



Region Halland

Benchmarking Large Language Models for ICD-10 Code Generation

White Paper within the project Data-Driven Organizations / SAIL

Version: 1.0 | **Date:** 2025-12-16 | **Author:** Aixia AB and Region Halland



Aixia AB | +46 31 762 02 40



Region Halland | +46 35 13 48 00

Table of contents

1. Executive Summary	3
2. Background	3
3. Objectives and Scope	4
4. Methodology	4
5. Metrics Used	5
6. Models Evaluated	5
7. Frameworks and Runtimes	6
8. Benchmarking Results	6
8.1 <i>Runtime Framework Comparison</i>	6
8.1.1 NVIDIA NIM - Characteristics and Use Cases	8
8.1.2 Ollama - Characteristics and Use Cases	8
8.1.3 PyTorch - Characteristics and Use Cases	9
8.1.4 Framework Selection Summary	10
8.2 <i>Hardware Performance Analysis</i>	10
8.2.1 Performance Tiers: Latency Comparison	10
8.2.2 Energy Efficiency: The Real Cost Story	11
8.2.3 DGX Spark GB10: Efficiency-First Architecture	12
8.2.4 Edge Computing Boundary: Raspberry Pi	13
8.2.5 Hardware Selection Summary	13
8.3 <i>Model Size Impact</i>	14
8.3.1 Small Models (3-10B)	15
8.3.2 Mid-Size Models (10-30B): The Sweet Spot	15
8.3.3 Large Models (30B-50B)	15
8.3.4 Very Large Models (50B+)	15
8.3.5. Model Size Impact Summary	16
9. Discussion	16
9.1 <i>Interpretation of Results</i>	16
10. Conclusion	17
Appendix A – The project “Data-driven organisations – Best practices for operationalisation of AI in Sweden	19
Appendix B – Hardware specs	21
Appendix C – Model list and links	22
Appendix D - Metrics explanation for Ollama and NIM tests	23
Appendix E – Metrics used for PyTorch tests	27



1. Executive Summary

This whitepaper benchmarks large language models (LLMs) for automated ICD-10 code generation, focusing on operational feasibility, sustainability, and real-world deployment conditions rather than accuracy alone. Conducted by Aixia AB in collaboration with Region Halland, the study evaluates a wide spectrum of models across different runtimes and hardware platforms to identify practical strategies for deploying LLMs in clinical environments.

Motivation:

Accurate ICD-10 coding is vital for medical documentation, reimbursement, and healthcare statistics. Without accurate coding, billing fails, research becomes unreliable, and continuity of care suffers. Automating this process can significantly reduce administrative burden—especially in outpatient care where manual coding resources are limited. However, automation is only viable if LLM solutions can be deployed securely, efficiently, and at sustainable operational cost.

Key findings:

- Deployment is a system-level decision: The best performance comes from matching model size, hardware, and runtime.
- Mid-sized models and mid-tier GPUs offer the best balance: Mid-sized models consistently delivered optimal performance and energy efficiency, especially on mid-tier GPUs.
- Runtime choice shapes feasibility: Abstractions like Ollama and NVIDIA NIM provide automatic optimizations, resource management, and faster time-to-production.
- Infrastructure needs depend on workload: For many healthcare environments with moderate volumes and flexible latency requirements, mid-tier hardware offers an optimal balance of capability and cost. High-end GPUs provide value primarily in high-traffic or strict real-time scenarios.

Impact:

The results demonstrate that healthcare providers can deploy LLM-based ICD-10 coding far more easily than commonly assumed—often achieving operational readiness in weeks rather than months. By selecting appropriate model sizes, leveraging optimized runtimes, and aligning infrastructure with real workload requirements, organizations can achieve efficient, maintainable, and trustworthy AI inference without large datacenter investments.

2. Background

Region Halland, a Swedish regional healthcare provider, has initiated this investigation into the operational feasibility of generating ICD-10 codes using large language models (LLMs). The aim is to evaluate alternative approaches to their current solution, which is based on a fine-tuned T5 model.

ICD-10 codes (International Classification of Diseases, 10th Revision) are standardized diagnostic codes used globally in healthcare. Accurate ICD coding is critical for medical record-keeping, clinical research, health statistics, and reimbursement processes. Automating ICD-10 code generation with



LLMs can reduce clinicians' administrative burden and improve coding quality – but only if such systems can be implemented in a secure, cost-effective, and sustainable way.

Unlike traditional benchmarking initiatives, which often prioritize model accuracy or leaderboard rankings, this project focuses on the operational deployment of LLMs in real-world healthcare contexts – domains characterized by limited computational resources, strict data protection requirements, and high demands for sustainability and trust.

The goal is not to identify the most accurate model “out of the box,” but to examine which models can realistically be deployed across different hardware environments – considering resource usage, energy consumption, latency, and integration potential – while fully complying with the stringent data protection standards of healthcare.

3. Objectives and Scope

This study aimed to map out the practical feasibility of deploying LLMs for ICD-10 code generation across different compute environments – from edge devices, through consumer and mid-tier GPUs, to workstation- and enterprise-grade GPUs, and further up to high-performance systems (see Appendix B for a full list of hardware and the technical specifications). In this way, the evaluation covered both consumer- and enterprise-oriented GPU-grades.

We tested models ranging from 135 million to 70 billion parameters, as well as a multi-expert architecture with *8x22B parameters (Mixtral)*, to better understand where the breakpoints lie between model size and infrastructure demands.

Benchmarking was carried out across three primary runtime environments (Ollama, NVIDIA NIM, raw PyTorch). By covering this spectrum, we were able to evaluate not only quantitative performance metrics such as inference time and memory usage, but also qualitative aspects such as installation complexity and runtime flexibility. Importantly, the study emphasizes cost-efficiency, sustainability, and data security as key success factors – not just accuracy. While basic accuracy measurements were recorded, they were treated as a secondary parameter, mainly for curiosity and comparison.

This white paper contributes to AI Sweden's and Vinnova's broader goal: to help Swedish organizations move from AI experimentation to sustainable, secure, and production-ready deployment – by identifying best practices for MLOps and operational AI design.

4. Methodology

Inference Task Definition

Each model was tasked with generating ICD-10 codes from clinical input texts—synthetic patient journal entries supplied by Region Halland. All inputs were standardized across models and platforms to maintain consistency.

Metric Collection Approach

Inference runs were orchestrated via automated Python scripts using the requests library or pipeline from transformers library where relevant, minimizing overhead and variation from platform-specific wrappers or SDKs. These scripts recorded key metrics such as latency, memory usage, and power



draw, as outlined in the Metrics section. Logging was integrated into the test flow to ensure uniform metric collection without minimal intervention.

Environment Isolation and Fairness

All experiments were containerized using Docker to isolate environments and support reproducibility. System cache clearing and garbage collection were performed between model runs to minimize residual memory effects.

Wherever possible, models of similar architecture and size were tested across both Ollama and NIM to allow for meaningful comparison. For PyTorch-based testing, the focus was instead on smaller models from Hugging Face that could realistically run on edge devices, such as the Raspberry Pi or lower-tier GPUs. This reflects a practical concern of the study: mapping which models are viable on which classes of hardware, rather than simply comparing identical models across platforms. All tests were conducted on self-hosted infrastructure to ensure data confidentiality. The test data consisted of synthetic patient journals provided by Region Halland. To explore language-related performance impacts, these were also translated into English using a local instance of LLaMA 3.3 70B.

5. Metrics Used

To evaluate operational feasibility, we measured resource consumption, performance, and cost aspects for each model and hardware configuration. The measurements included:

- **Setup & Model Size:** Time to download and initialize models, as well as storage size, which affects startup time and memory requirements.
- **Idle Resource Usage:** Baseline load on GPU and CPU when the model is loaded but not in use, including memory usage, power consumption, and utilization.
- **Inference Performance:** Resource usage and efficiency during execution, measured as GPU/CPU load, memory usage, power consumption, response time (latency), and energy cost per 100 calls.
- **Accuracy:** Weighted comparison of generated ICD-10 codes against a synthetic reference, used primarily as a comparative value rather than a primary metric.
- **Hardware Fit & Layer Allocation:** Ability to fully fit into GPU memory and, for Ollama, distribution of the model's layers between GPU and CPU.
- **Deployment Context:** Memory requirements, hardware specifications, quantization format, and number of parameters, to place performance results in the context of infrastructure capacity.

6. Models Evaluated

A wide range of models was benchmarked across Ollama, NVIDIA NIM, and raw PyTorch to reflect different levels of deployment complexity and hardware capability (see Appendix B for a detailed list of models with links). Model sizes ranged from lightweight architectures like Smollm2 at 135M parameters to large-scale systems such as Mixtral with 8×22B parameters. Many models represented the latest generation of open-weight LLMs, including Meta's LLaMA 3.3, Google's Gemma, DeepSeek-R1, Qwen3, and Mistral variants. The selection included general-purpose models as well as domain-specific ones, such as MedLLaMA and Meditron, trained on biomedical data to



explore performance in clinical language. Mixture-of-experts architectures (e.g., Mixtral 8×7B and 8×22B) were included to assess dynamic routing trade-offs on various hardware tiers.

NIM tests were conducted using NVIDIA’s LLM-specific NIM containerized models, which are pre-packaged and production-ready via Docker—highlighting a commercial deployment route. On Ollama, models were selected from their curated library of self-hostable LLMs optimized for local inference. PyTorch tests focused on smaller models such as T5, Flan-T5, and BART, primarily to explore viability on resource-constrained edge devices. Together, the model set provided a broad operational spectrum—from edge feasibility to enterprise-scale inference.

7. Frameworks and Runtimes

To cover a wide spectrum of deployment scenarios, three different runtimes were chosen for benchmarking:

- **NVIDIA NIM (NVIDIA Inference Microservices)** – a commercial solution optimized for production, offering standardized, containerized microservices for inference, enabling stability and straightforward scaling.
- **Ollama** – a lightweight and developer-friendly platform, particularly suited for local prototyping.
- **PyTorch** – used as a baseline to provide full transparency and control in comparisons.

Together, these frameworks illustrate how runtime choices shape the balance between ease of use, scalability, and control.

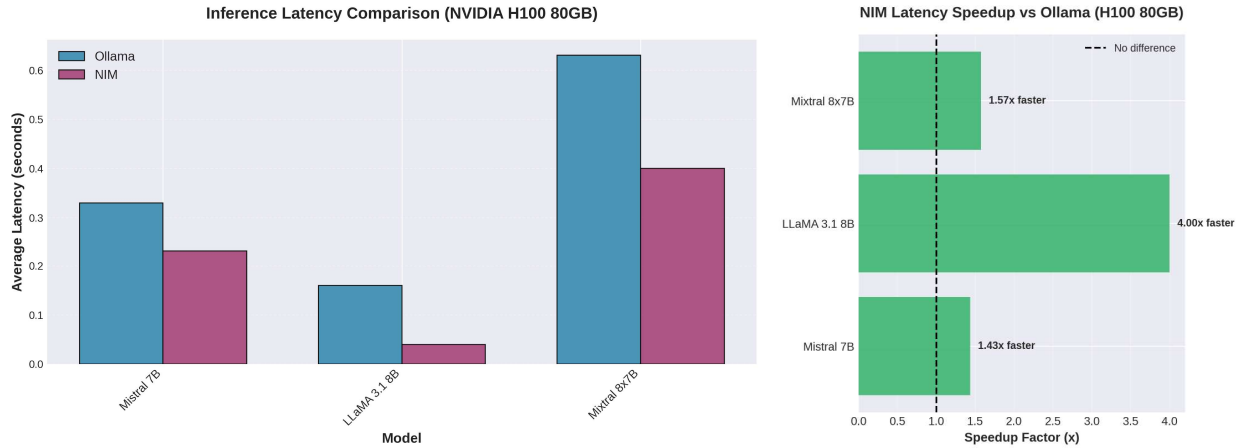
8. Benchmarking Results

The following sections present the overall results of the conducted tests.

8.1 Runtime Framework Comparison

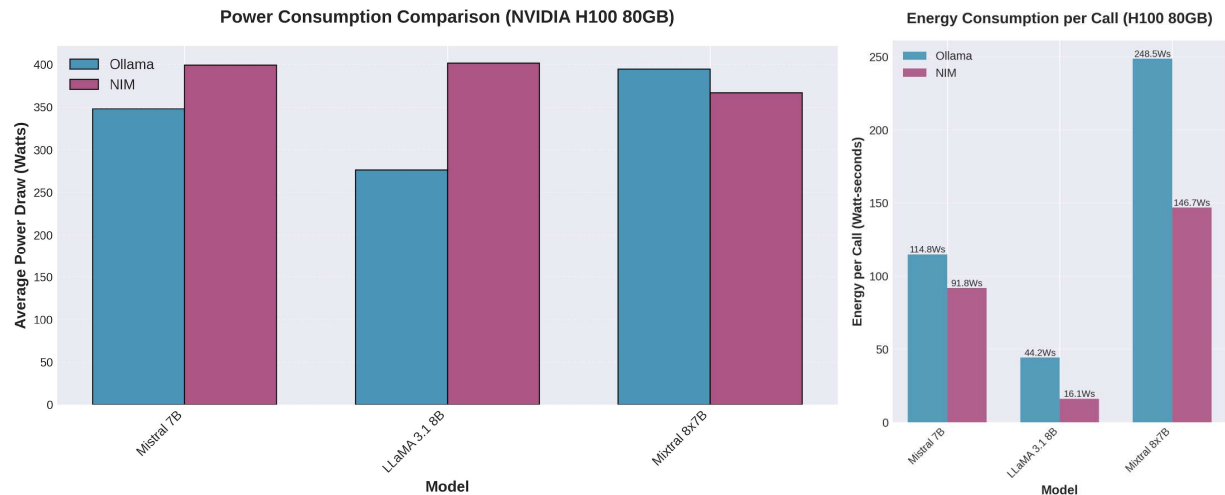
To understand how different frameworks shape the feasibility of deploying LLMs for ICD-10 coding, we first compare their runtime characteristics. This section highlights not only performance aspects like speed and stability, but also the practical trade-offs in setup complexity, resource use, and maintainability. By contrasting Ollama, NVIDIA NIM, and raw PyTorch, we show how runtime choice directly impacts both experimentation and production readiness—an essential step in moving from research to operational AI in healthcare.

Benchmarking Large Language Models for ICD-10 Code Generation



These charts illustrate the speed advantage of NVIDIA NIM's inference runtime compared to Ollama when deployed on identical hardware – in this case H100, and showing three selected models – mistral 7B, Llama 3.1 8B and Mixtral 8x7B.

Left panel shows absolute latency values for the three models across both runtimes. In all cases, NIM (purple bars) delivers faster inference than Ollama (blue bars). Right panel translates these differences into speedup factors, showing how much faster NIM performs relative to Ollama. The dashed vertical line at 1.0x represents equivalent performance—all bars extending rightward indicate NIM's advantage.



These charts reveal a critical nuance in runtime optimization. NIM's aggressive optimizations come at a cost in instantaneous power draw. For two of the three models presented here, NIM consumes more watts during active inference (as seen in the graph on the left). The power differential correlates with speedup magnitude: models where NIM achieves dramatic latency reductions (LLaMA 3.1 8B with its 4x speedup) show the highest power consumption increase. Interestingly, Mixtral 8x7B breaks this pattern. Despite NIM's 1.57x speedup, it consumes less instantaneous power than Ollama. This suggests that its model-specific container incorporates highly optimized kernels for mixture-of-experts execution. In this case, performance gains come from architectural efficiency rather than increased GPU load.



The right panel shows total energy consumed per inference call (measured in watt-seconds), which accounts for both power draw and execution time. Despite drawing more instantaneous power, NIM is significantly more energy-efficient per inference call because it completes requests so much faster. The trade-off in this case is the following: NIM sacrifices power efficiency for speed, but the speed gain compensates for it, resulting in lower total energy per request. For LLaMA 3.1 8B, this effect is dramatic: even though NIM uses 45% more power, its 4× speedup means each inference consumes 64% less total energy.

8.1.1 NVIDIA NIM - Characteristics and Use Cases

NVIDIA NIM provides pre-built, production-optimized containers for LLM inference with model-specific optimizations compiled for NVIDIA GPUs. These containers are part of the NVIDIA AI Enterprise ecosystem, meaning that long-term production use requires an enterprise license. Access to the NIM model containers is gated through the NVIDIA Enterprise Portal and authenticated via API keys, which adds an additional administrative step compared to open runtimes. While evaluation access is available, full deployment and ongoing support require an enterprise subscription, making NIM most suitable for organizations already invested in NVIDIA's enterprise stack.

Setup required some initial learning. We needed to obtain API keys through NVIDIA's enterprise portal and navigate their container registry. The key constraint is that containers only run on specific GPU models—there must exist a profile for the GPU that the container is being deployed on, otherwise the whole deployment process is likely to fail. However, once the initial setup and matching is complete, the deployment process is rather straightforward.

Runtime stability is excellent. Containers run without intervention or performance degradation. Deploying additional models was simple—just pull and launch.

Performance was strong but came with trade-offs. As shown in Section 8.1.1, NIM delivered 1.4-4× faster inference and 20-64% better energy efficiency per call. However, idle power consumption was noticeably higher—NIM keeps infrastructure "warm" for rapid response. This suggests that this runtime is best suited for high traffic scenarios, where power efficiency during inference outweighs the costs of idle power consumption.

The curated nature of NIM's model catalog imposes inherent limitations. While NVIDIA generally updates the catalogue promptly, users remain confined to the models that have been officially packaged and released, with no ability to deploy unsupported models.

Good use cases for NIM deployments are:

- High-volume workloads (1000+ calls/day)
- Real-time latency needs (<200ms)
- Standardized NVIDIA datacenter infrastructure
- Production environments prioritizing stability

8.1.2 Ollama - Characteristics and Use Cases

Ollama provides a streamlined, developer-friendly runtime for local LLM inference with automatic resource management that abstracts away many deployment complexities.



Setup was remarkably simple. We simply installed the Ollama client, specified a model name, and Ollama handled downloading, configuration, and deployment automatically. This proved invaluable during the experimental phase when we were testing multiple models rapidly.

Automatic resource management was the standout feature. Ollama dynamically handles quantization and memory allocation without manual configuration. When we ran a 70B parameter model on a laptop with insufficient VRAM (Video Random Access Memory), Ollama automatically offloaded layers to CPU rather than failing outright. The runtime intelligently partitions layers between GPU and CPU based on available memory, allowing operation even in constrained environments.

Performance was moderate but consistent. As shown in Section 8.1.1, Ollama delivered acceptable latency (0.16-0.63s range) though slower than NIM. However, instantaneous power consumption was overall lower, and the system maintained stable performance without the "warm" idle overhead of NIM.

Model availability was excellent. Any model in Ollama's library (or compatible with GGUF format) could be deployed immediately on any GPU – or even on RaspberryPi with only CPU compute. When we wanted to test newly released models or domain-specific architectures, Ollama's flexibility proved essential.

Good use cases for Ollama:

- Rapid prototyping and model experimentation
- Mixed hardware environments (consumer GPUs, laptops, varied infrastructure)
- Moderate workloads
- Teams prioritizing deployment speed over maximum optimization
- Intermittent usage patterns (lower idle power consumption)

8.1.3 PyTorch - Characteristics and Use Cases

Raw PyTorch deployment using the Hugging Face Transformers library represents baseline inference without production-serving optimizations.. We included PyTorch testing specifically to understand what is gained by using abstraction layers like Ollama and NIM.

Setup was straightforward but entirely manual. We were required to handle every aspect: model loading, tokenization, device placement, batch processing, and memory management. This provided complete visibility but placed the entire optimization burden on us.

Resource constraints were unforgiving. With limited runtime optimizations, GPU memory became a hard bottleneck. On consumer GPUs (RTX 2000 Ada, RTX 3060), only small models ran successfully. Attempting larger models like FLAN-T5-XL resulted in immediate out-of-memory failures.

Even on high-end GPUs (A10, A6000), resource utilization was poor. GPU utilization stayed below 50%, with disproportionately high CPU usage indicating preprocessing bottlenecks.

The Value of Runtime Abstraction

Framework	J/B params
PyTorch (FP16)	94.4
Ollama (quantized)	61.2



The table above shows energy efficiency normalized by model size, comparing PyTorch's raw FP16 inference against Ollama's optimized runtime on the same hardware (RTX A6000). Ollama achieved 35% better energy efficiency per parameter through quantization, optimized kernels, and intelligent memory management. The abstraction layer fundamentally improves operational efficiency, as well as deployment convenience.

Use cases for limited abstraction layers:

- Model development
- Custom inference pipelines
- Maximum transparency

8.1.4 Framework Selection Summary

Our testing revealed that runtime choice significantly impacts both operational characteristics and energy efficiency, independent of hardware selection.

Production-optimized runtimes (like NVIDIA NIM) delivered the fastest inference and best energy efficiency per call, but required specific GPU models, enterprise authentication, and maintained higher idle power consumption. Best suited for high-volume production workloads (1000+ calls/day) on standardized datacenter infrastructure where sub-200ms latency is critical.

Developer-friendly runtimes (like Ollama) provided the most accessible deployment experience with automatic resource management, broad hardware compatibility, and excellent model availability. Slower than production-optimized alternatives but adequate for most use cases (0.16-0.63s latency range), with lower idle power consumption. Ideal for prototyping, mixed hardware environments, and moderate workloads where deployment simplicity outweighs maximum optimization.

Raw framework deployment (like PyTorch with Transformers) offered complete transparency and control but required manual optimization of many inference components. Poor resource utilization and worse energy efficiency per parameter compared to optimized runtimes demonstrated the concrete value of abstraction layers. Only appropriate when ML engineering expertise is available on-site for model development and research—not recommended for production deployment without dedicated specialists.

For Region Halland's ICD-10 coding use case: We recommend starting with a developer-friendly runtime for rapid deployment and model experimentation, with potential migration to production-optimized alternatives if daily volume exceeds 500-1000 predictions or real-time clinical workflow integration demands sub-200ms response times.

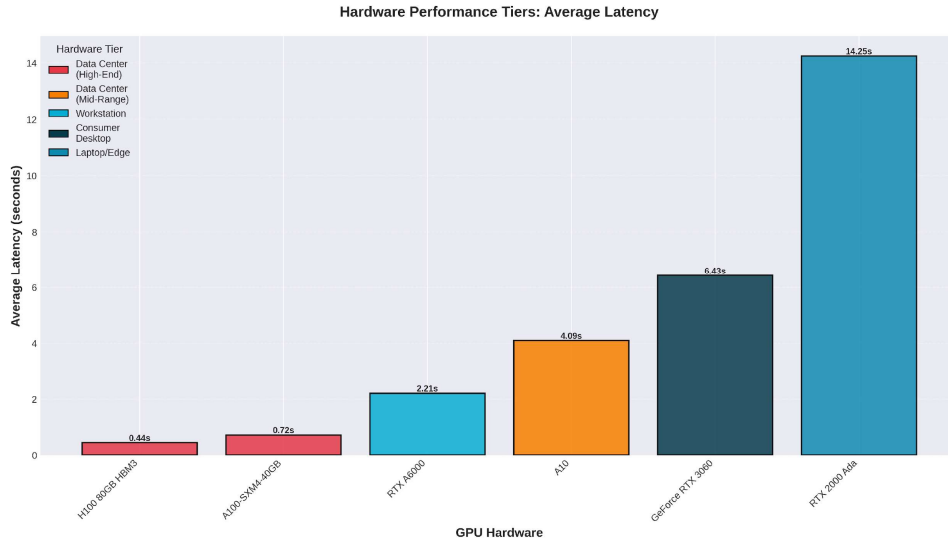
8.2 Hardware Performance Analysis

The following sections present the findings from the hardware performance perspective.

8.2.1 Performance Tiers: Latency Comparison



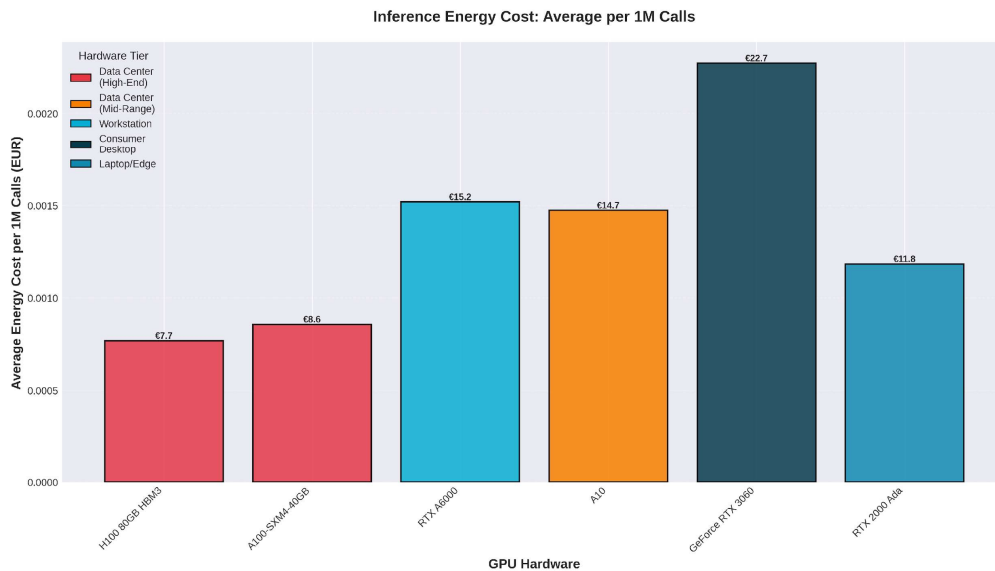
Benchmarking Large Language Models for ICD-10 Code Generation



This chart shows average inference latency across 14 LLM models (1.5B to 70B parameters) tested on six GPU tiers using Ollama. Results show expected performance scaling: high-end datacenter GPUs (H100, A100) deliver sub-second latency (0.44-0.72s), workstation and mid-range datacenter GPUs require 2-4 seconds, while consumer hardware ranges from 6-14 seconds. The 32x performance gap between the fastest (H100) and slowest (RTX 2000 Ada) hardware reflects clear tier stratification. Larger models on less powerful GPUs required automatic CPU offloading, significantly increasing latency but enabling the full test suite across all hardware.

8.2.2 Energy Efficiency: The Real Cost Story

Raw performance metrics tell only part of the story when evaluating hardware for LLM inference. While latency and throughput matter, sustainable deployment requires understanding total cost of ownership, which includes energy consumption, infrastructure requirements, and capital investment. Our analysis reveals that the inference cost is far more leveled than performance metrics alone would suggest.

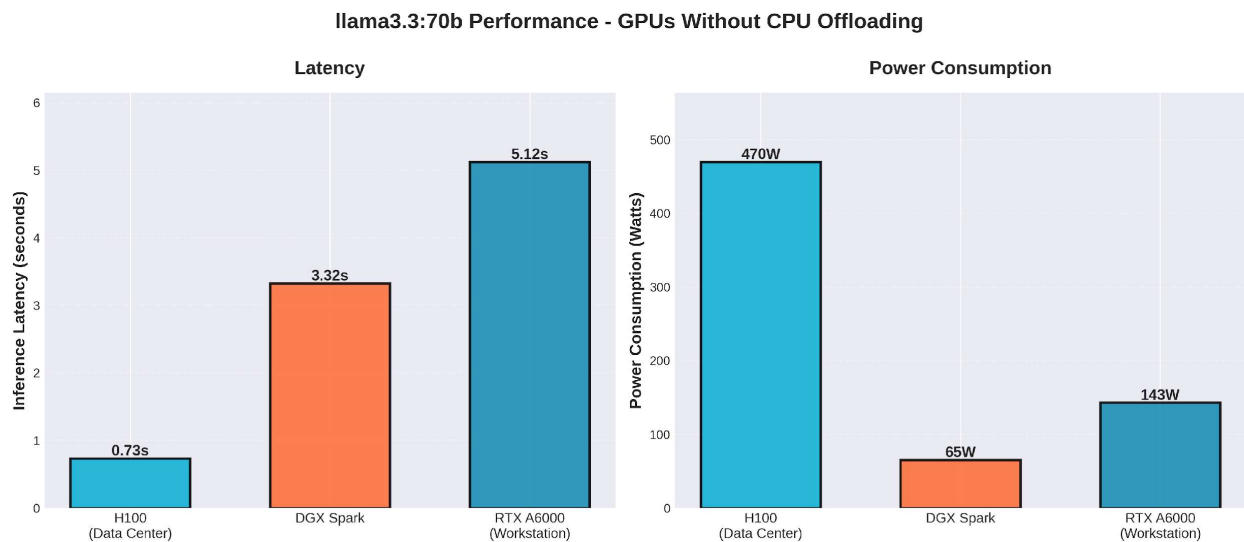


When we examine energy cost per million inference calls, the performance hierarchy in our tests compresses dramatically. High-end datacenter GPUs (H100, A100) cost €7.7-€8.6 per million calls, while workstation and mid-range alternatives (RTX A6000, A10) range from €11.8-€14.7. The RTX 2000 Ada laptop GPU costs €11.8 per million calls, and even the consumer RTX 3060 costs €22.7 per million calls. Despite the 32× performance gap shown in Section 8.3.1, the energy cost spread is modest—high-end datacenter GPUs are only 35-50% more efficient per call than workstation alternatives. This energy advantage is negligible compared to the substantial difference in hardware purchase cost.

These figures represent only energy consumed during active inference. They exclude infrastructure costs (advanced cooling, power distribution, rack space) that datacenter systems require, and critically, idle power consumption. As noted in Section 8.2, high-end GPUs maintain higher baseline power draw between inference requests.

At moderate workload volumes typical of healthcare settings, the annual energy cost difference between high-end and mid-range GPUs amounts to a few euros—negligible compared to the upfront hardware cost. When idle consumption is included, the "efficient" datacenter GPU may actually cost more in total energy for intermittent workloads. Organisations should match hardware to actual workload characteristics and utilization patterns, not theoretical peak performance.

8.2.3 DGX Spark GB10: Efficiency-First Architecture



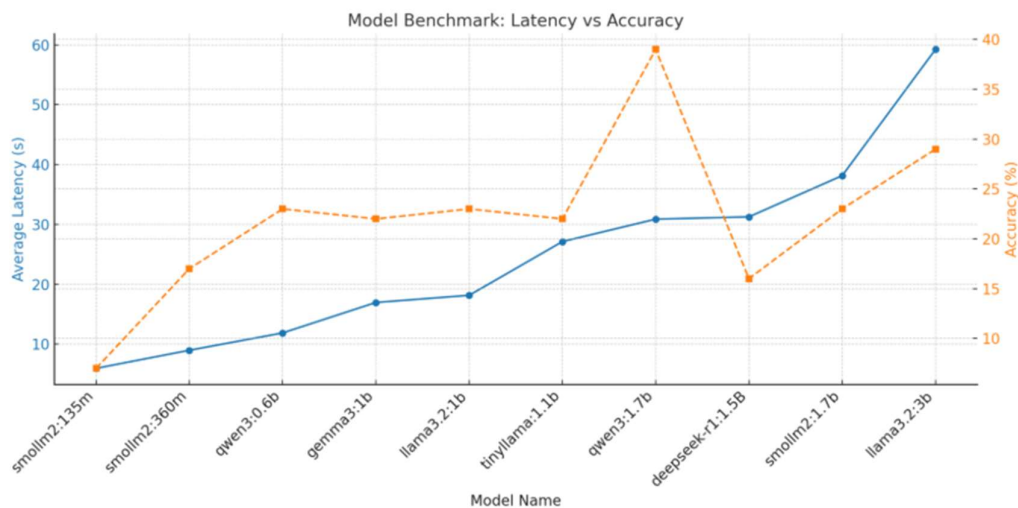
This comparison shows three systems capable of running the 70B parameter Llama 3.3 model entirely in GPU memory without CPU offloading—a demanding test that eliminates many mid-range options. The H100 delivers the fastest inference at 0.73s while drawing 470W. The RTX A6000 requires 5.12s at 143W. The DGX Spark GB10 sits between them at 3.32s latency while consuming only 65W.

The GB10's profile is notable: its 128GB unified memory allows it to run models that exceed the VRAM limits of many workstations, while consuming power closer to laptop hardware. At 65W during inference—less than half the A6000's draw and roughly one-seventh the H100's—the GB10 demonstrates that large model inference doesn't necessarily require high power consumption. The 3.32s latency, while 4.5× slower than the H100, remains within acceptable bounds for most non-real-time use cases.

This represents a fundamentally different approach to inference hardware: rather than maximizing performance at any power cost, the GB10 optimizes for sustained operation. For workloads running throughout the day, the cumulative energy difference becomes substantial—the GB10 consumes roughly the same power as a high-performance laptop while maintaining capabilities that typically require datacenter infrastructure.

8.2.4 Edge Computing Boundary: Raspberry Pi

At the opposite end of the hardware spectrum from datacenter GPUs, we tested edge devices to understand the practical boundaries of LLM deployment on severely constrained hardware.



Ollama's automatic resource management, which enabled CPU offloading on mid-range GPUs, also made it possible to run LLMs entirely without GPU acceleration on the Raspberry Pi 5, albeit with significant latency. While power consumption during inference was negligible, performance remained impractical for interactive use cases.

The significance lies not in the raw performance, but in demonstrating that LLM inference is technically feasible on edge devices with minimal power budgets. While current latency makes real-time applications impractical, this establishes a baseline for future optimization work. Techniques such as model distillation, aggressive quantization, and task-specific fine-tuning could potentially bring edge inference into viable territory for narrow use cases where power constraints outweigh latency requirements.

8.2.5 Hardware Selection Summary

Our hardware testing across six GPU tiers reveals that sustainable LLM deployment requires matching hardware to actual workload patterns rather than maximizing theoretical performance.

Performance scales predictably with hardware tier: High-end datacenter GPUs (H100, A100) deliver sub-second latency, workstation and mid-range systems require 2-4 seconds, and consumer/laptop hardware ranges from 6-14 seconds. This performance gap appears substantial, but translates to only 35-50% difference in energy cost per inference call—the performance hierarchy compresses dramatically when measured in operational costs.

The energy efficiency story challenges conventional assumptions: At moderate workload volumes typical of healthcare settings, the annual energy cost difference between high-end and mid-range GPUs amounts to a few euros—negligible compared to hardware purchase price differentials. When idle power consumption is included, "efficient" datacenter GPUs may actually cost more in total energy for intermittent workloads that leave hardware idle for significant portions of the day.

Different hardware tiers serve distinct purposes.

High-end GPUs (H100, A100) justify their investment for sustained high-volume workloads where real-time latency is critical and infrastructure is already available. Their superior performance-per-call efficiency only translates to cost savings at scale.

Efficiency-optimized systems like the DGX Spark GB10 represent an emerging category: datacenter-class memory capabilities with laptop-level power consumption (65W). For sustained moderate-volume workloads running throughout the workday, this approach delivers lower total cost of ownership than either traditional high-end or budget alternatives.

Mid-range GPUs (RTX A6000, A10) provide the best balance for most organisations: sufficient performance (2-4s latency), manageable infrastructure requirements, and energy efficiency that closely matches high-end systems at a fraction of the purchase cost.

Consumer and laptop hardware (RTX 3060, RTX 2000 Ada) remain viable for development, testing, and low-volume production where longer latency is acceptable. Their lower purchase cost and infrastructure simplicity often outweigh modest energy efficiency penalties.

Edge devices (Raspberry Pi) establish the technical boundary: LLM inference is feasible but not yet practical due to high latency. Future optimization may enable narrow use cases where power constraints dominate all other requirements.

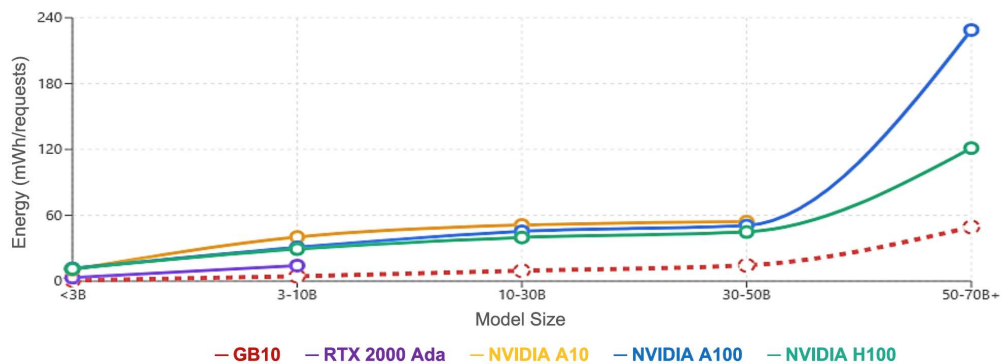
The fundamental principle is to match hardware to utilization patterns, not peak performance specifications. A moderately priced GPU running at 60% utilization delivers better economics than an expensive system running at 5% utilization, regardless of per-call efficiency metrics.

8.3 Model Size Impact

The relationship between model size and energy consumption reveals critical insights for hardware selection and deployment planning. Our testing across multiple hardware tiers and model sizes (ranging from under 1B to 70B+ parameters) demonstrates that efficiency optimization requires matching three interdependent factors: the task requirements, the model size, and the GPU capabilities.

Energy Consumption by Model Size

Energy Efficiency Trends



This chart tracks energy consumption per inference request (measured in milliwatt-hours) across five model size categories, tested on hardware ranging from DGX Spark (GB10) through laptops (RTX 2000 Ada) to datacenter systems (A10, A100, H100). Several patterns emerge that challenge conventional assumptions about GPU requirements and efficiency.

8.3.1 Small Models (3-10B)

For models in the 3-10B range, energy consumption grows gradually and steadily across all hardware tiers. This range demonstrates predictable scaling—larger models within this category consume proportionally more energy, but the relationship remains linear and manageable across different GPU classes.

8.3.2 Mid-Size Models (10-30B): The Sweet Spot

The 10-30B range shows a flatter energy consumption curve—the increase in energy per parameter slows considerably compared to the 3-10B range. This is where we begin to lose systems with insufficient VRAM, limiting hardware options, but the energy efficiency per capability unit improves. Models in this range provide sophisticated reasoning capabilities while maintaining relatively stable energy consumption.

8.3.3 Large Models (30B-50B)

The 30-50B range shows a plateau—energy consumption remains relatively flat despite increasing model size. Models in this range consume similar energy per request, but hardware options narrow significantly, with only GPUs with sufficient VRAM remaining viable.

8.3.4 Very Large Models (50B+)

Beyond 50B parameters, energy consumption spikes dramatically. High-end GPUs see energy costs jump to 120-230 mWh per request, while consumer/workstation hardware becomes insufficient to run such big models. Looking at the graph it becomes clear that more powerful GPUs (H100 vs A100) can actually be more energy-efficient for these massive models, consuming roughly half of the energy during inference. The GB10 demonstrates a new approach in hardware design, maintaining around 50 mWh through efficiency-first design, although it must be mentioned that it has significantly longer inference times compared to H100.



It must be stated that these large models are applicable only in vastly open-ended, complex tasks.

8.3.5. Model Size Impact Summary

Model selection has as much impact on operational efficiency as hardware choice, and the two decisions are interdependent.

The small to medium model size range represents the optimal balance for complex real-world applications. Energy consumption increases more slowly in this range compared to smaller or bigger models, and when combined with external knowledge retrieval, prompt optimization, or agentic approaches, these models deliver sophisticated capabilities without the resource demands of larger alternatives. This is our recommendation for applications like Region Halland's ICD-10 coding.

The fundamental lesson is to match model size to task requirements, not available hardware. For narrower applications like healthcare coding, mid-size models fine-tuned or equipped with external knowledge and domain data typically outperform larger general-purpose alternatives while consuming a fraction of the resources.

9. Discussion

It is expected that more powerful GPUs—such as the NVIDIA H100 or A100—would offer superior performance in terms of latency and accuracy. However, a key and somewhat surprising finding from this study is how well some language models performed on mid-tier or even consumer-grade GPUs such as the A10 or RTX 3060, particularly when deployed through optimized runtimes like Ollama. In several cases, even large and extra-large models were able to run—albeit with reduced speed—on hardware that would require careful configuration of quantization and memory management in PyTorch, but which Ollama handles automatically through pre-quantized models and CPU offloading. This demonstrates the power of runtime-level optimizations and quantization in enabling wider access to advanced models.

9.1 Interpretation of Results

The results from this study show that effective deployment of LLMs for tasks like ICD-10 code generation arises from a balanced interplay between three components: the runtime, the hardware, and the model architecture. These three factors form a single operational system, and their interactions determine real-world performance far more than any one of them in isolation.

A key finding is that efficient architectures and optimized runtimes can outweigh raw computational muscle. In several cases, mid-tier GPUs combined with quantized models or optimized containers delivered performance that challenged high-end hardware in terms of inference power efficiency. This means that before choosing any model or hardware stack, organisations must begin by asking how the solution will actually be used.

Key questions include:

Workload pattern: Will the system handle high traffic with continuous load, or will inference occur sporadically throughout the day?

Latency requirements: Is a response time of several seconds acceptable, or is sub-second latency essential for workflow integration?

Parallelism: Do we expect many simultaneous requests, or mostly sequential single-user traffic?



Existing infrastructure: Do we already operate high-end GPUs? If so, the cost of idle power is already sunk, and running inference on those machines becomes cost-effective. If not, the total cost of acquiring and maintaining additional GPUs becomes a central consideration.

Once these operational requirements are clarified, the results show that mid-tier GPUs (such as the NVIDIA A10 or A6000) often provide the best balance of latency, energy efficiency, and infrastructural simplicity. They avoid the cooling demands, footprint, and administrative overhead associated with high-end systems, while still enabling mid-sized LLMs (10–30B) to run with excellent performance. For new deployments—especially in healthcare environments where inference loads are moderate—this class of hardware represents a compelling sweet spot.

At the same time, the study confirms that very large models do benefit from more powerful GPUs, not because they always need extreme compute, but because they need memory. When models exceed the VRAM capacity of mid-tier cards, runtimes must fall back on CPU offloading, which has a dramatic impact on latency and total energy consumed. For organisations intending to serve large general-purpose models or to operate at scale, high-end GPUs remain the more efficient long-term choice.

Overall, operational cost is revealed to be more nuanced than the purchase price of hardware. Although GPU specifications outline theoretical power draw and performance ceilings, actual energy consumption depends strongly on how models are executed. Decisions about runtime optimizations, quantization formats, and model size have measurable effects on efficiency. Our benchmarks show that organisations can significantly influence consumption by choosing appropriate runtimes and models, rather than relying solely on hardware capabilities.

The tests also demonstrated the value of runtime-level abstractions. Optimized systems such as NVIDIA NIM extract maximum performance from compatible GPUs through model-specific kernels, while developer-friendly tools like Ollama provide intelligent resource management and automatic offloading. These abstractions make inference more stable, predictable, and maintainable—especially in environments where dedicated ML engineering expertise may be limited.

Finally, the results highlight the practical advantages of using widely supported, standardized tools and model formats. Throughout the benchmarking process, the ability to pull models directly from curated repositories and run them with minimal configuration enabled rapid experimentation across dozens of architectures and sizes. Choosing niche or unconventional frameworks would require significant in-house expertise for deployment, optimization, and long-term support. In contrast, staying within well-supported ecosystems enables organisations to leverage pre-optimized solutions, benefit from community-driven updates, and shorten the path from experimentation to production. In summary, the results show that successful LLM deployment depends on aligning the runtime, hardware, and model with the real operational needs of the environment. Efficient architectures, sensible hardware choices, and mature runtimes can deliver strong performance at sustainable cost—an important insight for healthcare organisations seeking to balance capability with practicality.

10. Conclusion

This study shows that effective LLM deployment for ICD-10 coding depends on choosing the right combination of runtime, hardware, and model size, rather than relying on raw GPU power alone.



Benchmarking Large Language Models for ICD-10 Code Generation

Optimized runtimes and efficient architectures allow efficient performance even on mid-tier hardware, while high-end GPUs become most valuable only in high-volume or low-latency scenarios.

The findings highlight that organisations should start with their actual workload and latency requirements, then select models and infrastructure that match those needs. For many healthcare settings, mid-sized models on mid-range GPUs offer an excellent balance of speed, energy efficiency, and maintainability.

Using standardized, well-supported runtimes and model formats further reduces deployment complexity and shortens the time from experimentation to production. With the right design choices, sustainable and cost-efficient LLM inference is within reach for healthcare providers, enabling Region Halland and similar organisations to move confidently toward practical, production-ready AI solutions.



Appendix A – The project “Data-driven organisations – Best practices for operationalisation of AI in Sweden

This material has been produced as part of the Vinnova-funded project Data-driven organisations – Best practices for operationalisation of AI in Sweden (DDO), a project lasting just under two years with twenty participants from private sector, public sector, and academia. Together, they tackled issues concerning large- and small-scale operation of AI solutions and how to enable and use AI broadly across an organisation.

The work focused on three specific use cases: Local sustainable operation of AI, legal and technical prerequisites for effective infrastructure, and how to create the best conditions for keeping thousands of AI models in operation.

A compilation of all material produced within the framework of DDO is available on AI Sweden's website – <https://www.ai.se/en/project/data-driven-organizations-best-practices-ai-operationalization-sweden>.

The organisations that participated in DDO were:

- Aixia <https://aixia.se>
- Hewlett Packard Enterprise <https://www.hpe.com>
- Hopsworks <https://www.hopsworks.ai>
- IBM <https://www.ibm.com>
- Linköpings Universitet <https://liu.se>
- NetApp <https://www.netapp.com>
- Predli <https://www.predli.com>
- Proact <https://www.proact.se>
- RISE <https://www.ri.se>
- RedHat <https://www.redhat.com>
- Region Halland <https://www.regionhalland.se>
- Sahlgrenska University Hospital <https://www.sahlgrenska.se>
- Statistics Sweden (Statistiska Centralbyrån) <https://www.scb.se>
- The Swedish Tax Agency (Skatteverket) <https://www.skatteverket.se>
- Stormgrid <https://www.stormgrid.ai>
- The Swedish Transport Administration (Trafikverket) <https://www.trafikverket.se>
- Volvo Parts <https://www.volvogroup.com>
- Region Västra Götaland <https://www.vgregion.se>
- Santa Anna <https://www.santa-anna.se>
- AI Sweden <https://www.ai.se/en>

The project was funded by the participating organisations and Vinnova. AI Sweden is in part financed by the EU.



SAIL (Sustainable AI Infrastructure Lifecycle)

This white paper is produced within the SAIL use case, which is part of the DDO project. SAIL focuses on building cost-efficient, environmentally sustainable, and operationally effective infrastructure that supports the entire AI lifecycle – from exploration and development to training, deployment, inference, and long-term operations.

The goal is to enable organizations to adopt and scale AI sustainably by reducing costs, minimizing environmental impact, and ensuring that AI systems can operate and evolve over time without unnecessary complexity.

Key areas of focus include:

- Designing scalable and flexible AI infrastructure that grows with organizational needs
- Exploring hardware, cloud, and modular/shared platform options
- Creating practical guidelines for long-term AI operations
- Optimizing resources and reducing technical and administrative overhead

The outcome will be a validated model and actionable recommendations that help organizations of all sizes and maturity levels build, operate, and evolve AI solutions in a sustainable, efficient, and future-proof way.

In addition to this white paper, SAIL has also produced the following white papers:

- **MLOps on-prem without Kubernetes – A Faster Path to Production:** Demonstrates how an efficient on-prem MLOps pipeline can be implemented without Kubernetes, emphasizing simplicity, reproducibility, and rapid deployment to production.
- **The AI Implementation Spectrum – Strategies for Sustainable and Scalable Adoption:** Introduces a framework for understanding different maturity levels of AI implementation and how organisations can build scalable and sustainable strategies across the full AI lifecycle.
- **Sustainable Image Inference in Practice:** Investigates which hardware platforms deliver the best balance of performance, energy efficiency, and cost for running AI inference on radiology images in an on-premises healthcare environment.

Appendix B – Hardware specs

System	RTX 2000 Ada Laptop	RTX 3060	A10	A600	A100-SXM4-40GB	H100 80GB HBM3	GB10 (DGX Spark)
OS	Ubuntu 24.04.2 LTS	Debian GNU/Linux trixie/sid	Debian GNU/Linux trixie/sid	Debian GNU/Linux trixie/sid	Ubuntu-based NVIDIA DGX OS	Ubuntu-based NVIDIA DGX OS	Ubuntu 24.04.3 LTS
Kernel	6.14.0-24-generic	6.12.12-amd64	6.12.12-amd64	6.12.12-amd64	5.15.0-1053-nvidia	5.15.0-1053-nvidia	6.11.0-1016-nvidia
CPU	Intel Core Ultra 9 185H	AMD Ryzen Threadripper PRO 3955WX	AMD Ryzen Threadripper PRO 3955WX	AMD Ryzen Threadripper PRO 3955WX	AMD EPYC 7742	Intel Xeon Platinum 8480CL	Cortex-X925 Cortex-A725
Cores/Threads	16 cores / 22 threads (2 threads per core)	16 cores / 32 threads (2 threads per core)	16 cores / 32 threads (2 threads per core)	16 cores / 32 threads (2 threads per core)	128 cores / 256 threads (2 threads per core)	112 cores / 224 threads (2 threads per core)	20 cores / 20 threads
Sockets	1	1	1	1	2	2	1
RAM	62 GiB	62 GiB	62 GiB	62 GiB	1.0 TiB	2.0 TiB	119 GiB
GPU	NVIDIA RTX 2000 Ada Gen Laptop	NVIDIA GeForce RTX 3060	NVIDIA A10	NVIDIA RTX A6000	NVIDIA A100-SXM4-40GB	NVIDIA H100 80GB HBM3	NVIDIA GB10
VRAM	8 GiB	12 GiB	23 GiB	49 GiB	40 GiB	80 GiB	119GB
Driver	550.163.01	535.216.03	535.216.03	535.216.03	550.54.15	535.161.08	580.95.05

Table 1: Hardware specs

Appendix C – Model list and links

Framework	Model	Link
Ollama	llama3.3:70b	https://ollama.com/library/llama3.3
Ollama	deepseek-r1:70b	https://ollama.com/library/deepseek-r1
Ollama	deepseek-r1:32b	https://ollama.com/library/deepseek-r1
Ollama	deepseek-r1:8b	https://ollama.com/library/deepseek-r1
Ollama	deepseek-r1:1.5B	https://ollama.com/library/deepseek-r1
Ollama	gemma3:27B	https://ollama.com/library/gemma3
Ollama	gemma3:4B	https://ollama.com/library/gemma3
Ollama	qwen3:30b	https://ollama.com/library/qwen3
Ollama	gemma3:27b-it-qat	https://ollama.com/library/gemma3
Ollama	llama3.1:8b	https://ollama.com/library/llama3.1
Ollama	meditron:7b	https://ollama.com/library/meditron
Ollama	meditron:70b	https://ollama.com/library/meditron
Ollama	medllama2:7b	https://ollama.com/library/medllama2
Ollama	mistral:7b	https://ollama.com/library/mistral
Ollama	mixtral:8x7b	https://ollama.com/library/mixtral
Ollama	mixtral:8x22b	https://ollama.com/library/mixtral
Ollama	phi3:3.8b	https://ollama.com/library/phi3
Ollama	mistral-nemo:12b	https://ollama.com/library/mistral-nemo
Ollama	tinyllama:1.1b	https://ollama.com/library/tinyllama
Ollama	gemma3:1b	https://ollama.com/library/gemma3
Ollama	qwen3:0.6b	https://ollama.com/library/qwen3
Ollama	qwen3:1.7b	https://ollama.com/library/qwen3
Ollama	llama3.2:1b	https://ollama.com/library/llama3.2
Ollama	smollm2:1.7b	https://ollama.com/library/smollm2
Ollama	smollm2:360m	https://ollama.com/library/smollm2
Ollama	smollm2:135m	https://ollama.com/library/smollm2
Ollama	llama3.2:3b	https://ollama.com/library/llama3.2
NIM	General Repository	https://docs.nvidia.com/nim/large-language-models/latest/supported-models.html
NIM	Deepseek-r1-32b	https://catalog.ngc.nvidia.com/orgs/nim/teams/deepseek-ai/containers/deepseek-r1-distill-qwen-32b
NIM	llama-3.1-8b-instruct	https://catalog.ngc.nvidia.com/orgs/nim/teams/meta/containers/llama-3.1-8b-instruct
NIM	mixtral-8x7b-instruct-v0-1	https://catalog.ngc.nvidia.com/orgs/nim/teams/mistralai/containers/mixtral-8x7b-instruct-v01
NIM	Mixtral-8x22B-Instruct-v0.1	https://catalog.ngc.nvidia.com/orgs/nim/teams/mistralai/containers/mixtral-8x22b-instruct-v01
NIM	Mistral-7B-Instruct-v0.3	https://catalog.ngc.nvidia.com/orgs/nim/teams/mistralai/containers/mistral-7b-instruct-v0.3
NIM	Mistral-Nemo-12B-Instruct	https://catalog.ngc.nvidia.com/orgs/nim/teams/nv-mistralai/containers/mistral-nemo-12b-instruct
NIM	Llama-3.3-70b-Instruct	https://catalog.ngc.nvidia.com/orgs/nim/teams/meta/containers/llama-3.3-70b-instruct
NIM	Deepseek-R1-Distill-Llama-8B	https://catalog.ngc.nvidia.com/orgs/nim/teams/deepseek-ai/containers/deepseek-r1-distill-llama-8b
PyTorch	google-t5/t5-base	https://huggingface.co/google-t5/t5-base
PyTorch	google/flan-t5-base	https://huggingface.co/google/flan-t5-base
PyTorch	google/flan-t5-xl	https://huggingface.co/google/flan-t5-xl
PyTorch	facebook/bart-base	https://huggingface.co/facebook/bart-base
PyTorch	facebook/bart-large	https://huggingface.co/facebook/bart-large

Table 2: Frameworks and models used in tests

Appendix D - Metrics explanation for Ollama and NIM tests

Pulling time (seconds)

This metric captures the time required to prepare a model for use. For Ollama, it specifically measures the duration of downloading the model. For NVIDIA NIM, it includes the time taken to deploy the container, pull the model, and complete setup until the model is ready to serve requests. While influenced by external factors such as network speed and container startup behavior, this metric provides a general sense of setup latency — included here primarily for illustrative comparison.

Size (GB)

This metric reflects the storage footprint of each model in gigabytes. For NVIDIA NIM, it refers to the compressed model size as published on the official model repository. For models run via Ollama, size was programmatically retrieved using the /api/tags endpoint, and converted from bytes to gigabytes. While not directly indicative of performance, model size influences download time, disk usage, and memory requirements, making it a relevant factor in deployment planning.

Idle Resource Usage

This group of metrics captures the baseline hardware consumption when the model is loaded but not actively serving inference requests. It includes:

- **Idle Memory (MiB):** GPU memory occupied in an idle state.
- **Idle GPU Utilization (%):** Percentage of GPU compute resources used while idle.
- **Idle Power (W):** Power consumption of the GPU during idle, measured in watts.
- **Idle CPU Usage (%):** Combined percentage of user and system CPU time during idle, averaged over a 1-second interval.

For Ollama and custom setups, GPU metrics were collected using NVIDIA's NVML Python API (pynvml), and CPU usage was measured with psutil.cpu_times_percent.

For NVIDIA NIM, GPU data was gathered using the nvidia-smi CLI with:

```
--query-gpu=index,name,memory.used,memory.free,memory.total,utilization.gpu,power.draw.
```

These metrics provide a snapshot of the static overhead introduced by different runtime environments and help evaluate their baseline efficiency — important for both resource planning and sustainability assessments in real-world deployments.

Runtime Performance Metrics

These metrics were collected during live inference using multithreaded sampling at high frequency (every 50 ms) across GPU and CPU resources. They provide a detailed view of system behavior under actual workload conditions:

- **Peak GPU Memory (MiB):** The highest amount of GPU memory used during a single inference. Measured via pynvml.nvmlDeviceGetMemoryInfo. This reflects the temporary footprint of model execution and is crucial for memory-constrained environments.
- **Average Power Draw (W):** The average wattage consumed by the GPU during inference, obtained via pynvml.nvmlDeviceGetPowerUsage. This is a key indicator of energy efficiency and operational sustainability.



- **Average GPU Utilization (%):** The mean percentage of GPU compute utilization during each inference, measured via `pynvml.nvmlDeviceGetUtilizationRates`. This metric gives insight into how well the model workload saturates available GPU compute.
- **Average CPU Utilization (%):** Combined system + user CPU load during inference, collected with `psutil.cpu_percent`. This helps assess the additional load imposed on host CPUs, which is particularly relevant for multi-user or server-based deployments.
- **Average CPU Memory Usage (MiB):** The mean amount of RAM used system-wide during inference, gathered with `psutil.virtual_memory().used`. Indicates overall pressure on system memory during operation.
- **Average Latency (seconds):** The time taken to generate ICD-10 predictions for a single patient journal, measured using `time.perf_counter` before and after the inference call. Latency is a critical factor for real-time or near-real-time applications in clinical contexts.

Accuracy Metric

Accuracy was measured by comparing each model's predicted ICD-10 codes against a synthetic ground truth. This ground truth was created by running the largest models multiple times on each journal entry and extracting the top 5 most frequently occurring ICD-10 codes. Each model's output was scored using a weighted matching system:

- Exact code match: 1.0
- Match on the first three characters: 0.75
- Match on the chapter letter (first character): 0.25
- No match: 0.0

The highest score from these comparisons was retained for each inference.

Note: All models were evaluated in their default state without fine-tuning. The purpose of the accuracy metric is not to validate clinical readiness but to offer a baseline comparison of out-of-the-box performance across model types and runtimes.

Additional note: To explore the impact of language and further probe model behavior, an extra evaluation run was performed using English translations of the same Swedish patient journals. This run had no specific application goal but contributed to a broader understanding of language sensitivity and generalization in ICD-10 code generation.

Model Layer Allocation (Ollama-specific)

For models executed using Ollama, three additional metrics were analyzed to describe how the model's internal layers were distributed across hardware:

- **Total Layers:** The total number of transformer layers in the model architecture. This provides a structural baseline for comparing offloading behavior.
- **GPU Layers:** The number of layers loaded onto the GPU during inference. This reflects the portion of the model that benefits from accelerated compute and lower latency.



- **CPU Layers:** The number of layers offloaded to the CPU. This occurs when the model is too large to fully fit on the GPU or when Ollama dynamically balances resource use.

Ollama includes an automatic offloading mechanism that partitions model layers between GPU and CPU based on available memory and compute capacity. This allows larger models to run even on constrained hardware but introduces a trade-off in inference speed and power consumption. Tracking this layer distribution enables a detailed understanding of model performance and hardware utilization under realistic conditions.

Fits on GPU (Boolean)

This binary metric indicates whether a model fits entirely into the available GPU memory during inference. A model is considered to "fit" if either: The number of GPU layers equals the total number of model layers, or The number of CPU layers is zero.

This condition reflects a successful full deployment of the model on GPU without offloading, which typically results in faster inference and more efficient resource utilization. Models that do not fully fit may require CPU fallback or hybrid strategies, leading to increased latency and power draw. This metric helps assess the suitability of a given model-hardware pairing for production environments.

Deployment Log Metrics

Several metrics were extracted from Ollama and NVIDIA NIM deployment logs to capture system-level context and resource usage at model load time. These help explain why some models fit on specific GPUs and how runtime behavior varies across environments.

VRAM Requirements

- `vram_available_gb` – GPU memory available at deployment.
- `vram_required_full_gb` – Estimated memory to fully load model on GPU.
- `vram_required_partial_gb` – Estimated memory needed for partial (offloaded) loading.

These are pre-deployment estimates and help predict whether offloading will occur.

Memory Buffers

- `gpu_model_memory_gb` – Actual GPU memory allocated for model weights.
- `cpu_model_memory_gb` – Memory allocated on CPU for offloaded components.

Reflect real post-deployment memory use, not just projections.

Hardware Info

- `gpu_name`, `gpu_vram_total_gb`, `gpu_vram_free_gb`
- `cpu_ram_total_gb`, `cpu_ram_free_gb`

Useful for matching deployment outcomes with hardware limits.

Model Format & Size

- `quantization` – Format/compression used (e.g., ggml q4), in some cases expressed precision, eg F16.
- `model_params_billions` – Total number of parameters (in billions).

These explain differences in resource usage and performance between models with similar architecture.

Energy Cost per 100 Calls (SEK)



This metric estimates the total electricity cost (in euros) to generate 100 ICD-10 predictions using a given model and hardware setup. It is calculated based on: the average power draw across all active GPUs during inference, the latency per call, and electricity price per kilowatt-hour (in this case, 0.386kr/kWh), which was the average electricity price for 2024 in Sweden (source: <https://www.elbruk.se/elpris-historik-2024>).

Hardware Price (EUR)

Estimated based on MSRP-equivalent prices for each GPU, converted to EUR. For multi-GPU systems (e.g., A100 ×2, H100 ×4), prices are aggregated.



Appendix E – Metrics used for PyTorch tests

Metric	What it Means	How It's Measured in t5_tests.py
model name	The Hugging Face model identifier being tested (e.g., google/flan-t5-base, facebook/bart-large)	Defined in the test loop; passed to Hugging Face's pipeline() function.
gpu name	The name of the GPU used during testing (e.g., NVIDIA_A10, RTX_A6000)	Captured using pynvml.nvmlDeviceGetName() and cleaned (replace(" ", "_")).
average peak memory MiB	The maximum GPU memory usage per inference call, averaged across all test samples	Sampled using pynvml.nvmlDeviceGetMemoryInfo() in a fast loop every 0.05s. Each call's peak memory is recorded, and the average is taken.
average power draw W	The average power consumption of the GPU (in watts) during inference	Sampled every 0.05s using pynvml.nvmlDeviceGetPowerUsage(), averaged over all calls.
average gpu util %	Average percentage of time the GPU was busy during inference	Measured with pynvml.nvmlDeviceGetUtilizationRates().gpu, averaged across samples per call.
average cpu util %	Average CPU usage percentage across all inference calls	Captured with psutil.cpu_percent() in a 0.05s sampling thread during inference.
average cpu memory MiB	Average RAM usage (in MiB) on the CPU side during model inference	Sampled with psutil.virtual_memory().used in the same loop, converted to MiB.
average latency seconds	Time taken to process each inference call, averaged	Measured using time.perf_counter() before and after the model call. Total duration is averaged.
accuracy %	How many outputs matched the expected ICD-10 codes	For each response, check_accuracy() compares predicted vs ground truth codes using string matching and partial matches (score: 1.0, 0.75, 0.25). Results are summed and averaged as a percentage.

Table 3: Detailed explanation of the metrics used

Project Context and Contributors

This white paper was developed as part of the Sustainable AI Infrastructure Lifecycle (SAIL) use case within the national project Data-Driven Organizations (DDO), coordinated by AI Sweden.

The purpose of the SAIL use case is to explore financially and environmentally sustainable approaches to AI infrastructure that support the entire AI lifecycle — from research and development to deployment, inference, and long-term operation.

Project Partners

The work has been carried out in collaboration between: Aixia AB, Region Halland, and AI Sweden, with additional insights shared through the broader DDO consortium including industry, academia, and public-sector partners.

Authors

Milena Miernik, Aixia AB

Use Case SAIL

Aixia AB: Cecilia Millheim, Ellen Reinhardt, Jonas Nordin, Klas Ludvigsson, Milena Miernik, Olof Sandell, Simon Janeck
Region Halland: Georgios Bramis, Karin Westerberg, Lina Gårdemark, Stefan Bäckström, Torbjörn Olander

Aixia AB
Hälsingegatan 10
414 63 Göteborg

www.aixia.se

Region Halland
Box 517
301 80 Halmstad

www.regionhalland.se

