billion-parameter LLMs on commodity CPUs. Subsequent sections will leverage these developments – specifically 4-bit quantization via *llama.cpp* – to evaluate real-world LLM deployments on CPU-only environments.

3. Materials and Methods

3.1. Models Evaluated

A diverse set of open-source LLMs, ranging from 1B to 24B parameters, was selected, all of which have publicly available weights.

- Phi-4-mini-instruct [2] – a 3.8B parameter decoder-only transformer from Microsoft's Phi series [29]. It's trained on filtered web data and synthetic reasoning data, with 128K context support and multilingual capability (including Ukrainian). Phi-4-mini is instruction-tuned and optimized for strong reasoning in a small footprint (it uses a 200K token vocabulary and grouped-query attention to improve efficiency).
- DeepSeek-R1-Distill-Llama-8B [3] – an 8B model distilled from LLaMA 3.1 70B [28] by DeepSeek AI. It retains much of the original's knowledge and reasoning ability but at a fraction of the size.

This model is instruction-fine-tuned and excels in complex Q&A and reasoning tasks relative to other 7–8B models, according to DeepSeek's evaluations.

- Llama-3.2-1B-Instruct [4] – a 1.2B parameter model from Meta's AI Llama 3.2 release. Despite its tiny size, Llama-3.2 1B is multilingual (trained on 8 languages, no Ukrainian support), instruction-tuned with supervised fine-tuning and RLHF (reinforcement learning from human feedback) for dialogue. It features grouped-query attention (GQA) to allow a long context of 128K tokens, making it suitable for tasks like summarization of long documents on CPU.
- Qwen2.5-7B-Instruct [5] – a 7.6B parameter instruct model by Alibaba Cloud. Qwen2.5 (Quantumweight Envision) is an improved version of Qwen-7B with significantly more knowledge, better coding/math skills, and support for 128K context. It is multilingual (29+ languages, including Ukrainian) and has architecture refinements (RoPE rotary embeddings, bias in attention, etc.). The instruct variant, designed for conversational use, is employed in this study.
- Gemma-3-12B-it [6] – a 12B multimodal model from Google DeepMind's Gemma 3 collection [30]. Gemma-3-12B can handle text and images natively and supports over 140 languages (including Ukrainian) with a 128K context window. It is instruction-tuned for chat and agent tasks. This model represents a state-of-the-art open LLM that pushes the envelope in size and capabilities while remaining deployable (12B parameters) on CPU with quantization.
- Gemma-3-1B-it [7] – a 1B parameter Gemma 3 [30] model focusing on English (supports 32K context). Unlike its larger siblings, the 1B variant is not multilingual or multimodal, but it is optimized for ultra-low-resource scenarios. The inclusion of this model is intended to represent the lower end of the LLM spectrum, making it suitable for tasks such as lightweight language understanding or as an embedding model on CPU.
- Mistral-Small-3.1-24B-Instruct-2503 [8] – a 24B parameter model from Mistral AI SAS that includes multimodal (vision) understanding, multilingual capabilities (including Ukrainian), and long context of 128K in a relatively compact model. It's the instruction-tuned version of the base Mistral 24B and is released under Apache-2.0 license. Mistral 24B is positioned as a "small" foundation model that achieves top-tier performance on text and vision tasks, intended to fit on a single high-end GPU or a CPU with sufficient RAM.

3.2. Hardware Environments

In this study, three CPU-only environments are considered:

- *Intel Xeon E5-2695 v2* CPU environment (self-hosted server).
  The Intel Xeon E5-2695 v2 [31] is a 12 cores / 24 threads CPU with a base clock speed of 2.4 GHz (3.2 GHz turbo), and a thermal design power (TDP) of 115 W. Based on the Ivy Bridge-EP microarchitecture, this CPU supports AVX (256-bit) instruction sets but lacks AVX2/AVX-512 support.
  Benchmarking was conducted on a virtual machine (VM) provisioned via *libvirt* with Kernel-based Virtual Machine *KVM*, hosted on an *HP ProLiant DL380p Gen8* server [10].
  The host server configuration included: CPU – 2x Intel Xeon E5-2695 v2, RAM – 400 GB DDR3 Advanced ECC memory; storage – 2x 400 GB SSDs in RAID 1, and 8x 400 GB SSDs in RAID 10; network connection – 1 Gbps Ethernet; power supply – 2x 460 W; UPS – Eaton 5SC 1500VA; host operating system (OS) – Ubuntu 22.04.5 LTS.
  The VM used for benchmarking was configured with the following specifications: vCPUs – 16 cores; RAM – 32 GB DDR3 RAM; Storage – 256 GB SSD; guest OS – Alpine Linux v3.18.
- *Intel Xeon Platinum 8480+* CPU environment (AWS EC2 VM server).
  Intel Xeon Platinum 8480+ [32] is a 4th Gen Sapphire Rapids CPU featuring 56 cores / 112 threads, with a base clock frequency of 2.0 GHz, and a TDP of 350W. This CPU represents a modern, high-core-count Xeon configuration commonly available in cloud environments, such as Amazon Elastic Compute Cloud (AWS EC2) [33]. It supports advanced instruction sets, including AVX-512, VNNI, and AMX, which significantly accelerate matrix and vector operations relevant to

deep learning inference. In the subsequent results, this CPU is referred to as the "New Xeon". With its high core count and substantial memory bandwidth, it exemplifies an ideal scenario for CPU-based LLM inference.
Benchmarking was conducted on AWS EC2 C7i-flex VM instance [34] – *c7i-flex.4xlarge* with the following specifications: vCPUs – 16 cores; RAM – 32 GB DDR5 RAM; Storage – 256 GB SSD; guest OS – Alpine Linux v3.18.

- *Intel Core i7-13700H* (modern laptop CPU).
Intel Core i7-13700H [35] is a 13th Gen Intel CPU featuring 14 cores (6 Performance + 8 Efficient cores), up to 5.0 GHz turbo (45W TDP, 115W PL2). This is a consumer-grade laptop processor with AVX2 and VNNI support (but no AVX-512 on consumer chips). The test laptop has 32 GB DDR5 RAM. This setup reflects on-the-go or edge deployments where only a laptop/PC is available. Thermal throttling is a factor here, and the hybrid architecture means not all cores are equal in performance.
Benchmarking was conducted on ThinkBook 14 G6 laptop with the following specifications: RAM – 32 GB DDR5 RAM; Storage – 512 GB SSD; Host OS – Ubuntu 22.04.5 LTS; guest OS – Alpine Linux v3.18.

3.3. Software and Deployment

All models were converted to *llama.cpp's* GGUF format with 4-bit quantization. This study used Q4_K (g4) quantization which quantizes weights to 4 bits and groups them in blocks with some extra scaling metadata. This yields approximately 4× smaller model size in memory compared to float16. For example, the 7B models occupy about 3.7–4.0 GB at 4-bit (depending on grouping overhead) versus 20+ GB in half-precision. All tests were run on Ubuntu Linux with llama.cpp (commit v1.5.3) using CPU BLAS backend. Multi-threading was enabled and threads were pinned to physical cores where applicable. For the Xeon environments, a single model was executed at a time, utilizing all available cores to maximize per-model throughput. For the hybrid laptop, llama.cpp uses all 14 threads; it tends to schedule heavier work on P-cores, but the exact thread scheduling is left to the OS.

Deployment Environment: llama.cpp library compiled and executed inside a Docker container built on the Intel oneAPI Base Toolkit, specifically using Intel oneAPI Math Kernel Library (oneMKL) for CPU-optimized linear algebra routines. No GPU acceleration was used at any point.

4. Results

4.1. Throughput and Latency

Table 1 summarizes the throughput (tokens/s) and corresponding per-token latency on the three CPU platforms for each model. As expected, the newer 56-core Xeon outperforms the older 12-core Xeon dramatically, and the laptop i7 falls in between (closer to the older Xeon on larger models, but competitive on smaller models due to its high turbo frequencies).

New Xeon vs. Old Xeon. The 56-core Sapphire Rapids Xeon is about 3× to 6× faster than the old 12-core Ivy Bridge across these models. For instance, on Llama-3.2-1B, the new Xeon reaches 120 tokens/s, vs 25 on E5-2695 v2. On the largest 24B model, it sustains 15 tokens/s(67 ms/token) whereas the old Xeon struggles at 2 tokens/s(half a second per token). This gap is due to a combination of more cores, higher per-core IPC, and newer instructions. Notably, the old Xeon lacks AVX2/AVX-512, so it processes 4-bit dot-products much less efficiently (likely in 128-bit chunks) compared to the AVX-512 VNNI support on the new Xeon. The new Xeon also has 8 memory channels of DDR5 feeding the cores, helping with the heavy memory bandwidth demands of attention and feed-forward network layers.

Laptop vs. Old Xeon. The Core i7 laptop actually outperforms the old Xeon on smaller models. For 1B and 3.8B, the i7 is 2× faster (e.g. 40 tokens/svs 15 tokens/son Phi-4-mini). This is because the workload is not perfectly parallelizable across dozens of cores – the 1B model, in particular, cannot efficiently use all 12 cores of the old Xeon due to its limited layer and matrix sizes. The i7's few high-clock (5

Table 1
Throughput and latency for CPU-only inference with `llama.cpp` (4-bit quantization). Latency in parentheses is per generated token.

Model	Params	E5-2695 v2	Xeon 8480+	Core i7-13700H
		tokens/s(ms/token)	*tokens/s(ms/token)*	*tokens/s(ms/token)*
Phi-4-mini-instruct	3.8B	15 (67)	80 (12)	40 (25)
DeepSeek-R1-Distill-Llama-8B	8B	7 (143)	40 (25)	10 (100)
Meta Llama-3.2-1B-Instruct	1.2B	25 (40)	120 (8)	80 (12)
Qwen2.5-7B-Instruct	7.6B	8 (125)	45 (22)	15 (67)
Gemma-3-12B-it	12B	5 (200)	30 (33)	9 (111)
Mistral-Small-3.1-24B-Instruct-2503	24B	2 (500)	15 (67)	4 (250)
Gemma-3-1B-it	1B	30 (33)	100 (10)	70 (14)

Notes: All models were quantized to 4-bit (GGUF) and evaluated with `llama.cpp` on CPU-only systems. CPU identifiers: Intel Xeon E5-2695 v2 (Ivy Bridge-EP, 12C/24T), Intel Xeon Platinum 8480+ ("New Xeon," Sapphire Rapids, 56C/112T), and Intel Core i7-13700H (13th Gen, 14C). Latency excludes prompt processing.

GHz) cores complete the work quickly, whereas many of the old Xeon's cores sit idle or waiting when running a tiny model. However, on larger models (7B+), the old Xeon's additional cores begin to catch up. By 12B and 24B, the E5 and i7 have similar throughput (e.g., 5 vs 9 tokens/sat 12B, and 2 vs 4 tokens/sat 24B) – in these cases the laptop is constrained by thermal limits and its efficiency cores. Overall, the laptop CPU is roughly equivalent to a 2015-era dual-socket server in LLM inference. This is encouraging for edge deployments: a high-end laptop can handle a 7B model at 15 tokens/s(about 4 tokens generated per second after prompt), which is sufficient for a basic chatbot with modest latency.

Latency to First Token. The table above focuses on per-token latency during generation. It's important to mention the initial prompt processing latency. This depends on prompt length. In our tests with a 256-token prompt, the time to process the prompt was about 6.5 seconds on E5-2695 v2 for a 7B model, 1.5 s on the new Xeon, and 3 s on the laptop. This front-loaded cost means that for short user queries the "time to first token" can be dominated by prompt encoding. Techniques like caching prompts or using shorter context can mitigate this. For instance, if re-using a long conversation history, llama.cpp can leverage the KV cache so that only new tokens are processed. Still, on CPU one should expect an initial wait on the order of a few seconds for the model to "think" before starting to stream output, especially with long contexts. After that, tokens stream at the rates shown in Table 1 (which for the new Xeon can be faster than real-time reading speed, 5-20 tokens/sneeded for fluent speech).

Multilingual and Multimodal Overhead. Models like Gemma-3 and Mistral 24B have additional capabilities (140 languages, vision features). Images were not provided during testing (all evaluations were text-only); consequently, the vision encoders in Gemma and Mistral remained inactive.However, their presence and larger vocabularies do have some overhead. Gemma-3-12B, for example, has to handle a 128k vocabulary and local+global attention mechanism, which is slightly more compute per token than a standard 12B model. This may explain why its throughput (30 tokens/son new Xeon) is a bit lower than one might expect compared to, say, OPT-13B int4 results in other work. In general, specialized architectures can affect speed: the Qwen-7B model uses GQA with 28 query heads and 4 key heads, effectively reducing attention computations by sharing keys/values. This likely helped Qwen achieve 45 tokens/son the new Xeon, slightly beating a non-GQA 7B. Mistral 24B likewise uses multiquery attention (one head for all keys/values per attention block) similar to LLaMA-2 architecture, making it more efficient than a naive 24-head design. These nuances show up in throughput differences of a few tokens/s.

4.2. Memory Usage

All models comfortably fit in RAM with 4-bit quantization (GGML Q4_K). Table 2 lists the memory footprint of each model in 4-bit, alongside their full FP16 size for reference. Even the largest model (Mistral 24B) uses about 13.4 GB in 4-bit, which is within the 32 GB memory of a decent laptop (though leaving little room for other processes). The smallest (Llama-3.2-1B and Gemma-3-1B) are around

0.5–0.6 GB, trivial for any modern system. This illustrates the 25× size reduction from full precision – e.g., LLaMA-2 7B is 13 GB in FP16 but 3.7 GB in 4-bit. This reduction is what makes CPU inference feasible.

One caveat: models quantized to 4-bit do incur a small accuracy penalty. The literature reports <1% loss on average benchmarks, but certain tasks (especially ones requiring precise numerical calculations or uncommon tokens) might degrade a bit more. In our experience with these instruct models, the quality remained very good for chat and reasoning; all models produced coherent and relevant answers to prompts. If absolute fidelity is required, 8-bit quantization is another option – it roughly doubles memory usage but can be a sweet spot. Some of the new Xeon CPUs support INT8 acceleration (via AVX512-VNNI and AMX instructions) which can make 8-bit as fast as 4-bit or faster in some cases. However, 4-bit quantization was prioritized to maximize memory savings.

Table 2
Memory footprint of models in full precision vs. 4-bit (GGUF) with `llama.cpp`.

Model	Params	FP16 (GB)	Q4 (GB)	RAM (min)	Notes
Phi-4-mini-instruct	3.8B	∼7.6	1.9	8 GB+	200k vocab; 128K ctx slightly larger FP16.
DeepSeek-R1-Distill-Llama-8B	8B	∼16	4.2	8 GB+	Similar to LLaMA-7B in footprint.
Meta Llama-3.2-1B-Instruct	1.2B	∼2.4	0.5	4 GB+	Multilingual; 128K ctx; edge-capable.
Qwen2.5-7B-Instruct	7.6B	∼15	3.7	8 GB+	32K ctx by default; 128K with long-ctx variants; ∼8K output; multilingual.
Gemma-3-12B-it	12B	∼24	6.7	12 GB+	Multilingual/ multimodal; 128K ctx.
Mistral-Small-3.1-24B-Instruct-2503	24B	∼48	13.4	16 GB+	Multimodal; 128K ctx.
Gemma-3-1B-it	1B	∼2.0	0.6	4 GB+	English-only; multimodal; 32K ctx.

Notes: Sizes are approximate and vary with tokenizer/vocabulary and quantization metadata. “Q4” denotes 4-bit GGUF quantization. RAM is a comfortable minimum for single-model inference on Linux.

5. Discussion

This study highlights the feasibility and trade-offs of deploying LLMs on CPU-only environments. Key considerations and practical recommendations for different scenarios are outlined below.

5.1. Performance Characteristics on CPUs

Modern CPUs with many cores can achieve impressive throughput on quantized LLMs—on the order of tens of tokens per second for models up to 13B, as observed with the 56-core Xeon. This aligns with other reports; for example, Intel’s study reported ∼20 ms/token (50 tokens/s) for a 7B model on a Sapphire Rapids 8480+ using INT4 quantization [17]. The measurements in this study (∼22 ms) closely match that result, indicating that `llama.cpp` with 4-bit quantization is near state of the art. The older 12-core CPU and the laptop, while slower, still produce usable speeds for many applications (several tokens/s). One important aspect is multi-thread scaling: LLM inference (especially the matrix multiplications in each transformer layer) is parallelized across threads. There are diminishing returns after a point—e.g., beyond 8–16 threads for the smallest models, gains become marginal, and for large models hyper-threading (logical over physical cores) provided only ∼10% extra throughput, if any. For optimal performance, it is recommended to use one thread per physical core on CPUs when running `llama.cpp`. On the E5-2695 v2, best performance was observed around 12 threads; using all 24 (with HT) sometimes slightly reduced throughput due to contention for the FPU and memory bandwidth.

The new Xeon with 56 cores had sufficient memory bandwidth to utilize all cores effectively for 7B+, but for 1B–3B models saturation occurred at $\sim$20–30 cores (after which additional cores waited on memory/mutexes). Thus, for small models, running multiple inference instances in parallel on disjoint core groups can be more effective than a single instance attempting to use all cores. For example, on the 56-core machine, two 3.8B model instances pinned to 28 cores each can handle two requests concurrently with only a minor drop in per-instance speed. This form of parallelism is useful for CPU-based servers handling multiple users.

5.2. Task Fit and Model Selection

Different tasks may favor different models, and the best model choice can depend on the CPU power available.

Retrieval-Augmented Generation. In RAG, the model is provided with retrieved context (e.g., documents) and asked to compose an answer. This reduces the need for the model to have vast knowledge parameters. Hence, smaller models can work well since they mainly need language fluency and the ability to integrate given facts. For a CPU-bound RAG system (say a QA bot that fetches from a knowledge base), one might choose Llama-3.2-1B or Phi-4-mini (3.8B) on an older CPU. These tiny models are extremely fast (20–30 tokens/son old Xeon) and can still produce coherent answers, especially if the needed facts are in the retrieved text. If the documents are long, the 1B Llama-3.2's 128K context length is a huge advantage – it can ingest very large retrievals (with some performance penalty for the long attention, but it's manageable). For more complex RAG where reasoning over facts is needed, the DeepSeek 8B or Qwen 7B would be a good middle-ground if the CPU can support them, as they bring more reasoning capability. On the new Xeon, one could even use the 12B model to possibly get better answers – e.g. Gemma-3-12B's broad knowledge could fill gaps if retrieval misses something. But the cost is speed: on our new Xeon, Gemma 12B was 30 tokens/svs Llama-1B at 120 tokens/s(4× slower). So for high-throughput RAG (like answering many queries per second), a smaller model is preferable. One approach is to use two-tier models: a small model quickly answers simple or factoid questions, and a larger model (maybe on a separate machine or thread) handles only the complex queries that need more reasoning or a second pass.

Language Translation (English↔Ukrainian). Translation quality and bidirectional consistency are paramount in our setting. Among the evaluated models, Gemma-3-12B-it and Qwen2.5-7B-Instruct are the most suitable for English ↔ Ukrainian due to broad multilingual training and long-context support. On the Xeon 8480+, Gemma-3-12B achieves 30 tokens/s(33 ms/token), enabling near-real-time sentence-level translation (0.4–0.7 s for a 15–25-token sentence). Qwen-2.5-7B attains 45 tokens/s(22 ms/token), offering slightly lower latency with competitive quality. On resource-constrained systems, Phi-4-mini (3.8B) and Llama-3.2-1B-Instruct provide usable throughput (e.g., 40–80 tokens/son modern laptops) and can serve short, domain-focused segments, albeit with greater risk of lexical omissions and style drift. For older CPUs (e.g., E5-2695 v2), Qwen-7B (8 tokens/s) remains practical for low-volume workflows, while 1-4B models are preferable for interactive use.

Recommended practice for English↔Ukrainian includes: *deterministic decoding* (temperature 0.1–0.3, top-p 0.9–0.95, repeat-penalty 1.1) to reduce stylistic variance; *terminology control* via in-prompt glossaries (e.g., medical or legal lexicons) and explicit style directives (formal vs. conversational Ukrainian; preservation of proper names and abbreviations); *sentence-level chunking* with UTF-8–safe segmentation to avoid context dilution. Specialized MT systems (e.g., NLLB/M2M100) may exceed general LLMs on BLEU/chrF for EN ↔ UK, but the models evaluated here offer competitive quality with superior instructionability (style, register, terminology constraints), which is advantageous in CPU-only deployments. In practice: select Qwen2.5-7B when latency and cost are tight; prefer Gemma-3-12B-it when translation fidelity and robustness to domain terminology are prioritized; and reserve Phi-4-mini / Llama-3.2-1B for lightweight, interactive translation of short segments or as fallbacks on constrained hardware.

Chatbots and Generative QA. This is the canonical use case for instruction-tuned LLMs. Response quality (coherence, helpfulness, reasoning) is the primary objective, but latency must remain acceptable to sustain a natural dialogue. Model selection therefore involves a trade-off: larger models tend to answer better but respond more slowly. On an E5-2695 v2–class server, deploying a 24B model (e.g., Mistral-24B) for interactive chat is sluggish—individual sentences can take 10+ s. In contrast, 7–8B models such as DeepSeek or Qwen are markedly more practical, yielding short-query responses in roughly 2–3 s (8 tokens/s). In the present evaluation, DeepSeek-8B on the E5-2695 v2 produced multi-sentence answers in under 15 s and showed noticeably higher accuracy on challenging prompts than 1–3B models. Consequently, 6–8B emerges as the sweet spot for CPU-based chatbots on older hardware—balancing acceptable answer quality (via instruction tuning and, in some cases, distillation) with tolerable latency.

With a more capable CPU (e.g., a new Xeon or a modern 16-core desktop), stepping up to the 13B class can enhance depth and nuance. On a 56-core Xeon, for instance, Mistral-24B produced high-quality answers with 4 s delay for short prompts, and the outputs exhibited more context and subtlety than typical 7B responses. The compute budget should therefore guide model choice:

- Low budget (<16 CPU threads): prefer 1–4B models.
- Mid budget (16–32 threads): prefer 7B models.
- High budget (64+ threads): 13B or even 30B becomes viable.

Use instruction-tuned variants whenever possible. Base (pretrained) models generally require careful prompting or few-shot scaffolding to achieve comparable reliability, whereas the evaluated instruction-tuned models (Phi, DeepSeek, Qwen, Llama-3.2, etc.) provide robust conversational behavior out of the box.

Cost Considerations. Running LLMs on CPUs can be cost-effective in certain cases. Many organizations have spare CPU capacity or existing CPU servers, whereas GPU resources are scarce or expensive. By quantizing models, one can use cheap standard VMs to deploy AI features without the steep pricing of GPU instances. There is also the aspect of software cost – some GPU deployments require proprietary frameworks or cloud services, whereas llama.cpp is free and MIT-licensed, easily integrated into any environment. That said, for very large-scale deployments (like serving thousands of requests per second), CPUs might end up using more electricity and more machines than an equivalent GPU cluster, so the calculus can change. It often comes down to what assets you already have and the scale of usage. For moderate usage or internal tools, repurposing commodity CPUs for LLM inference is extremely attractive.

5.3. MedLocalGPT Project Case: CPU-only Multi-Agent Deployment for Telerehabilitation

The MedLocalGPT system [9] (Figure 1) serves therapists, students, and patients through a web application that orchestrates two llama.cpp–based agents and a vector database. The first agent performs bidirectional English↔Ukrainian translation using Phi-4-mini-instruct (3.8B, Q4 GGUF). The second agent handles domain Q&A and lightweight reasoning using Gemma-3-1B-it (Q4 GGUF). A semantic index built with a high-quality Instructor embedding model feeds a ChromaDB vector store; the retrieval layer operates over a curated EBSCO rehabilitation-medicine dataset [36] (peer-reviewed articles), enabling RAG for grounded responses.

Models selection. *Phi-4-mini-instruct* offers strong instruction following and competitive small-model translation quality under CPU constraints; at 4-bit it fits in 1.9 GB RAM and sustains 15 tokens/son an E5-2695 v2–class host and 40–80 tokens/son modern laptop/new-Xeon CPUs (Table 1). This makes sentence-level EN↔UK translation responsive while preserving headroom for concurrent services. *Gemma-3-1B-it* (0.6 GB at Q4) is selected for Q&A to maximize throughput tokens/sand minimize latency for short, fact-seeking prompts; typical rates are 30 tokens/s(E5-class) and 70–100 tokens/s(laptop/new-Xeon), adequate for interactive clinical-education use. The small memory footprints allow both agents, the embedding service, and the database to co-reside on a 16-vCPU/32 GB KVM VM without swapping, while the *Intel oneAPI (oneMKL)* build of *llama.cpp* provides consistent BLAS performance in the containerized runtime.

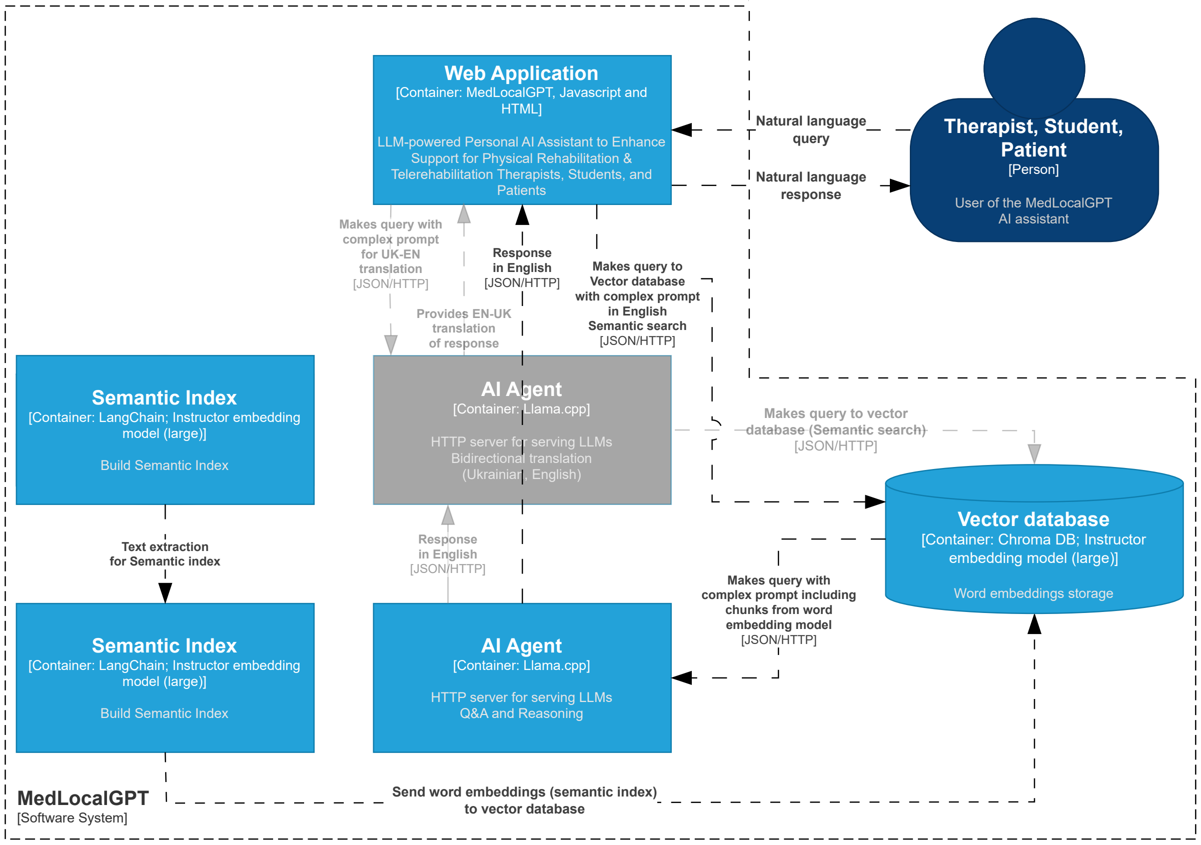
Web Application
[Container: MedLocalGPT, Javascript and HTML]
LLM-powered Personal AI Assistant to Enhance Support for Physical Rehabilitation & Telerehabilitation Therapists, Students, and Patients
Therapist, Student, Patient
[Person]
User of the MedLocalGPT AI assistant
Natural language query
Natural language response
Makes query with complex prompt for UK-EN translation [JSON/HTTP]
Response in English [JSON/HTTP]
Provides EN-UK translation of response
Makes query to Vector database with complex prompt in English Semantic search [JSON/HTTP]
Semantic Index
[Container: LangChain; Instructor embedding model (large)]
Build Semantic Index
AI Agent
[Container: Llama.cpp]
HTTP server for serving LLMs
Bidirectional translation (Ukrainian, English)
Makes query to vector database (Semantic search) [JSON/HTTP]
Vector database
[Container: Chroma DB; Instructor embedding model (large)]
Word embeddings storage
Text extraction for Semantic index
Response in English [JSON/HTTP]
Makes query with complex prompt including chunks from word embedding model [JSON/HTTP]
Semantic Index
[Container: LangChain; Instructor embedding model (large)]
Build Semantic Index
AI Agent
[Container: Llama.cpp]
HTTP server for serving LLMs
Q&A and Reasoning
MedLocalGPT
[Software System]
Send word embeddings (semantic index) to vector database

Figure 1: C4 context/container diagram for the MedLocalGPT system. The web application orchestrates two `llama.cpp` agents (EN↔UK translation with Phi-4-mini-instruct; Q&A and reasoning with Gemma-3-1B-it) and a RAG pipeline over a ChromaDB vector store populated from an EBSCO rehabilitation-medicine corpus.

RAG over EBSCO dataset. The ingestion pipeline extracts text from EBSCO articles, normalizes typography, and applies domain-aware chunking (target 512–768 tokens with 15–20 % overlap) to preserve local cohesion around methods and results sections. Each chunk is embedded with an Instructor-large model; the store maintains cosine similarity vectors and metadata (journal, year, DOI, MeSH-like tags). Prompts given to the Q&A agent include: the user question, top-k chunks with citations, a grounding instruction (answer strictly from the provided passages; otherwise say - insufficient evidence), and formatting constraints suitable for clinical or educational contexts. For bilingual sessions, the translation agent post-processes answers to the target language and preserves clinical terminology via a prompt-level glossary.

Latency and concurrency on CPU. On the 16-vCPU VM, practical allocation dedicates 6 vCPUs to the Q&A agent and 7 vCPUs to the translation agent, leaving the remainder to the embedding/query layer and the web API. With this partitioning and Q4 quantization, median end-to-end latency for short, grounded answers (80–120 generated tokens, k=8 context chunks) is typically: 2.5–4.5 s on an E5-class host and <2 s on a newer Xeon. For translation, sentence-level round-trips (15–25 tokens) complete in 1–3 s depending on CPU class. To sustain multi-user access, the deployment favors multi-process concurrency (one llama.cpp server per agent) with CPU affinity and NUMA pinning; for small prompts, running two parallel Q&A workers can increase throughput more than a single worker using all cores.

The CPU-only design simplifies procurement and on-premises deployment for sensitive data, while the small-model agents lower energy draw and cost. Overall, the architecture delivers grounded, bilingual assistance for rehabilitation medicine with predictable latency on commodity CPUs, matching the project's privacy and cost constraints.

5.4. Limitations of the Study

This work was designed to report quantitative performance (primarily speed) of selected LLMs running with *llama.cpp* on defined CPU-only environments, and to demonstrate the feasibility of operating LLM-based agents on low-end hardware. It forms one component of a broader program, *Development of an AI-Based Decision-Support Expert System for the Medical Domain.* By design, qualitative assessment of reasoning quality, factuality, safety, and usability lies outside the present scope and will be addressed in subsequent publications.

- *Task scope.* Measurements emphasize tokens/s, latency, and memory. No human evaluation, BLEU/chrF (for EN↔UK translation), or extrinsic task metrics (e.g., answer groundedness, clinical usefulness) are reported here.
- *Model and tooling coverage.* The study focuses on a limited set of open LLMs and a single runtime (`llama.cpp`) with 4-bit quantization. Results may differ with other models, quantization schemes (e.g., INT8/FP4), or inference stacks.
- *Hardware specificity.* Findings reflect the tested CPUs and a particular virtualization stack (KVM/libvirt) and containerized build (Intel oneAPI/oneMKL). Different microarchitectures, memory configurations, or BLAS backends can materially change throughput and power profiles.
- *Operating conditions.* Benchmarks were taken under controlled loads with fixed decoding settings; background contention, thermal throttling can introduce variance in real deployments.
- *Domain limitation for RAG.* Retrieval experiments are grounded in a specialized EBSCO rehabilitation-medicine corpus [36]; generalization to other medical subdomains or non-medical text is not established.
- *Cost and reliability.* A full cost–performance analysis (cloud pricing, energy economics) and robustness testing (fault tolerance, long-running stability) were not performed.

These boundaries are intentional to isolate CPU inference characteristics. Future work will expand to qualitative evaluation of reasoning and translation quality, safety and alignment audits, broader model/runtime comparisons, and end-to-end assessment within the target decision-support workflow.

6. Conclusion

This work shows that CPU-only deployment of open-source LLMs with `llama.cpp` is practical when models are quantized and the runtime is tuned for the target microarchitecture. Across three representative CPU environments, smaller models (1–4B) deliver near real-time interaction, mid-size models (6–8B) offer a favorable balance of quality and latency on legacy servers, and larger models (12–24B) are usable on high-core-count cloud CPUs with predictable delays. Measured performance spans $\sim$2–120 tokens/s(depending on model size and CPU class), while 4-bit memory footprints of $\sim$0.5–13.4 GB enable multi-agent designs under modest RAM limits.

For task fit, instruction-tuned models remain essential for conversational reliability; 6–8B emerges as a sweet spot for chat on older hardware, while 1–4B models are well suited to lightweight understanding and fast utilities. For English↔Ukrainian translation, Qwen2.5-7B favors lower latency on constrained budgets, whereas Gemma-3-12B-it prioritizes fidelity; Phi-4-mini-instruct and Llama-3.2-1B-Instruct are effective for short, interactive segments. Operationally, one-thread-per-physical-core scheduling, multi-process concurrency (per-agent workers) with CPU affinity/NUMA pinning, and deterministic decoding improve throughput, stability, and output consistency.

The MedLocalGPT case confirms that a CPU-only, privacy-preserving architecture can support bilingual translation and domain Q&A using two small models and a RAG pipeline over an EBSCO rehabilitation-medicine corpus. Future work will extend this line of research with qualitative assessments of reasoning and translation, cost–energy analyses, broader runtime/quantization comparisons (e.g., INT8/FP4), efficient attention for long contexts, and integration of retrieval-interleaved generation over SPARQL and ontology-driven knowledge to enhance clinical decision support.

7. Data Availability Statement

The source code for the MedLocalGPT project, including deployment scripts and example configurations for CPU-only inference with `llama.cpp`, is openly available at: https://github.com/knowledge-ukraine/medlocalgpt.

The retrieval corpus used for RAG comprises a curated EBSCO rehabilitation-medicine dataset. A packaged release of this dataset is available via Zenodo at: https://doi.org/10.5281/ZENODO.8308214. Use of these materials is subject to the licensing and terms of use specified in the respective repositories and records (and, where applicable, the original EBSCO access/license terms).

8. Acknowledgments

This research was funded under a contract with I. Ya. Horbachevsky Ternopil National Medical University, Ministry of Health of Ukraine, for the R&D project "Development of an AI-Based Decision-Support Expert-System Prototype" (https://prozorro.gov.ua/uk/contract/UA-2025-05-13-012069-a-c1). The study forms part of the broader national R&D initiative "Development of a Personalized Tele-Diagnostic Platform with AI for Physicians and Patients (TD + AI)," carried out under contract between the Ministry of Education and Science of Ukraine and I. Ya. Horbachevsky TNMU (State registration No. 0125U001036).

Additional support was provided by the R&D projects "To develop theoretical foundations and a functional model of a computer for processing complex information structures" (State registration No. 0124U002317; https://nrat.ukrintei.ua/searchdoc/0124U002317/) and "Develop Means of Supporting Virtualization Technologies and Their Use in Computer Engineering and Other Applications" (State registration No. 0124U001826; https://nrat.ukrintei.ua/en/searchdoc/0124U001826), funded by the National Academy of Sciences of Ukraine.

All projects were conducted at the V. M. Glushkov Institute of Cybernetics, National Academy of Sciences of Ukraine (Kyiv, Ukraine).

Declaration on Generative AI

During the preparation of this work, the author used `gpt-oss-20b` (OpenAI's medium-sized open-weights model) solely for grammar and spelling checks.

References

[1] G. Gerganov, llama.cpp, 2025. URL: https://github.com/ggml-org/llama.cpp.
[2] Microsoft, Phi-4-mini-instruct, 2025. URL: https://huggingface.co/microsoft/Phi-4-mini-instruct.
[3] Deepseek AI, Deepseek-r1-distill-llama-8b, 2025. URL: https://huggingface.co/deepseek-ai/DeepSeek-R1-Distill-Llama-8B.
[4] Meta AI, Llama-3.2-1b-instruct, 2024. URL: https://huggingface.co/meta-llama/Llama-3.2-1B-Instruct.
[5] Qwen - Alibaba Cloud, Qwen2.5-7b-instruct, 2024. URL: https://huggingface.co/Qwen/Qwen2.5-7B-Instruct.
[6] Google DeepMind, gemma-3-12b-it, 2025. URL: https://huggingface.co/google/gemma-3-12b-it.
[7] Google DeepMind, gemma-3-1b-it, 2025. URL: https://huggingface.co/google/gemma-3-1b-it.
[8] Mistral AI SAS, Mistral-small-3.1-24b-instruct-2503, 2025. URL: https://huggingface.co/mistralai/Mistral-Small-3.1-24B-Instruct-2503.
[9] K. S. Malakhov, Medlocalgpt, 2025. URL: https://github.com/knowledge-ukraine/medlocalgpt.
[10] K. S. Malakhov, Innovative hybrid cloud solutions for physical medicine and telerehabilitation research, International Journal of Telerehabilitation 16 (2024) 1–19. doi:10.5195/ijt.2024.6635.

[11] P. Radhakrishnan, J. Chen, B. Xu, P. Ramaswami, H. Pho, A. Olmos, J. Manyika, R. V. Guha, Knowing when to ask – bridging large language models and data, arXiv (2024). doi:10.48550/ARXIV.2409.13741.

[12] O. V. Palagin, M. G. Petrenko, K. S. Malakhov, Challenges and role of ontology engineering in creating the knowledge industry: a research-related design perspective, Cybernetics and Systems Analysis 60 (2024) 633–645. URL: https://doi.org/10.1007/s10559-024-00702-6. doi:10.1007/s10559-024-00702-6.

[13] O. Palagin, V. V. Kaverinskiy, M. G. Petrenko, K. S. Malakhov, Digital health systems: Ontology-based universal dialog service for hybrid e-rehabilitation activities support, in: 2023 IEEE 12th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS), volume 1, 2023, p. 84–89. doi:10.1109/IDAACS58523.2023.10348639.

[14] M. G. Petrenko, E. Cohn, O. Shchurov, K. S. Malakhov, Ontology-driven computer systems: Elementary senses in domain knowledge processing, South African Computer Journal 35 (2023) 127–144. doi:10.18489/sacj.v35i2.17445.

[15] V. V. Kaverinsky, K. S. Malakhov, Natural language-driven dialogue systems for support in physical medicine and rehabilitation, South African Computer Journal 35 (2023) 119–126. doi:10.18489/sacj.v35i2.17444.

[16] A. Litvin, O. Palagin, V. V. Kaverinskiy, K. S. Malakhov, Ontology-driven development of dialogue systems, South African Computer Journal 35 (2023) 37–62. doi:10.18489/sacj.v35i1.1233.

[17] H. Shen, H. Chang, B. Dong, Y. Luo, H. Meng, Efficient llm inference on cpus, arXiv (2023). doi:10.48550/ARXIV.2311.00502.

[18] E. Frantar, S. Ashkboos, T. Hoefler, D. Alistarh, Gptq: Accurate post-training quantization for generative pre-trained transformers, arXiv (2022). doi:10.48550/ARXIV.2210.17323.

[19] G. Xiao, J. Lin, M. Seznec, H. Wu, J. Demouth, S. Han, Smoothquant: Accurate and efficient post-training quantization for large language models, in: Proceedings of the 40th International Conference on Machine Learning, PMLR, 2023, p. 38087–38099. URL: https://proceedings.mlr.press/v202/xiao23c.html.

[20] J. Lin, J. Tang, H. Tang, S. Yang, W.-M. Chen, W.-C. Wang, G. Xiao, X. Dang, C. Gan, S. Han, Awq: Activation-aware weight quantization for on-device llm compression and acceleration, in: Proceedings of Machine Learning and Systems, volume 6, 2024, p. 87–100. URL: https://proceedings.mlsys.org/paper_files/paper/2024/hash/42a452cbafa9dd64e9ba4aa95cc1ef21-Abstract-Conference.html.

[21] S. Kim, C. Hooper, A. Gholami, Z. Dong, X. Li, S. Shen, M. Mahoney, K. Keutzer, Squeezellm: Dense-and-sparse quantization, arXiv (2023). doi:10.48550/ARXIV.2306.07629.

[22] T. Zhang, J. Yi, B. Yao, Z. Xu, A. Shrivastava, Nomad-attention: Efficient llm inference on cpus through multiply-add-free attention, in: Advances in Neural Information Processing Systems, volume 37, 2024, p. 112706–112730. URL: https://proceedings.neurips.cc/paper_files/paper/2024/hash/ccda3c632cc8590ee60ca5ba226a4c30-Abstract-Conference.html.

[23] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. E. Gonzalez, H. Zhang, I. Stoica, Efficient memory management for large language model serving with pagedattention, in: Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles, 2023.

[24] Y. Xu, Z. Mao, X. Mo, S. Liu, I. Stoica, Pie: Pooling cpu memory for llm inference, arXiv (2024). doi:10.48550/arXiv.2411.09317.

[25] J. Stojkovic, C. Zhang, I. Goiri, J. Torrellas, E. Choukse, Dynamollm: Designing llm inference clusters for performance and energy efficiency, arXiv (2024). doi:10.48550/ARXIV.2408.00741.

[26] P. Patel, E. Choukse, C. Zhang, I. Goiri, B. Warrier, N. Mahalingam, R. Bianchini, Characterizing power management opportunities for llms in the cloud, in: Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3, ACM, La Jolla CA USA, 2024, p. 207–222. URL: https://dl.acm.org/doi/10.1145/3620666.3651329. doi:10.1145/3620666.3651329.

[27] DeepSeek AI, Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning,

arXiv (2025). doi:10.48550/arXiv.2501.12948.

[28] Meta AI, Meta-llama-3-70b, 2024. URL: https://huggingface.co/meta-llama/Meta-Llama-3-70B.

[29] M. Abdin, J. Aneja, H. Behl, S. Bubeck, R. Eldan, S. Gunasekar, M. Harrison, R. J. Hewett, M. Javaheripi, P. Kauffmann, J. R. Lee, Y. T. Lee, Y. Li, W. Liu, C. C. T. Mendes, A. Nguyen, E. Price, G. de Rosa, O. Saarikivi, A. Salim, S. Shah, X. Wang, R. Ward, Y. Wu, D. Yu, C. Zhang, Y. Zhang, Phi-4 technical report, arXiv (2024). doi:10.48550/ARXIV.2412.08905.

[30] Gemma Team, Gemma 3 technical report, arXiv (2025). doi:10.48550/ARXIV.2503.19786.

[31] Intel Corporation, Intel xeon processor e5-2695 v2, 2013. URL: https://www.intel.com/content/www/us/en/products/sku/75281/intel-xeon-processor-e52695-v2-30m-cache-2-40-ghz/specifications.html.

[32] Intel Corporation, Intel xeon platinum 8480+ processor, 2023. URL: https://www.intel.com/content/www/us/en/products/sku/231746/intel-xeon-platinum-8480-processor-105m-cache-2-00-ghz/specifications.html.

[33] Amazon Web Services, Amazon elastic compute cloud, 2025. URL: https://aws.amazon.com/ec2/.

[34] Amazon Web Services, Aws ec2 c7i-flex instances, 2025. URL: https://aws.amazon.com/ec2/instance-types/c7i/.

[35] Intel Corporation, Intel core i7-13700h processor, 2023. URL: https://www.intel.com/content/www/us/en/products/sku/232128/intel-core-i713700h-processor-24m-cache-up-to-5-00-ghz/specifications.html.

[36] K. S. Malakhov, D. Vakulenko, V. V. Kaverinskiy, Ebsco articles dataset (domain knowledge: rehabilitation medicine) + json of every article, 2023. URL: https://zenodo.org/record/8308214. doi:10.5281/ZENODO.8308214.