Practical Framework for Estimating Cost for AI Projects Beyond the Cloud

Adnan Bayoun

August 28, 2025

Abstract

This paper introduces a preliminary framework for modeling per-inference costs of *LLMs* on on-premise hardware. Motivated by the growing need for cost transparency which can be used to forecast more accurate ROI in enterprise AI deployments (especially pre-implementation), where cloud-based deployments are not suitable (for security or privacy purposes), the proposed framework considers compute and energy as core modules; however, proposes further modules like latency and memory depending on the use case.

While not derived from formal benchmarking, the framework uses a conceptual approach and is intended as a starting point for discussion and refinement, rather than a definitive or academically rigorous analysis. Readers are invited to critique, validate, and extend this approach to support more robust cost modeling practices.

1 Introduction

While exploring methods to design multisession agent orchestration workflows and build reliable yet cost-effective hardware infrastructure, I came across a paper that examined cost factors in agentic system; specifically focusing on routing agents based on two key metrics: (1) query complexity and the model's ability to provide confident results, and (2) cost; for more information, see[2].

The paper's equations were intended to guide model selection from a pool of compatible models for specific use cases.

Although agent routing is an important challenge, most enterprise scenarios still hugely benefit from using sophisticated *intent managers*, which offer greater control over agentic systems.

Nevertheless, these equations provided valuable insights, through trial and error, into a potential framework for estimating inference costs and making informed decisions.

Businesses work in cases where limited information meets rapid requirements for implementation, this is meant to provide a starting point where quick but accurate estimates can be made, that help empower decision makers with more data, thereby assisting in decision making.

2 Inference Costing Framework

We define the total per-inference cost of running model M_k on-premise as the sum of two core components:

$$\boxed{C(M_k) = C_{\text{compute}}(M_k) + C_{\text{energy}}(M_k)}$$

Each term corresponds to a distinct dimension of hardware and workload related cost. The framework is modular, which means individual components can be substituted with more accurate formulations as empirical data becomes available.

$C_{\text{compute}}(M_k) \rightarrow$ aims to capture the raw processing effort required to execute floating point operations on the chosen hardware.

$C_{\text{energy}}(M_k) \rightarrow$ reflects the electricity required to sustain inference, including both active compute power and idle consumption.

—

I initially identified 2 more cost related formulas from the original paper that I thought could have a worthwhile impact for practical use:

$C_{\text{memory}}(M_k) \rightarrow$ accounts for the overhead of storing and retrieving model parameters, caches, and embeddings

While expanding it out theoretically, it wasn't very practical to split out the memory TCO from the overall HW/Compute TCO due to the nature of GPUs. I didn't think the practical business benefits yielded enough value to warrant the extra complexity; for what most use cases wouldn't need to begin with.

However, the below are cases where it might be worthwhile exploring:

1. You want to allocate a portion of the cost of the compute to the memory - attributing it separately for extra granularity. Then you will need to:
   (a) Allocate a portion of the TCO to memory
   (b) Reduce the compute price per FLOP accordingly so the two would end up summing back up to the same value
2. A more warranted situation is if you're doing price comparisons on two otherwise identical pieces of HW with different VRAM. This way you would set up the cost

equation to something like:

$$\kappa_{mem} = \frac{C_{(hiVRAM)} - C_{(loVRAM)}}{(C_{cap,hi} - C_{cap,lo}) \cdot T}$$

This thereby allows you more granular understanding of your compute components.

3. You can also use the memory to attribute VRAM differences in different GPUs, calculating the potential improvements and returns when upgrading your hardware (like an upgrade between H100 and H200; when the upgrade would be from 80 GB to 141 GB respectively)

$C_{\text{latency}}(M_k) \rightarrow$ aims to capture the $TTFT$ in relation to business expectations, compute, prompt complexity, and model size

After performing a few examples, it was clear to me that there wasn't as much value to most users as initially anticipated, and thought it would be better added as an additional module rather than part of the core framework.

Latency is critical in some domains (especially SLA driven services), but for most organizations' use cases its better treated as a secondary cost driver since it mostly manifests indirectly through productivity and scalability. Therefore, I exclude it from the core framework but provide a dedicated "Latency Module" appendix for organizations where TTFT based penalties are quite impactful.

2.1 Compute Cost $C_{compute}(M_k)$: Rooted in Hardware FLOPs

Compute cost is modeled around available floating point operations per second vs consumed per inference / query in relation to TCO of on-premise hardware.

$$\boxed{C_{\text{compute}}(M_k) = \beta_k \cdot \text{AvgFLOPs}(M_k)}$$

Where:

- $\beta_k \rightarrow$ computational efficiency factor
- $\text{AvgFLOPs}(M_k) \rightarrow$ floating point operations for the specific model in question on average

2.1.1 β_k

This is the computational efficiency factor, it represents the effective cost per FLOP delivered by the hardware k over its usable lifetime, which normalizes the TCO through; maximum theoretical throughput, utilization, and asset lifetime.

$$\beta_k = \frac{C_{\text{HW}}}{F_{\text{max}} \cdot u \cdot T}$$

Where:

- $C_{HW} \rightarrow$ TCO of the hardware
- $F_{max} \rightarrow$ the maximum FLOPs/second the hardware can deliver (maximum theoretical output)
- $u \rightarrow$ utilization rate of the hardware
- $T \rightarrow$ total operational time in seconds of the hardwares life / or the hardwares depreciation schedule

2.1.2 $AvgFLOPs(M_k)$

FLOPs increase with prompt complexity and size, and decrease the simpler it gets. This framework aims to take an average of all complexity levels expected from the use case this model is serving.

Note: *It is possible to define the compute cost per inference for each separate complexity level, and depending on the use case, this might yield extra value.Where the equations would be targeted to each intent / task type rather than an average.*

$$AvgFLOPs(M_k) = \frac{\sum_{i=1}^{n} FLOPs_i}{N}$$

Where the simplest form to calculate FLOPs per inference is:

$$FLOPs \approx 2 \times N_{params} \cdot (S_{in} + S_{out})$$

And:

- $N_{params} \rightarrow$ number of model parameters
- $S_{in} \rightarrow$ number of input tokens
- $S_{out} \rightarrow$ number of output tokens

2.2 Energy Cost $C_{energy}(M_k)$

Estimating energy cost requirements can be quite convoluted. I won't pretend to be a data center engineer, there are internationally accepted standards for DC planning; which includes energy requirements, cooling requirements, overheads, and the works. Instead, we will take a page from their book and outline how to look at energy related costs for AI workloads on premise.

In this section and throughout the paper, I've focused more on practicality of use over ultra accurate cost calculations, the more accurate we want to be, the larger our reliance is on empirical data and experimentation; both of which are considered luxuries in real world scenarios. Therefore, I have tried building up the cost model below on publicly available

data, and international metrics to ensure that we're being as accurate as possible during our budgeting phases.

$$\boxed{C_{energy}(M_k) = (E_{IT} + E_{cooling}) \cdot \rho_{kWh}}$$

We split energy into two sections; the actual AI workload energy requirements (including overheads) and the direct cooling related energy requirements.

2.2.1 E_{IT}

Energy is a product of the hardware's energy draw during active compute time along with the idle energy draw during idle time.

$$E_{IT} = \frac{(P_{active} \cdot T_{active}) + (P_{idle} \cdot T_{idle})}{3600 \times 1000}$$

Where:

- $P_{active} \rightarrow$ Average power consumption while the model is active computing and inferencing in Watts, per hour
- $T_{active} = N \cdot t_{inf}$ which is the total amount in seconds that the GPU is active
- $t_{inf} \rightarrow$ time the model spends inferencing and processing the request in seconds. If no empirical data available, then you can use values between 1 - 3 seconds depending on complexity of the task. [4]
- $P_{idle} \rightarrow$ the average power consumption during idle time in Watts
- $T_{idle} \rightarrow$ the idle time of the hardware in seconds (all seconds where model is not inferencing or training)

We divide by 3,600 to get Wh and then by 1,000 to get kWh. And:

$$P_{active} = GPU\ TDP \cdot a \cdot i_n$$

Where:

- $GPU\ TDP \rightarrow$ is the max power requirements of a specific GPU often available in OEM spec sheets.
- $a \rightarrow$ is an active ratio estimate of power consumption, 0.75 being an accepted value
- $i_n \rightarrow$ this is the inference count second

2.2.2 $E_{cooling}$

As a rule of thumb, (almost) all electrical power consumed by IT equipment turns into heat. That means that $Compute\,Load(kW) \approx Heat\,Output(kW)$

The conversion for this:

- $1\,kW = 3,412\,BTU/hr$
- $1\,Ton\,of\,Cooling = 12,000\,BTU/hr \approx 3.5\,kW$

Which means $100\,kW$ IT load would require about 2.85 tonnes of cooling ($100 \times 3,412 \div 12,000$)

Another general rule to keep in mind:

$CRAC\,System\,Rating \approx (IT\,Load \times 1.3) + Additional\,Overheads$ [3]

When AI workloads are involved, each GPU could draw up to 700 W and sometimes more, and when a single rack goes above 40 kW, air cooling alone becomes inefficient. Liquid cooling becomes required. [1]

We then incorporate our energy calculations to find out what our cooling requirements are as well:

If we require the value in tons we can also take $Cooling\,Tons = \frac{IT\ Load(kW) \times 1.3}{3.517}$

The above assumes E_{IT} is in kWh if in your calculation its in kW or W you need to adjust the formulas accordingly.

Where:

- $1.3 \rightarrow$ constant from available standards
- $COP \rightarrow$ the coefficient of performance (usually available on OEM data or spec sheets), interchangeable with EER, however we're using COP because it's dimensionless as it's a ratio of the same type of quantity (kW). It also fits better with our current calculations which are all in W or kW.

$$E_{cooling} = \frac{E_{IT} \times 1.3}{COP}$$

2.2.3 Sidebar: *PUE and Planning*

There is a simpler approach to estimate the total energy requirements for a the entire setup, this doesn't give the same granularity as the above mechanisms, but helps create quick estimates to determine feasibility before more detailed planning.

To estimate without knowing the full values, we can use industry benchmarks [10] [9] where the average data center PUE is around **1.56** as of 2020, however in more recent AI data centers, we can see this number improving massively to "**1.3** and sometimes better" [8]

Hyperscalers like Google and AWS report PUE values between 1.1–1.2 for AI-optimized facilities, while Uptime Institute reports an industry average of 1.55.

A **PUE** of **1.3** is therefore safe to use in your estimates, ensuring you stick to best practices during the actual build.

$$Total\ Energy\,(kWh) \approx IT\ Energy\,(kWh) \times PUE$$

3 Practical Example

The new Hopper [6] generation GPU's have been game changing, that's why our example will walk through the H200, the latest GPU's from Nvidia, from Hopper generation (hence the H in the name).

First lets understand the GPU specs [7]. Be sure to do this for whatever GPU's you're thinking of getting:

For inference we will be going with the FP8 Tensor Core as I think that is accurate enough for the use cases we have in place, but you can choose higher or lower precision points if you'd like. The numbers do have the caveat of applying sparsity.

I have also used the SXM model as that is the better GPU for inferencing for our use cases, it also has higher max FLOPs.

3.1 Specifications and Assumptions

Table 1: H200 SXM Specifications

Specification	**Value**
Price (approx.)	$40,000
Annual maintenance (20% MSRP)	$8,000
Peak FP8 performance (Tensor Core)	3,958 TFLOPs
GPU memory	141 GB
Maximum TDP	700 W
Deployment horizon	3 years

Further assumptions; with related caveats:

Table 2: Assumptions for LLM Inference

Metric	**Assumption**
Average prompt input size	1,000 tokens
Average output size (generated)	200 tokens
Model size	70B parameters
Model utilization	70%
Active power (per device)	0.75 kW
Electricity price	\$0.12/kWh
Idle power (per device)	100 W
Average inference latency	1 second
Inferences per minute (per device)	5
Cooling COP	6

- Inference Time on Average: this can range as we said based on the complexity of the task (our task is relatively simple; as you can see $S_{in} = 1,000$ and $S_{out} = 200$ which is a relatively small average token consumption for a task. Additionally, it's important to note that **H200** boasts a **1.9x improved inference speed** over H100 (the previous model) [5].
- **Electricity Price:** price is usually different from country to country, from city to city. There are every some peak tariffs as well sometimes. It's important to check relevant government and utitlities company websites for a better understanding of the price breakdown. We're using \$0.12 here just as an example.
- **COP:** is an estimate based on modern averages for liquid cooling, CRACs usually have lower average COP (around 4 - 5 depending on the generation), the less effective the cooling system the smaller this number. The OEM can also provide the COP ratings and help you compute them if you have a complicated setup.

3.2 Compute Costs

Based on the above we now have enough information to proceed:

First the computational efficiency factor:

$$\beta_k = \frac{40000 + (8000 \times 3)}{(3.958 \times 10^{15}) \times 0.7 \times (3 \times 365 \times 24 \times 3600)} \approx \$2.44 \times 10^{-19}/FLOP$$

Now we can calculate the average FLOPs based on the in and out token count for a 70B parameter model:

$$AvgFLOPs = 2 \times 70,000,000,000 \times (1000 + 200) = 1.68 \times 10^{14}$$

Now we bring it all together:

$$C_{compute} = (2.44 \times 10^{-19}) \times (1.68 \times 10^{14}) \approx 4.0992 \times 10^{-5} \textbf{or } \$\,0.000040992\,/\,inference$$

3.3 Energy Costs

Using the figures in 3.1 Specifications & Assumptions we can extrapolate the variables for the energy cost formula, the workings of which are below:

3.3.1 E_{IT}

At 5 inferences per minute

$$N = 5 \times 60 = 300\ inferences/hour$$

Each inference lasts $t_{inf} = 1s$ based on our assumptions. So active seconds / hour:

$$T_{active} = N \cdot t_{inf} = 300 \times 1 = 300s$$

Which means idle time in seconds / hour:

$$T_{idle} = 3600 - 300 = 3300s$$

Next we can calculate the IT energy per hour

$$E_{IT} = \frac{(P_{active} \cdot T_{active}) + (P_{idle} \cdot T_{idle})}{3600 \times 1000}$$

with $P_{active} = 700 \times 0.75 = 525\,W,\ P_{idle} = 100\,W$ So:

$$E_{IT} = \frac{525 \cdot 300 + 100 \cdot 3300}{3600 \times 1000} = \frac{487,500}{3.6 \times 10^6} \approx 0.135\,kWh$$

To get the energy spend per inference then:

$$= \frac{0.135}{300} \approx 4.5 \times 10^{-4}\,kWh/inf$$

3.3.2 $E_{cooling}$

Assuming COP of 6 and overhead of 30%

$$E_{cooling} = \frac{(4.5 \times 10^{-4}) \times 1.3}{6} \approx 9.75 \times 10^{-5}$$

3.3.3 Bringing it together

$$((4.5 \times 10^{-4}) + (9.75 \times 10^{-5})) \cdot 0.12 \approx 6.6 \times 10^{-5}\ \$/inf$$

3.4 Total Cost

$$C(M_k) = C_{\text{compute}}(M_k) + C_{\text{energy}}(M_k) \approx \$\,0.000106992/inf$$

This equates to roughly $107 per 1,000,000 (1 million) inferences.

4 Conclusion

Throughout this paper I have introduced a practical and modular framework for estimating costs per inference of LLM's when deployed on on-premise hardware.

By separating compute and energy costs into distinct components and structuring the formulas around measurable elements, that don't require empirical or historical or even testing data, the framework allows for a more transparent and accurate baseline for decision makers.

However, it is my belief that the broader implication is that the fit between workload and hardware is as important as the efficiency of the hardware itself. This is where an underutilized system system may look inefficient in raw-cost terms, while the same setup with a higher utilization and more load would yield very favorable economics. This framework equips organizations to quantify that relationship early. Providing a quick way to test assumptions, compare quotes, and avoid mismatched investments. This will allow you to map a value for each inference, and understand under which amount of usage (prompts and token count for complexity) will it be make fiscal sense to purchase specific hardware, and where you might break-even in raw cost per inference.

It is my plan through future work to extend the framework with memory-aware modules, latency penalties (you can visit the latency module for a deeper understanding), and empirical benchmarks, as well as integrating networking and storage considerations. For now, the approach offers enterprises a simple but rigorous way to align AI infrastructure choices with their actual usage patterns, ensuring that discussions around ROI are grounded in cost transparency rather than abstract performance claims.

It is my hope that through this iteration and future iterations of this framework, that I can allow organizations to properly understand what they need and what it is they would be getting out of specific hardware choices.

4.1 Afterthought

I also would like to note that the current costing structures for hyperscalers on AI are tiered into two a term referred to as *provisioned* and *on demand* both of which boil down costs to either reserving specific TPS (token per second) or not having priority when it comes to *on-demand* for processing your tokens and requests. Considering how new everything is; I understand that more nuanced costing models haven't had time to mature.

I feel it's important to highlight the scarcity of robust on-premise costing and estimation tools, especially in a landscape increasingly dominated by hyperscalers. To be clear, I'm not opposed to the shift toward hyperscalers; in fact, I think that a hybrid solution between on-premise workloads and cloud workloads, with sophisticated redundancies (DR & BC) is (generally) unequivocally the best choice for most organizations.

However, my focus is on the underrepresented segment that still need to operate on-premise. It's their unique requirements that drive this exploration.

Additionally, I'd like to point out that current AI-related costing models from hyperscalers

are typically divided into two options: *provisioned* and *on-demand*. These models essentially boil down to whether you reserve a specific TPS (tokens per second) capacity or accept lower priority for processing requests. Given how new this space is, I understand that more nuanced and flexible pricing structures are still evolving; I foresee more granular costing per use-case and segment in the near future, with the ability to have shared pools of provisioned TPS.

5 Glossary

Credit: *Generated detailed glossary using AI (Copilot & GPT-5) with minor manual tweaking where relevant.*

5.1 LLM (Large Language Models)

A type of artificial intelligence model trained on vast amounts of text data to understand, generate, and interact using human language. LLMs use deep learning architectures (transformer architecture, hence GPT: generative pre-trained transformer) to capture complex patterns in language, enabling tasks like translation, summarization, Q&A, and code generation.

LLMs now power modern AI applications such as chatbots, virtual assistants, search engines, and coding copilots, offering flexible context-aware language understanding.

—

5.2 ROI (Return on Investment)

A performance metric used to evaluate the efficiency and/or profitability of a particular investment. ROI in turn measures the output of a solution either:

- **Directly:** through financial gains (i.e. increased revenue or cost savings)
- **Indirectly:** through productivity gains and improvements, and quality of life enhancements (i.e. time saved, enhanced workflow efficiency)

—

5.3 TCO (Total Cost of Ownership)

A financial metric, usually an estimate, that helps decision makers determine the **complete cost** of a product, system, or solution over it's entire lifecycle. This includes:

- **Direct costs:** such as the purchase price, implementation costs, licensing fees, training costs
- **Indirect costs:** such as maintenance, management, support, potential downtime, and the eventual disposal of the asset

TCO provides a more comprehensive view of the costs associated with a particular initiative, helping organizations and decision makers make more informed investment decisions through the evaluation of long term value and sustainability of a said initiative.

—

5.4 FLOPs (Floating Point Operations)

A unit of measure that quantifies the number of arithmetic operations involving floating point numbers that a system can perform. FLOPs are commonly used to assess the computational performance of hardware, especially in fields like AI.

- 1 FLOP = 1 floating points operation (i.e. addition or multiplication)
- FLOPs/sec (or FLOP/s) = how many floating point operations a system can perform per second.

FLOPs help compare the raw processing power of HW. However, real world performance also depends on memory bandwidth, architecture, and software optimization.

—

5.5 FP (Floating Point)

A numerical representation used in computing to handle real numbers with fractional components. Floating point numbers allow for a wide range of values by using a format that includes a significand (mantissa) and exponent bits, making them ideal for scientific calculations, graphics and machine learning.

FP arithmetic enables precise and scalable computation, especially in domains requiring high numerical accuracy or large dynamic ranges.

Common Precision Points:

- FP32
- FP16
- BF16
- MXFP4

Larger number of bits (FP32) allows for capturing more information, and dimensionality, thereby allowing for higher quality and accuracy; however at the expense of much higher memory requirements, compute time, and costs. The choice of precision depends on the specific needs of the task.

—

5.6 GPU (Graphics Processing Unit)

A specialized processor designed to accelerate rendering of images and video. Modern GPUs are also widely used for parallel computing tasks, especially in AI, deep learning, and scientific simulations.

The ability to run thousands of parallel executions, high memory bandwidth, and the fact they are optimized for matrix / vector operations make them crucial for AI.

—

5.7 TTFT (Time to First Token)

A performance metric used in AI and language model inference to measure latency between a user's prompt or request and the model's first generated token. TTFT reflects how quickly a model begins responding and it's critical for real-time applications.

Lower TTFT improves user experience in interactive systems, chabots, and voice assistants by reducing *perceived delay*.

—

5.8 Intent Management

A structured approach used in agent based systems to route user queries (intents) to the most appropriate models or agents. Intent manager essentially act as *task routers*, ensuring that each query is handled by the best suited solution / agent.

Key Setup Characteristics / Steps:

1. **Intent Definition:** Identify what users expect from the system (e.g. summarization, translation, Q&A, tool calling). These expectations are formalized into intents and expanded into workflows
2. **Fallbacks:** Set up mechanisms to handle unclear user intents or prevent agents from operating outside of their intended scope
3. **Agent Selection:** Test and evaluate models to determine which are the best suited for each intent using traditional metrics and experiments
4. **Manager Construction:** Build workflows that route intents to selected models, incorporating:
    - Guardrails
    - Prompt-engineering
    - Ground truths and testing
    - Fine tuning - in the cases where there is a measurable enhancement in performance

Benefits:

- **Control:** This helps ensure a more consistent, compliant behavior across systems and agents
- **Explainability:** Every decision and output is traceable and auditable, making it suitable for enterprise and regulatory requirements

—

5.9 BTU (British Thermal Unit)

A unit of energy used to measure heat. One BTU is the amount of energy needed to raise the temperature of one pound of water by one degree Fahrenheit.

This is commonly used in HVAC systems an data center cooling to quantify thermal output or cooling capacity.

—

5.10 CRAC (Computer Room Air Conditioning)

A specialized air conditioning unit designed for cooling data centers and server rooms. CRAC systems maintain optimal temperature and humidity levels to ensure reliable operation of IT equipment.

—

5.11 COP (Coefficient of Performance) & EER (Energy Efficiency Ratio)

COP: A ratio that measures the efficiency of heating or cooling systems. It is calculated as the amount of heating or cooling provided divided by the energy consumed

EER: Similar to COP but typically used for cooling systems, EER measures the ratio of cooling output (in BTUs) to electrical input (in watts)

—

5.12 TDP (Thermal Design Power)

The maximum amount of heat a computer chip (CPU, GPU, etc.) is expected to generate under typical workloads. TDP guides the design of cooling solutions to ensure safe and efficient operation.

Helps system designers choose appropriate cooling systems and manage power consumption in high-performance computing environments.

—

5.13 PUE (Power Usage Effectiveness)

A metric used to determine the energy efficiency of a data center. It is calculated as the ratio of total facility energy to IT equipment energy. Lower PUE values indicate more efficient data centers, with less energy wasted on cooling, lighting, and other non-IT infrastructure.

References

[1] AIRSYS North America. Understanding the design & cooling of ai data centers. https://airsysnorthamerica.com/understanding-the-design-cooling-of-ai-data-centers/, 2024.

[2] Anonymous. Cost factors in agentic systems. *arXiv preprint*, 2025. URL: https://arxiv.org/abs/2506.06579.

[3] APC. Calculating total cooling requirements for data centers. https://www.apc.com/us/en/support/resources-tools/white-papers/calculating-total-cooling-requirements-for-data-centers.jsp, 2024.

[4] et al. Erdil. Empirical per-token inference latencies of large language models. *arXiv preprint*, 2025. Reports empirical latencies: 4ms (8B), 13ms (62B), 32ms (540B). Supports using 1–3s defaults for planning. URL: https://arxiv.org/abs/2506.04645.

[5] NVIDIA Data Center. Nvidia h200 tensor core gpu. https://www.nvidia.com/en-us/data-center/h200/, 2023.

[6] NVIDIA Newsroom. Nvidia announces hopper architecture – the next generation of accelerated computing. https://nvidianews.nvidia.com/news/nvidia-announces-hopper-architecture-the-next-generation-of-accelerated-computing, 2022.

[7] NVIDIA Resources. Hpc datasheet sc23. https://resources.nvidia.com/en-us-gpu-resources/hpc-datasheet-sc23, 2023.

[8] Upsite Technologies. Data center trends: Rack density rises while pue and outage frequency remain flat. https://www.upsite.com/blog/data-center-trends-rack-density-rises-while-pue-and-outage-frequency-remain-flat/, 2024.

[9] Uptime Institute. Global pues – are they going anywhere? https://journal.uptimeinstitute.com/global-pues-are-they-going-anywhere/, 2023.

[10] Uptime Institute. 2024 global data center survey. https://datacenter.uptimeinstitute.com/rs/711-RIA-145/images/2024.GlobalDataCenterSurvey.Report.pdf, 2024.

A Latency Cost ($C_{latency}(M_k)$): TTFT-Centric Approach

Latency cost reflects the impact of inference delay on business value. The most salient metric of LLM responsiveness is TTFT. This metric is directly perceived by end users, and it often determines whether the system is usable in production for the specific use case.

Therefore latency cost is determined as a function of TTFT, combining both productivity loss from waiting and potential penalties for violating *SLA*.

$$C_{latency}(M_k) = \gamma \cdot t_{TTFT}(M_k) + \lambda \cdot \max(0, t_{TTFT}(M_k) - t_{SLA})^p$$

Where:

- $\gamma \rightarrow$ productivity cost coefficient (\$/s of waiting time; e.g., employee time value)
- $t_{TTFT}(M_k) \rightarrow$ time to first token for model M_k
- $\lambda \rightarrow$ SLA penalty scaling factor
- $t_{SLA} \rightarrow$ contractual SLA response time threshold
- $p \rightarrow$ penalty exponent ($p = 1$ linear growth; $p = 2$ quadratic growth)

The objective is to capture both:

1. **Productivity:** Every second of TTFT carries a measurable productivity cost.
2. **SLA Obligations:** Crossing defined latency thresholds may introduce additional penalties under SLA terms.

A.1 TTFT Decomposition

Empirical measurement of TTFT is ideal. However, we can approximate it as:

$$t_{TTFT}(M_k) = t_{queue} \cdot \frac{\kappa f(S) L_{in}(1+\chi)}{F_{max} u \eta} \cdot t_{alloc} + t_{sched} + t_{io}$$

Where:

- $t_{queue} \rightarrow$ queuing delay under concurrent load
- $\kappa f(S) \rightarrow$ per-token FLOPs scaling with model size S
- $L_{in} \rightarrow$ input length in tokens
- $\chi \rightarrow$ prompt complexity factor
- $F_{max} \rightarrow$ maximum FLOPs/s hardware delivers
- $u \rightarrow$ utilization rate
- $\eta \rightarrow$ efficiency factor (quantization, parallelism, etc.)

- $t_{alloc}, t_{sched}, t_{io} \rightarrow$ overheads from memory allocation, scheduling, and I/O

Following Casson's transformer FLOPs derivation [**?**], FLOPs/token for inference is:

$$N = 2 \cdot d_{model} \cdot n_{layer} \cdot (2 \cdot d_{attn} + d_{ff})$$

with:

- $d_{model} \rightarrow$ hidden dimension
- $n_{layer} \rightarrow$ number of transformer layers
- $d_{attn} \rightarrow$ attention head dimension (often $= d_{model}$)
- $d_{ff} \rightarrow$ feed-forward layer size (often $4 \cdot d_{model}$)

A.2 TTFT Reduction Theory

Based on the decomposition, practical levers to reduce TTFT include:

- Reducing prompt complexity (χ) or input length (L_{in})
- Selecting smaller or quantized models (S)
- Improving utilization (u) and efficiency (η) through batching or optimized scheduling
- Mitigating queuing effects (t_{queue}) under concurrency

—

B PowerShell Cost Calculator

While I was cleaning up a script I wrote for everyone to be able to try it themselves, ChatGPT had some other ideas, and came up with something better, as it sometimes does. So here's something the majority of you will be able to run super easily. A straightforward calculator for cost-per-inference calculation.

All you have to do to run this is save it as a file, and

```
# Windows PowerShell
.\ "filename".ps1

# Linux/macOS with PowerShell Core
pwsh ./ "filename".ps1
```

Listing 1: Example run of the cost calculator

```
<#
AI_OnPrem_Cost_Calculator.ps1
Backcompatible with the original defaults; adds:
- Output switch: -Output text|json|csv (default text)
```

```
- Optional multiGPU scaling: -NumGPUs (default 1)
- Currency symbol override: -Currency '$'
- Emits beta and FLOPs/inf in output (for auditability)
- Safer guards for edge cases (e.g., zero/negative throughput, minutes)
#>

[CmdletBinding()]
param(
  # ==== Workload / Model ====
  [double]$Params = 70e9,
  [int]$TokensIn = 1000,
  [int]$TokensOut = 200,

  # ==== Hardware / Compute ====
  [double]$GpuTFLOPs = 3958, # per GPU TFLOPs/s
  [int]$NumGPUs = 1, # NEW: number of GPUs amortized together
  [double]$Utilization = 0.7,
  [double]$Years = 3,
  [double]$HardwarePrice = 40000,
  [double]$AnnualMaintenance = 8000,

  # ==== Energy / Power ====
  [double]$GpuTDP_W = 700, # per GPU
  [double]$ActivePowerRatio = 0.75,
  [double]$IdlePower_W = 100, # facility idle attributable to this node
  [double]$ElecPricePerkWh = 0.12,

  # Workload throughput / timing
  [double]$InferencesPerMinute = 5,
  [double]$InferenceTimeSec = 1,

  # ==== Cooling method selection ====
  [ValidateSet('COP','PUE')]
  [string]$EnergyMethod = 'COP',
  [double]$COP = 6,
  [double]$OverheadFactor = 1.3,
  [double]$PUE = 1.3,

  # ==== Output / Formatting ====
  [ValidateSet('text','json','csv')]
  [string]$Output = 'text',
  [string]$Currency = '$',
  [int]$Decimals = 10
)

function Throw-If($cond, $msg) { if ($cond) { throw $msg } }
```

```powershell
# ---------- Derived values ----------
$C_HW = $HardwarePrice + ($AnnualMaintenance * $Years)

# FLOPs capacity in FLOPs/sec (scale by NumGPUs)
$F_max = $GpuTFLOPs * 1e12 * [Math]::Max(1, $NumGPUs)

$T_max = $Years * 365 * 24 * 3600

Throw-If(($F_max -le 0) -or ($Utilization -le 0) -or ($T_max -le 0),
  "Throughput, utilization, and lifetime seconds must be > 0.")

$beta = $C_HW / ($F_max * $Utilization * $T_max)

$AvgFLOPs = 2.0 * $Params * ($TokensIn + $TokensOut)
$C_compute_per_inf = $beta * $AvgFLOPs

$inferencesPerHour = $InferencesPerMinute * 60.0
Throw-If($inferencesPerHour -le 0, "InferencesPerMinute must be > 0.")

$activeSecondsHour = $inferencesPerHour * $InferenceTimeSec
if ($activeSecondsHour -gt 3600) {
  Write-Warning "Active seconds per hour ($activeSecondsHour) exceed 3600.
      Clamping to 3600."
  $activeSecondsHour = 3600
}
$idleSecondsHour = [Math]::Max(0, 3600.0 - $activeSecondsHour)

# Power (scale active power by GPU count)
$P_active_W = ($GpuTDP_W * $ActivePowerRatio * [Math]::Max(1, $NumGPUs))
$P_idle_W = $IdlePower_W

$E_IT_hour_Wh = ($P_active_W * $activeSecondsHour + $P_idle_W * $idleSecondsHour)
    / 3600.0
$E_IT_hour_kWh = $E_IT_hour_Wh / 1000.0
$E_IT_per_inf_kWh = $E_IT_hour_kWh / $inferencesPerHour

if ($EnergyMethod -eq 'COP') {
  $E_cooling_per_inf_kWh = ($E_IT_per_inf_kWh * $OverheadFactor) / $COP
  $C_energy_per_inf = ($E_IT_per_inf_kWh + $E_cooling_per_inf_kWh) *
      $ElecPricePerkWh
} else {
  $E_total_per_inf_kWh = $E_IT_per_inf_kWh * $PUE
  $C_energy_per_inf = $E_total_per_inf_kWh * $ElecPricePerkWh
}

$C_total_per_inf = $C_compute_per_inf + $C_energy_per_inf
$C_per_million = $C_total_per_inf * 1e6
```

```
96
97  # Lifetime totals at the stated throughput
98  $life_infs = [Math]::Floor($inferencesPerHour * 24 * 365 * $Years)
99  $life_cost = $C_total_per_inf * $life_infs
100
101 # ---------- Output helpers ----------
102 function Money([double]$x, [int]$d) { return ($Currency + [Math]::Round($x, $d).
        ToString("F$d")) }
103
104 if ($Output -eq 'text') {
105   Write-Host ""
106   Write-Host "==== RESULTS ====" -ForegroundColor Cyan
107   Write-Host ("beta ($/FLOP) : {0}" -f ([Math]::Round($beta, 14)))
108   Write-Host ("Avg FLOPs / inference : {0}" -f ([Math]::Round($AvgFLOPs, 0)))
109   Write-Host ("Compute cost / inference : {0}" -f (Money $C_compute_per_inf
          $Decimals))
110   Write-Host ("Energy cost / inference : {0} (method: {1})" -f (Money
          $C_energy_per_inf $Decimals), $EnergyMethod)
111   Write-Host ("------------------------------------------------------------")
112   Write-Host ("TOTAL cost / inference : {0}" -f (Money $C_total_per_inf $Decimals
          ))
113   Write-Host ("Cost per 1,000,000 infs : {0}" -f (Money $C_per_million 4))
114   Write-Host ""
115   Write-Host "---- Lifetime (at stated load) ----" -ForegroundColor DarkCyan
116   Write-Host ("Total inferences (life) : {0}" -f $life_infs)
117   Write-Host ("Total cost over life : {0}" -f (Money $life_cost 4))
118   Write-Host ""
119   Write-Host "---- Key Assumptions ----" -ForegroundColor DarkCyan
120   Write-Host ("Params (N) : {0}" -f $Params)
121   Write-Host ("Tokens (in/out) : {0} / {1}" -f $TokensIn, $TokensOut)
122   Write-Host ("GPU TFLOPs/s (per) : {0}, GPUs={1}" -f $GpuTFLOPs, $NumGPUs)
123   Write-Host ("Utilization (u) : {0}" -f $Utilization)
124   Write-Host ("Horizon (years) : {0}" -f $Years)
125   Write-Host ("Hardware cost (USD) : purchase={0}, maint/yr={1}, total={2}" -f
          $HardwarePrice, $AnnualMaintenance, $C_HW)
126   Write-Host ("TDP/Active/Idle (W) : {0} / {1} / {2}" -f ($GpuTDP_W * [Math]::Max
          (1, $NumGPUs)), [Math]::Round($P_active_W,2), $P_idle_W)
127   Write-Host ("Inferences/min : {0}" -f $InferencesPerMinute)
128   Write-Host ("Inference time (sec) : {0}" -f $InferenceTimeSec)
129   Write-Host ("Elec price (USD/kWh) : {0}" -f $ElecPricePerkWh)
130   if ($EnergyMethod -eq 'COP') {
131     Write-Host ("Cooling (COP) : COP={0}, OverheadFactor={1}" -f $COP,
            $OverheadFactor)
132   } else {
133     Write-Host ("PUE : {0}" -f $PUE)
134   }
135   Write-Host ("E_IT per inf (kWh) : {0}" -f ([Math]::Round($E_IT_per_inf_kWh, 8))
```

```
    )
  Write-Host ""
  Write-Host "Tip: JSON output -> add: -Output json" -ForegroundColor DarkGray
}
elseif ($Output -eq 'json') {
  $obj = [ordered]@{
    params = $Params
    tokens_in = $TokensIn
    tokens_out = $TokensOut
    gpu_tflops_per = $GpuTFLOPs
    num_gpus = $NumGPUs
    utilization = $Utilization
    years = $Years
    capex = $HardwarePrice
    annual_maintenance = $AnnualMaintenance
    elec_price_per_kwh = $ElecPricePerkWh
    energy_method = $EnergyMethod
    cop = $COP
    overhead_factor = $OverheadFactor
    pue = $PUE
    beta_per_flop = [Math]::Round($beta, 14)
    flops_per_inf = [Math]::Round($AvgFLOPs, 0)
    compute_cost_per_inf = [Math]::Round($C_compute_per_inf, $Decimals)
    energy_cost_per_inf = [Math]::Round($C_energy_per_inf, $Decimals)
    total_cost_per_inf = [Math]::Round($C_total_per_inf, $Decimals)
    cost_per_million = [Math]::Round($C_per_million, 4)
    life_inferences = $life_infs
    life_cost = [Math]::Round($life_cost, 4)
    e_it_per_inf_kwh = [Math]::Round($E_IT_per_inf_kWh, 10)
  }
  $obj | ConvertTo-Json -Depth 3
}
else {
  # csv
  $row = [pscustomobject]@{
    params=$Params; tokens_in=$TokensIn; tokens_out=$TokensOut;
    gpu_tflops_per=$GpuTFLOPs; num_gpus=$NumGPUs; utilization=$Utilization; years=
        $Years;
    capex=$HardwarePrice; annual_maintenance=$AnnualMaintenance;
        elec_price_per_kwh=$ElecPricePerkWh;
    energy_method=$EnergyMethod; cop=$COP; overhead_factor=$OverheadFactor; pue=
        $PUE;
    beta_per_flop=[Math]::Round($beta, 14); flops_per_inf=[Math]::Round($AvgFLOPs
        ,0);
    compute_cost_per_inf=[Math]::Round($C_compute_per_inf, $Decimals);
    energy_cost_per_inf=[Math]::Round($C_energy_per_inf, $Decimals);
    total_cost_per_inf=[Math]::Round($C_total_per_inf, $Decimals);
```

```
    cost_per_million=[Math]::Round($C_per_million, 4);
    life_inferences=$life_infs; life_cost=[Math]::Round($life_cost,4);
    e_it_per_inf_kwh=[Math]::Round($E_IT_per_inf_kWh,10)
  }
  $csvPath = "cost_results.csv"
  $row | Export-Csv -NoTypeInformation -Path $csvPath
  Write-Output ("CSV written to {0}" -f (Resolve-Path $csvPath))
}
```

Listing 2: AI_OnPrem_Cost_Calculator.ps1