MVA Internship Report

Latency and Energy Optimization for Generative AI Models

Julien Delavande

Master MVA, ENS Paris-Saclay

julien.delavande@ens-paris-saclay.fr

August 2025

Internship Host: Hugging Face
Supervisor: Régis Pierrard
MVA Academic Tutor: Nathanaël Fijalkow

Abstract

The rapid adoption of Large Language Models (LLMs) and multimodal generative systems has raised growing concerns about their computational and environmental costs. This internship was focused on characterizing and optimizing the latency and energy consumption of generative AI models, combining theoretical analysis with fine-grained empirical measurements on state-of-the-art GPUs.

We first examined **micro-interactions with LLMs**, using politeness as a controlled case study to quantify the energy cost of seemingly trivial prompts such as "thank you." We then investigated **system-level design choices**, showing how numerical precision, batch size, and request scheduling can yield orders-of-magnitude differences in per-request energy use, often overshadowing architectural factors. Finally, we extended the analysis to **text-to-video diffusion models**, developing a compute-bound theoretical model and validating it across multiple open-source systems, highlighting quadratic scaling in spatial and temporal dimensions as a critical bottleneck for sustainability.

Together, these studies provide a multi-level perspective - spanning user interactions, serving infrastructure, and generative architectures - on where inefficiencies arise in inference pipelines and how they can be mitigated. Beyond their scientific contributions, these results have been integrated into Hugging Face's open-source ecosystem, promoting reproducibility and paving the way for more energy-aware deployment practices in the community.

1 Introduction

Large Language Models (LLMs) and multimodal generative systems have become central to modern AI applications, powering search engines, conversational agents, code assistants, and creative tools. Their capabilities have advanced rapidly, but so have the computational demands required to deploy them at scale. Inference - executed billions of times daily across datacenters and user devices - is now widely recognized as the dominant source of both latency bottlenecks and energy consumption. Understanding and mitigating these costs is essential, not only for efficient service delivery but also for the long-term sustainability of AI.

This internship at Hugging Face focused on developing a fine-grained understanding of latency and energy behavior in generative models. The goal was twofold: (i) to design analytical models capturing the relationship between input/output parameters, architectural choices, and hardware characteristics, and (ii) to validate these models through systematic measurement campaigns on state-of-the-art GPUs. This dual approach provides actionable insights for practitioners deploying generative models in production while contributing to the broader scientific effort of quantifying the environmental footprint of AI.

The work is organized around three complementary case studies:

- **Micro-interactions with LLMs.** Using politeness as a motivating example, we quantify the energy cost of seemingly innocuous prompts such as "thank you," highlighting how prompt length, output verbosity, and model size affect the footprint of everyday interactions with LLMs.
- **System-level design choices.** We analyze how numerical precision, batch size, and request scheduling impact inference efficiency, showing that deployment strategies can induce orders-of-magnitude variations in per-request energy use.
- **Generative video models.** We develop a compute-bound theoretical model of text-to-video diffusion systems, validate it on the WAN2.1 model, and extend it to a benchmark of six popular open-source models, exposing video generation as an extreme case of inference inefficiency.

Together, these studies form a coherent body of work that bridges theoretical modeling with empirical validation. They shed light on where and why inefficiencies arise in generative inference pipelines, and propose practical strategies - from prompt design to serving infrastructure - to mitigate their environmental impact. Additional resources, including source code and datasets used in this work, are available at:

- `github.com/JulienDelavande/benchlab`
- `huggingface.co/jdelavande`.

2 Related Work

Research on the environmental impact of machine learning has expanded rapidly over the past five years. The pioneering study of Strubell et al. [1] first quantified the carbon footprint of training a Transformer model, sparking a wave of work on the costs of ML training across tasks and hardware platforms [2, 3, 4, 5, 6]. While model training initially received the most attention due to its large up-front costs, inference is increasingly recognized as the dominant contributor to the environmental footprint of deployed AI systems [7, 8, 9]. This has motivated a shift towards fine-grained studies of inference efficiency and its determinant factors.

Energy cost of LLM inference. Inference with LLMs has been studied from multiple perspectives: energy per query [10], hardware comparisons between CPUs and GPUs [11], and workload characteristics such as prompt length and verbosity [12, 13, 14, 15]. A growing body of work highlights the discrepancy between theoretical FLOPs or utilization estimates and actual measured energy consumption [16], underlining the need for empirical benchmarks. More broadly, efficiency has been advocated as a core research metric [17], with calls for standardized reporting [18, 9], improved cost indicators [19], and system-level carbon accounting [5]. Despite this, few studies have focused on micro-interactions or the impact of factors such as output length and precision, which may appear negligible but compound significantly at scale.

System-level optimizations. Inference efficiency is shaped not only by model architecture but also by deployment choices. Quantization methods - ranging from FP16 and BF16 to INT8 and INT4 - reduce memory and compute costs through techniques such as activation-aware weight quantization [20], GPTQ [21], and FP8 formats [22]. While these methods can reduce latency and energy in compute-bound regimes, their benefits often diminish under memory-bound conditions due to dequantization overheads and bandwidth limits [23]. Batching is another key lever: it amortizes kernel overheads but can suffer from padding inefficiencies in prefill or irregular sequence lengths [8, 24]. Modern inference engines such as Hugging Face's Text Generation Inference (TGI) [25] and vLLM [26] implement continuous batching [27], kernel fusion [28, 29], and paged attention to maximize GPU utilization. Scheduling strategies like query routing [30] and speculative decoding [31] further improve throughput, though their impact on energy remains underexplored [32]. These works highlight that infrastructure and workload shaping are as important as model internals in determining sustainability.

Generative video models. Compared to text or image generation, video generation is particularly energy-intensive due to the quadratic growth of transformer-based attention

with spatial and temporal dimensions. Luccioni et al. [9] reported large disparities in energy costs across tasks, with image generation far exceeding text. Li et al. [33] quantified the energy use of Open-Sora [34], showing that iterative denoising dominates and that energy requirements scale near-quadratically with resolution. However, published studies of text-to-video models remain scarce, and most focus on single systems with narrow configurations. This leaves open a need for systematic analysis across models and scales.

Measurement frameworks. Reliable measurement tools are essential for this line of research. Frameworks such as CodeCarbon [35], pyRAPL [36], and NVIDIA's NVML API provide the basis for tracking GPU, CPU, and RAM energy usage. These tools enable fine-grained decomposition of inference into phases (prefill vs. decode) and components, supporting reproducible comparisons across models and settings.

Summary. In summary, prior work has laid the foundations for understanding the energy footprint of AI systems, but important gaps remain. While prompt design, quantization, batching, and scheduling have each been studied in isolation, few analyses integrate these perspectives into a unified empirical and theoretical framework. Video generation, in particular, remains underexplored despite its extreme computational costs. This motivates our internship work, which aims to bridge user-level, system-level, and architectural perspectives on inference efficiency.

3 Case study I: Saying Thank You to a LLM Isn't Free: Measuring the Energy Cost of Politeness

Politeness is a crucial part of human communication, and users often carry this behavior into their interactions with AI systems. However, in the context of large language models (LLMs), even a simple message such as "thank you" triggers a full inference pass, activating billions of parameters and consuming non-negligible amounts of energy. While trivial in isolation, such micro-interactions occur at massive scale and may accumulate into a substantial environmental footprint.

In this study, we use politeness as a controlled and reproducible proxy to explore the energy cost of micro-interactions with LLMs. We quantify how input length, output verbosity, and model size influence energy consumption, and decompose usage across prefill and decode phases. This analysis provides insights into the hidden costs of everyday conversational exchanges and illustrates broader trade-offs between efficiency, usability, and sustainability in human–LLM interaction.

3.1 Methodology

3.1.1 Dataset and Experimental Setup

To evaluate the energy cost of polite interactions, we constructed a dataset of 10,000 chat-based conversations ending with a "thank you" message from the user. These were derived from the `ultrachat_200k` [37] dataset and reformatted to match the instruction-following prompt template expected by `Instruct` models, simulating real-world assistant usage scenarios. Each prompt submitted to the model included the entire conversation history up to that point, ensuring a realistic multi-turn context. The resulting dataset is available at: `ultrachat_10k_thank_you`.

For every unique prompt, we performed 5 warmup runs to stabilize performance and cache behavior, followed by 10 measurement runs for each generation phase. Specifically, we conducted 10 *prefill-only* generations (constrained to a single output token) and 10 full generations (up to 256 output tokens). This allowed us to separately estimate the energy consumption of the *prefill* and *decode* phases, by subtracting the average prefill energy from the full generation energy. We also logged input/output lengths, latency, and energy consumption by hardware component for each run.

3.1.2 Hardware and Software Environment

All experiments were run on a dedicated inference server equipped with an NVIDIA H100 SXM GPU (80GB) and 8 AMD EPYC 7R13 CPU cores, with no co-scheduled jobs.

Energy was measured using:

- **GPU:** NVIDIA Management Library (NVML),
- **CPU:** `pyRAPL` (Intel RAPL counters),
- **RAM:** CodeCarbon's model-based estimation[1].

3.1.3 Models and Precision

Our core analysis focused on the **LLaMA 3.1–8B-Instruct** model in `float32`, run via the standard Transformers library [38]. In this setting, generation used `batch size 1`. We chose **LLaMA 3.1–8B-Instruct** for its strong relevance: as of the time of testing (July 2025), it is the **2nd most liked** and **3rd most downloaded** model on the Hugging Face Hub (after DeepSeek R1–680B, GPT-2, and Qwen2–7B), making it a representative and impactful open-source choice.

We then extended our tests to a range of open-source instruction-tuned LLMs, including:

- **Qwen 2.5 family:** 0.5B, 1.5B, 3B, 7B, and 14B,
- **Mistral-7B-Instruct-v0.3**.

3.2 The Energy Cost of a Simple "Thank You"

We measured the energy required to generate a reply to a polite "thank you" message using the **LLaMA 3.1–8B-Instruct FP32** model, deployed on a single NVIDIA H100 GPU. Across 10,000 such interactions, we observed a mean total energy consumption of:

- **0.202 ± 0.096 Wh** on the GPU,
- **0.024 ± 0.014 Wh** on the CPU,
- **0.019 ± 0.010 Wh** from RAM.

The total energy per polite interaction thus averages **0.245 Wh**, which is equivalent to powering a representative 5W LED bulb for nearly 3 minutes [2].

GPU usage dominates the total energy profile, with a contribution nearly an order of magnitude larger than the CPU or RAM. The GPU also exhibits higher variance, reflecting its sensitivity to sequence length and runtime context. Figure 1 shows the distribution of GPU energy per generation, which is **right-skewed with a long tail** - indicating that some completions are disproportionately costly, especially when the model produces longer or more verbose replies. This variability is primarily driven by differences

[1] `https://mlco2.github.io/codecarbon/methodology.html#ram`

[2] The "±" symbol denotes the variance in energy usage across generations in the dataset

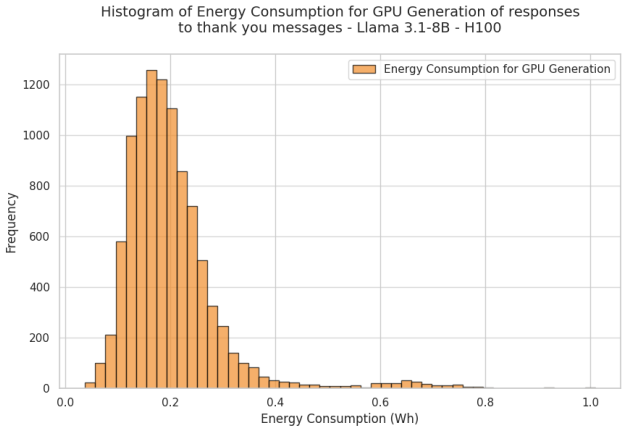
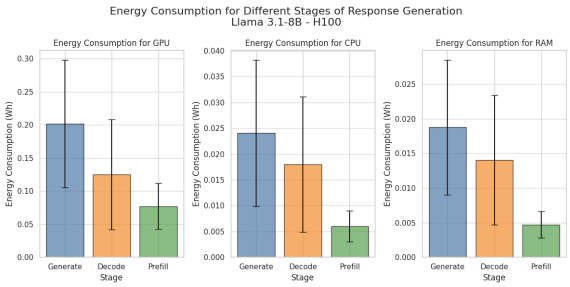


Figure 1: Distribution of GPU energy consumption across "thank you" generations. The long tail indicates variability due to prompt and output length.

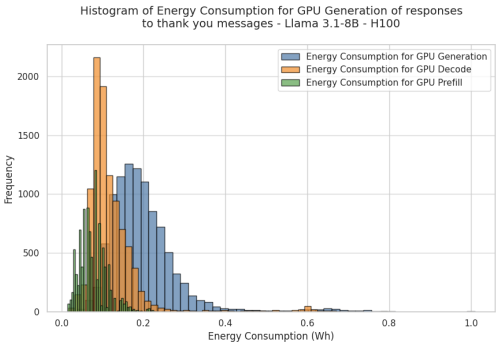
in **prompt and output length**. While "thank you" is always the last user message, the surrounding conversation history and the model's verbosity can significantly influence token count - and thus, energy consumption.

3.3 Component-wise Energy Consumption

To better understand where energy is spent during LLM inference, we break down the generation process into two main phases: *prefill*, which encodes the entire prompt and generates the first token, and *decode*, which generates the remaining tokens one by one using the cached context.



(a) Energy consumption by hardware type (GPU, CPU, RAM) and phase (prefill, decode, generate). The GPU consistently dominates across all phases, while CPU and RAM contributions remain minor.



(b) Histogram of GPU energy consumption for prefill and decode phases. Decode contributes most to the long tail due to repeated sequential steps, whereas prefill energy is substantial but occurs only once per request.

Figure 2: Breakdown of energy consumption by hardware and phase (left) and distribution of GPU usage in prefill vs. decode (right).

As shown in Figure 2a, the **GPU is by far the primary consumer of energy** throughout all of the stages of the inference process. In contrast, CPU and RAM usage

contribute only marginally to the total energy budget.

When analyzing the inference process, the prefill phase incurs the highest computational cost relative to the number of output tokens generated. We believe that this is because the model processes the entire input sequence and computes key/value caches for all input tokens before generating the first token. While computationally expensive, this step remains highly parallelizable on the GPU, which helps mitigate its efficiency.

The *decode* phase is inherently **autoregressive** and therefore less parallelizable: each new token depends on the previous one and reuses the cached context without recomputing the entire key/value cache. As a result, each decode step is lighter in isolation but must be repeated sequentially for each token generated. This sequential nature keeps the GPU occupied for a longer time, leading to higher total energy for the decode step, especially for longer output lengths. Figure 2b highlights this: while prefill consumes a significant chunk of energy up front, the long tail of high energy usage comes mainly from the repeated decode operations that accumulate over long outputs.

3.4 Energy and Latency Dependence on Input and Output Length

3.4.1 Theoretical Latency Model

To better understand how input and output lengths influence energy consumption, we adopt a closed-form latency model based on the dominant operations during LLM inference. Each GPU kernel is modeled as either compute-bound or memory-bound, depending on its floating-point operation count F_o and data transfer volume D_o, relative to hardware ceilings: floating-point throughput $F_{\max}$ and memory bandwidth $B_{\max}$. The theorical effective latency of an operation is:

$$t_o = \max\left(\frac{F_o}{F_{\max}}, \frac{D_o}{B_{\max}}\right),$$

The total latency is the sum over all operations:

$$T = \sum_o t_o.$$

An operation is *compute-bound* when its execution time is limited by arithmetic throughput, and *memory-bound* when data movement dominates. On modern GPUs, compute and memory operations can often proceed asynchronously, competing for shared resources like streaming multiprocessors (SMs) or memory buses. Our model conservatively assumes no overlap, thus upper-bounding the true latency.

We neglect kernel launch overhead, although it can become significant when GPU kernels execute faster than the CPU can dispatch them - especially in regimes with short

or tightly chained kernels (e.g., decode or layernorm blocks), where inter-kernel gaps due to CPU scheduling latency inflate total latency. Additionally, actual data transfer speeds can vary depending on caching effects - for instance, if intermediate activations are reused and remain in L2 or shared memory, memory-bound operations may be accelerated.

To account for these approximations, we introduce empirical efficiency factors for compute and memory:

$$F_{\text{eff}} = \mu_{\text{comp}} \cdot F_{\text{max}}, \quad B_{\text{eff}} = \mu_{\text{mem}} \cdot B_{\text{max}},$$

where μ_{comp} and μ_{mem} absorb multiple sources of inefficiency. These include suboptimal GPU occupancy, memory misalignment penalties, limited overlap between data fetch and compute, and variable cache residency for activations.

The values of $\mu_{\text{comp}} = 0.675$ and $\mu_{\text{mem}} = 0.443$ used throughout this section were calibrated empirically using profiling data from LLaMA 3.1–8B (FP32) inference on an NVIDIA H100 SXM GPU.

Prefill phase. This phase processes the prompt of length s through $N = 32$ transformer blocks. The dominant operations are compute bound for input sequence greater than 100 (batch = 1) and include QKV projections, feed-forward layers and FlashAttention. Using the theoretical model described above, we approximate the latency of the prefill phase with the following fitted expression:

$$t_{\text{prefill}}(s) \approx \alpha s + \beta s^2 + \gamma,$$

$$\alpha \approx 3.18 \times 10^{-4} \text{ s/token}, \quad \beta \approx 1.17 \times 10^{-8} \text{ s/token}^2.$$

$$\gamma \approx 1.68 \times 10^{-2} \text{ s}$$

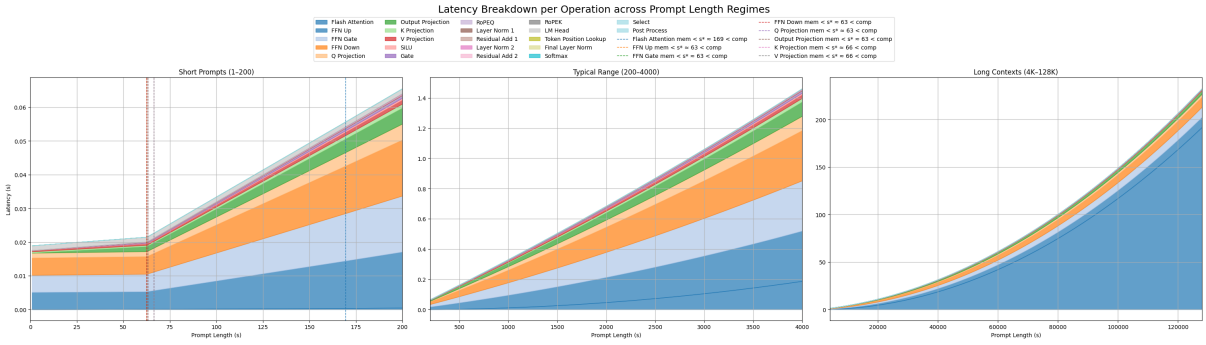


Figure 3: Theoritical latency breadown per operation accoss prompt length regimes for the Prefill phase.

In practice, latency is constant for very short prompts ($s \lesssim 60$ - the regime is memory bound), linear across most real-world prompts ($s < 4{,}000$), and quadratic only for extreme

lengths ($s > 30{,}000$) (Fig 3).

Decode phase. This phase generates g tokens autoregressively, attending to a growing context $\ell = s + t - 1$. All operations remain memory-bound for batch equals 1. Total latency scales as:

$$t_{\text{decode}}(s, g) \approx \eta g + \theta s g + \phi g^2 + \rho,$$

$$\eta \approx 2.61 \times 10^{-2} \text{ s/token}, \quad \theta \approx 3.31 \times 10^{-7} \text{ s/token}^2.$$

$$\phi \approx 5.86 \times 10^{-8} \text{ s/token}^2, \quad \rho \approx -5.32 \times 10^{-2} \text{ s}.$$

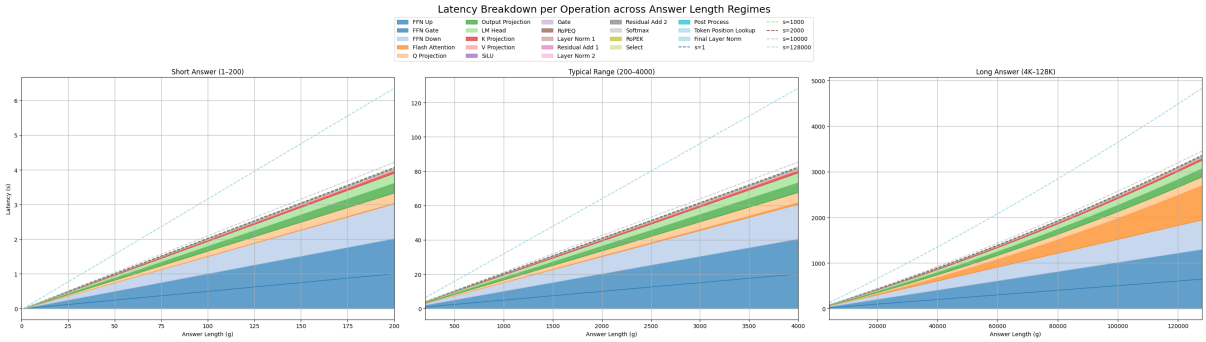


Figure 4: Theoritical latency breadown per operation accoss prompt length regimes for the Decode phase.

Quadratic effects in g exist theoretically but appear only for $g \gtrsim 10^5$, beyond practical usage (Fig 4).

3.5 Link to Empirical Energy Trends

The latency trends outlined above align closely with our energy measurements. On the H100 GPU, we can record the average power during each phase - 684 W during prefill (for average s) and 293 W during decode (for average s and batch size of 1). As a result, energy consumption becomes approximately proportional to runtime, with an effective power $\bar{P}_{\text{eff}}$ that remains nearly constant within each phase. This proportionality justifies the direct mapping between theoretical latency and empirical energy trends.

Figure 5 illustrates the observed relationship: energy in the prefill phase scales linearly with input length, while decode energy grows primarily with output length.

Prefill energy. Energy in the prefill phase increases linearly with the number of input tokens:

$$E_{\text{prefill}}(s) \approx A \cdot s + B,$$

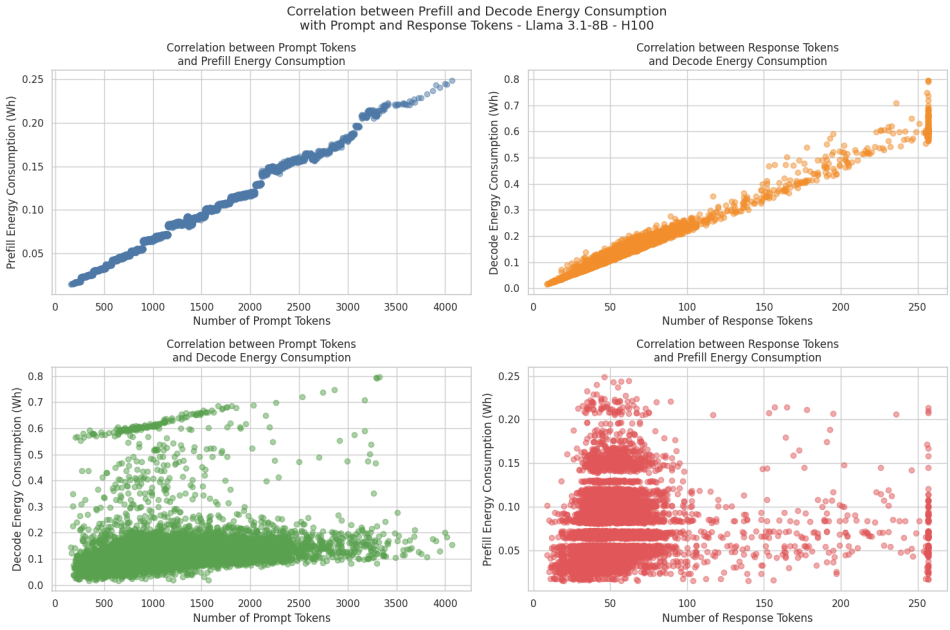


Figure 5: Correlation between token lengths and GPU energy consumption in prefill and decode phases.

with fitted values:

$$A \approx 6.05 \times 10^{-5} \text{ Wh/token}, \quad B \approx 5.00 \times 10^{-3} \text{ Wh}.$$

This is consistent with the compute-bound regime described in the latency model.

Decode energy. For generation lengths $g << 10^5$, decode energy follows the linear regime predicted by:

$$E_{\text{decode}}(s, g) \approx Cg + Dsg + E$$

with fitted values:

$$C \approx 2.13 \times 10^{-3} \text{ Wh/token} \quad D \approx 2.87 \times 10^{-7} \text{ Wh/token}^2.$$

$$E \approx -4.71 \times 10^{-3} \text{ Wh}$$

This reflects the cumulative cost of attending to a growing context at each step. The weak dependence on s corresponds to the repeated attention over the prompt tokens during generation.

No quadratic growth. While the theoretical model includes a quadratic term in g, this component remains negligible in practice given that our dataset does not include sequences long enough ($g \gtrsim 10^5$) to observe this behavior.

Kernel effects. Discrete jumps and non-linearities in the measured energy arise from low-level kernel effects such as block alignment and tiling. These artifacts do not contradict the overall linear trends and are typical of GPU workloads (see bottom right subplot on Figure 5).

Summary. Energy during inference scales linearly with prompt and generation lengths, matching the theoretical latency model under stable power conditions. The decode phase shows a bilinear dependency, with output length as the dominant factor and a minor prompt-length contribution.

3.6 Impact of Model Size on Energy Consumption

To assess how model scale influences energy usage, we extended our analysis beyond LLaMA 3–8B to include models from the **Qwen 2.5 family** (ranging from 0.5B to 14B parameters) and **Mistral–7B**. Since Qwen models share the same architecture and tokenizer, this allowed for a controlled comparison focused solely on model size.

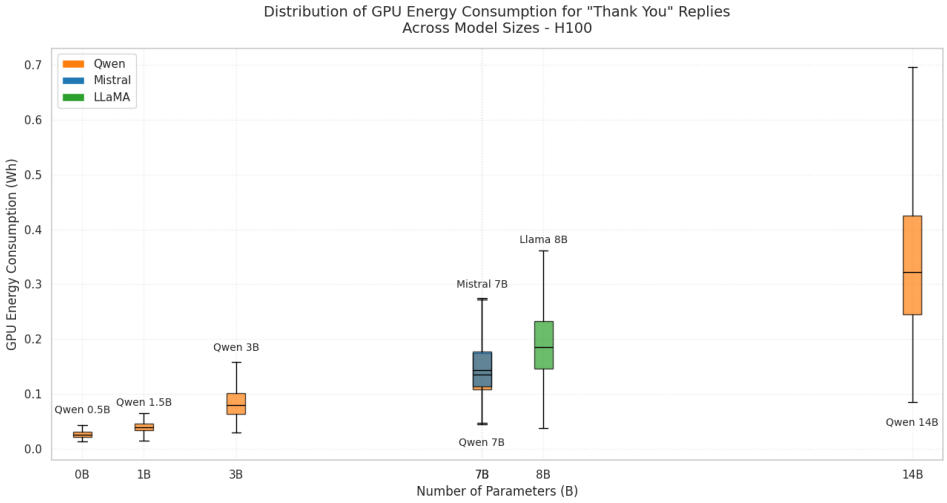
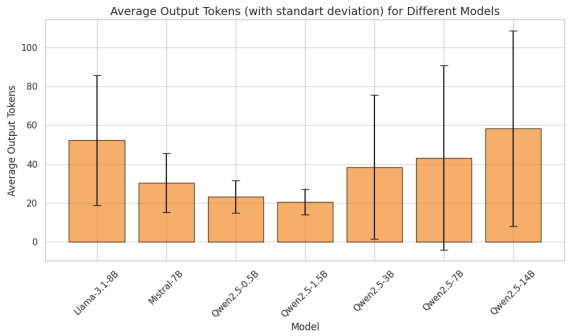


Figure 6: GPU energy consumption during generation as a function of model size. Boxes represent the distribution across 10,000 replies including polite "thank you" phrases.

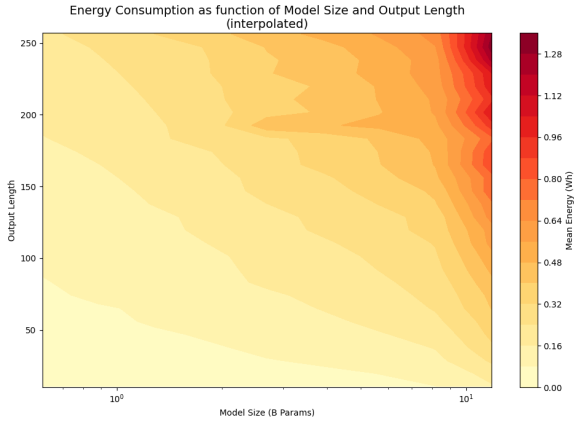
We observed that **larger models have the tendency to produce longer outputs** on average (Figure 7a). This reflects their greater capacity to elaborate responses, but also contributes to increased compute and energy during inference [39].

As shown in Figure 6, energy usage scales with model size – while smaller models are more energy-efficient, larger models incur significantly higher costs to generate responses of similar or slightly longer lengths.

To isolate the effects of model size and verbosity, we analyzed decode-phase energy as a function of both parameters. Figure 7b shows that **model size is the dominant factor**,



(a) Mean and standard deviation of output token lengths across models. Larger models tend to generate longer responses on average.



(b) Interpolated contour plot of GPU energy consumption (Wh) during the decode phase, as a function of model size (log scale) and output length.

Figure 7: (Left) Output length increases with model size; (Right) decode energy depends on both output length and model size.

with energy increasing steeply along the model axis. Output length has a secondary influence but remains within a narrower energy band.

Link to theoretical latency analysis. The observed increase in energy with model size can be directly explained by the architectural scaling effects. Specifically:

- The number of transformer blocks N grows with model size and contributes **linearly to the total latency** in both prefill and decode phases.
- In the prefill phase, dominant operations such as QKV projection and feedforward layers involve matrix multiplications with complexity $\mathcal{O}(sNh^2)$, leading to a **quadratic dependence on the hidden dimension** h.
- In the decode phase, energy scales as $\mathcal{O}(gNh^2)$ in the memory-bound regime

Together, these explain why energy increases steeply with model size: larger models have deeper networks (N), wider layers (h), and tend to generate longer outputs (g). This is reflected in the contour plot of Figure 7b, where energy increases most rapidly along the model-size axis.

In summary, scaling up model size increases energy use significantly - not only due to longer replies, but also due to larger hidden dimensions and more layers. These results emphasize the importance of considering model size–efficiency trade-offs, even in seemingly trivial interactions such as replying to a "thank you". Although larger models consume more energy, their outputs may exhibit higher helpfulness, informativeness, or linguistic fluency [40] - aspects which we do not evaluate in this study.

3.7 Conclusion and Takeaways

We used the "thank you" prompt as a reproducible micro-interaction to study the energy profile of LLM inference. This seemingly trivial interaction exposes clear structural trends in energy usage - and reflects broader trade-offs in sustainable AI deployment.

- **Input/output length matters.** Energy grows linearly with prompt and generation length. Decode dominates for long outputs due to its sequential nature.
- **Model size matters.** Larger models produce longer replies and incur steeper energy costs - both due to deeper architectures and longer inference runtimes.
- **Phase distinction is key.** Prefill is compute-bound and benefits from hardware parallelism; decode is memory-bound and more sensitive to sequential overhead.
- **Latency models are useful.** Our closed-form model aligns with empirical trends, and offers a foundation for predicting inference energy at scale.

In short: Politeness isn't free - but understanding its cost helps build more efficient and sustainable LLM deployments. By quantifying where energy is spent, we can begin to optimize not just model performance, but also its environmental footprint.

4 Case study II: Understanding Efficiency: Quantization, Batching, and Arrival Shaping in LLM Energy Use

Beyond model architecture, system-level design choices strongly influence the efficiency of large language model (LLM) inference. Numerical precision, batching strategy, and request scheduling can all reshape the balance between compute- and memory-bound workloads, leading to significant variations in both latency and energy use.

In this study, we systematically evaluate how these deployment factors affect inference efficiency on NVIDIA H100 GPUs. By analyzing quantization formats, batch sizes, and traffic patterns under realistic LLM serving conditions, we show that energy consumption per request can vary by up to two orders of magnitude. This highlights the importance of infrastructure and scheduling - not just model internals - for building sustainable and scalable LLM services. Our contributions are a detailed study of quantization across five precisions, an analysis of batch size effects including energy per token and padding trade-offs, a benchmark of Hugging Face's Text Generation Inference (TGI) server showing that traffic shaping can reduce per-request energy by up to 100×, and practical guidelines for energy-efficient LLM inference.

4.1 Experimental Setup

We benchmarked a selection of some of the most downloaded instruction-tuned open-source LLMs on Hugging Face as of July 2025, focusing on standard model sizes in the range of a few billion parameters. Our benchmark includes:

- **Qwen 2.5**: 0.5B, 1.5B, 3B, 7B, 14B
- **Mistral-7B-Instruct-v0.3**
- **LLaMA 3.1–8B-Instruct**

Each model was evaluated under five numerical formats:

- `float32`, `bfloat16`, `float16` (native support via PyTorch)
- `int8`, `int4` using `bitsandbytes` [41] quantization (via the LLM.int8() and LLM.int4() formats).

For `int8` and `int4`, we applied post-training quantization using `bitsandbytes`, which compresses the feed-forward and attention projection weights using vector-wise quantization. For `int8`, LLM.int8 performs 8-bit matrix multiplications with outlier-aware mixed precision, isolating rows or columns with large activation features and computing them in

16-bit to preserve accuracy [42]. For `int4`, weights are packed two per byte and stored in a NormalFloat4 (NF4) format; custom CUDA kernels perform on-the-fly dequantization before matmuls [43].

All models were loaded and executed using the `Transformers` library [38].

All runs were conducted on a dedicated NVIDIA H100 SXM GPU (80GB) and 8 AMD EPYC 7R13 CPU cores, with no co-scheduled jobs. GPU and CPU energy were measured using the `CodeCarbon` library [35], which leverages NVML and pyRAPL for real-time energy monitoring, while RAM energy was estimated via a CodeCarbon heuristic [3] based on CPU count and usage duration. Latency was recorded at the CUDA kernel level.

Each request was preceded by a warmup phase of 5 iterations to stabilize memory and kernel behavior. For each configuration, we repeated the same request 10 times and report the average energy and latency to reduce variability.

We reused the "thank you" prompt subset from our previous work (Section 3) which studies the energy impact of polite interactions with LLMs. This dataset provides a controlled and reproducible input distribution while preserving real-world relevance. Specifically, we used 10,000 polite prompts (ending in "thank you") sampled from a custom subset of the UltraChat-200k dataset [37], available at `ultrachat 10k`. Prompts ranged from 200 to 4000 tokens, and outputs were relatively short - typically between 10 and 300 tokens - due to the nature of the dataset. Prompts were adapted to match the input format expected by each model.

To analyze energy and latency independently for *prefill* and *decode*, we split the inference into two steps:

- **Prefill**: Forward pass over the full prompt (with generation stopped at the first token).
- **Decode**: Autoregressive generation of the remaining tokens, attending to cached context.

The full `generate` phase corresponds to the sum of *prefill* and *decode*. In practice, we isolate *prefill* by generating a single token, and obtain *decode* as the difference between the full generation and the prefill run. This decomposition enables us to capture the distinct compute regimes that characterize each phase:

The *prefill* phase is not uniformly compute-bound: for very small input sizes, most operations are memory-bound due to limited arithmetic intensity. (Memory-bound operations are limited by data movement rather than computation; although memory and compute can be executed asynchronously on GPUs, one often becomes the bottleneck, depending on the workload.) As the input length (s) increases, compute-heavy operations - such as feedforward layers and QKV (query, key, and value) projections - begin to

[3] `https://mlco2.github.io/codecarbon/methodology.html#ram`

dominate, especially in large models with wider hidden dimensions. Compute-bound operations, by contrast, are limited by the rate at which arithmetic can be performed. The transition point from memory-bound to compute-bound depends primarily on the model's hidden size, with larger models entering the compute-bound regime earlier. Increasing the batch size also accelerates this transition by increasing the FLOP-to-memory ratio.

In contrast, the *decode* phase remains fully memory-bound for smaller batch sizes, regardless of model size. This is due to the autoregressive nature of generation: each token is produced sequentially and involves computing attention over cached prompt representations at each decoding step, leading to small, fragmented memory operations. Only by increasing the batch size does the decode phase start to exhibit compute-bound characteristics.

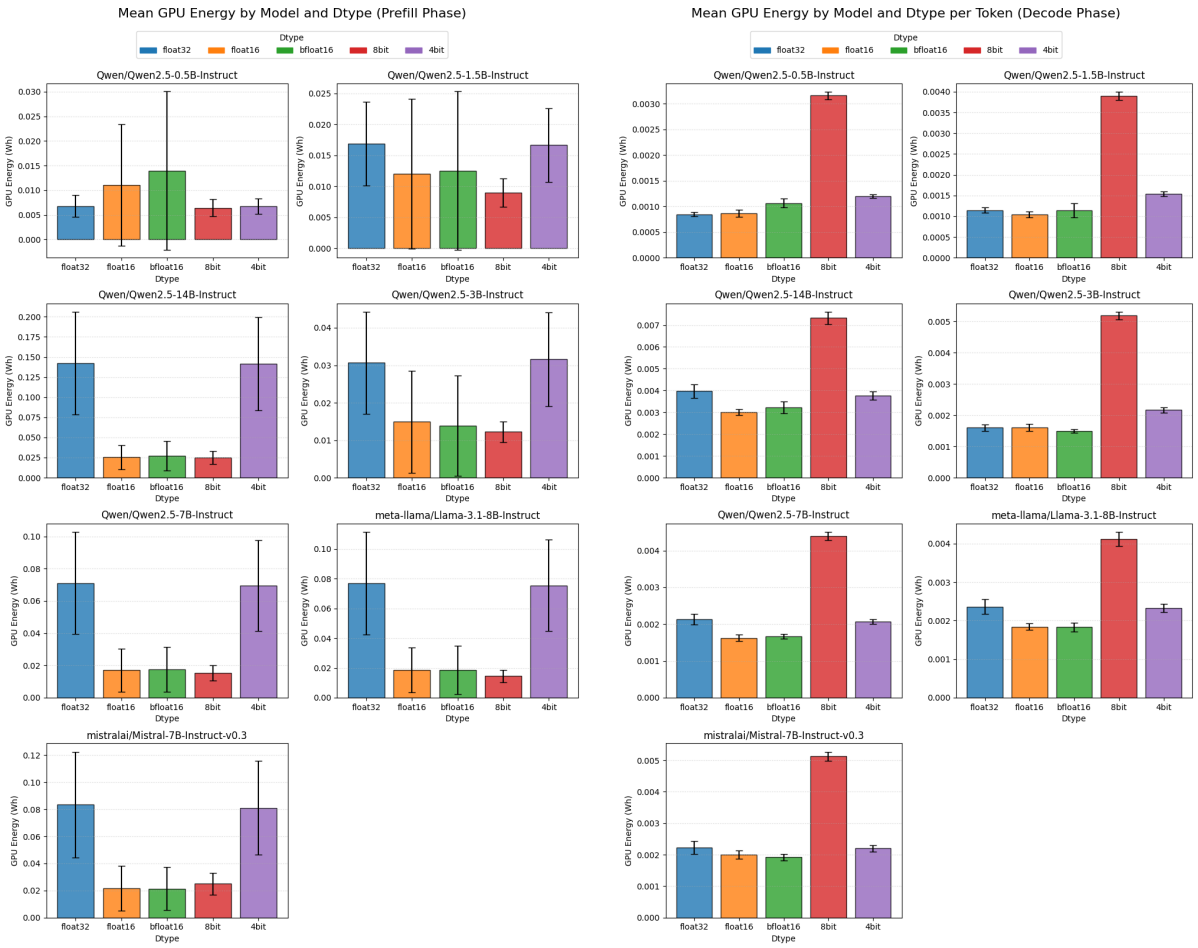
Idle time. Finally, GPU utilization can be impacted by idle times between kernels. When the CPU thread issuing kernels is slower than the GPU execution, the GPU may stall despite its asynchronous capabilities - leading to gaps where no work is scheduled. This underutilization becomes more pronounced in workloads with small or irregular kernel launches.

4.2 Impact of Numerical Precision on Latency and Energy Consumption

As LLMs grow in size, the adoption of lower-precision numerical formats - such as `bfloat16`, `int8`, or `int4`-has become a widespread strategy to reduce memory footprint and enable inference for larger models on constrained hardware. While these formats can also improve throughput and hardware utilization, their actual benefits are often phase-dependent and not always straightforward. In this section, we dissect how numerical precision impacts both latency and energy consumption across the two main phases of inference: *prefill* and *decode*. We show that precision reduction yields significant gains primarily in compute-bound regimes, whereas in memory-bound settings, aggressive quantization may introduce dequantization overheads or bandwidth saturation that offset the expected improvements.

4.2.1 Prefill Phase: Acceleration without Proportional Energy Savings

In the prefill phase, we observe up to **4× reduction in GPU energy** when switching from `float32` to lower-precision formats such as `float16`, `bfloat16`, or `int8` - particularly for larger models (e.g., LLaMA 8B or Qwen 14B) - see Figure 8a. These models are predominantly compute-bound at the input lengths seen in our dataset (typically $s_{mean} \approx 1200$), and benefit fully from the activation of **Tensor Cores**, which enable fused matrix multiplications with up to 15× higher throughput.



(a) Mean GPU energy consumption by model and dtype during the prefill phase.

(b) Mean GPU energy consumption per token by model and dtype during the decode phase.

Figure 8: Impact of model size and numerical precision (dtype) on GPU energy consumption during (a) prefill and (b) decode phases.

Smaller models, in contrast (e.g., Qwen-0.5B and 1.5B), remain memory-bound across most of the prompt lengths we tested, as their hidden sizes are smaller and their compute intensity lower. As a result, they gain little to no advantage from Tensor Core acceleration. In some cases, we even observe a slight increase in energy consumption for `float16`/`bfloat16`, likely due to the activation of specialized compute kernels (Tensor Core paths) that add overhead without enough work to amortize it (Figure 8a).

For quantized models (`int8` and `int4`), performance is further impacted by on-the-fly dequantization: during inference, weights stored in compressed integer formats are unpacked and converted to higher-precision tensors (typically `float16`/`bfloat16` or `float32`) before computation. This unpacking adds extra kernel launches and memory movement, which can partially negate the benefits of quantization, especially when the operations are memory-bound or irregular.

While latency does decrease significantly in many of these configurations - up to 10× in large models (Figure 16) - the energy savings are smaller. This is due to a higher

average power consumption when using Tensor Cores: they complete the computation faster, but at a higher instantaneous power draw. As a result, the time is shorter but the power is higher, limiting the total energy saved.

4.2.2 Decode Phase: Quantization Pitfalls in Memory-Bound Regimes

In contrast to prefill, the *decode* phase is fully memory-bound for all model sizes and sequence lengths considered. Each generated token reuses cached activations (KV caching) and performs attention over the accumulated context, with little opportunity for parallel compute acceleration.

As a result, energy per generated token remains largely invariant across float32, float16, and bfloat16, with minor improvements (or slight degradations) in both energy (Figure 8b) and latency (Figure 17). This suggests that lower-precision Tensor Cores do not provide significant benefits in this memory-bound regime. *Theoretically*, in a bandwidth-limited regime, - and thus latency and energy per token - should scale inversely with the memory word size b_w: reducing from `float32` (32 bits) to `float16` (16 bits) or `int8` (8 bits) should yield ideal 2× or 4× gains, respectively. However, such improvements are not observed in practice.

The reason lies in the energy profile of memory-bound workloads: while kernels may run slightly faster with lower precision, the GPU spends a disproportionate amount of time idle between kernel launches, waiting for synchronization, scheduling, or small fragmented memory operations. Since GPU idle power remains non-negligible - typically around 120 W even when no kernel is running - reducing kernel duration has little effect on total energy per token. The energy saved from faster compute is offset by the energy burned during idle time.

Quantized formats like `int8` and `int4` further exacerbate this issue: they introduce additional dequantization kernels that are small, memory-bound, and irregular, increasing the number of launches and stream fragmentation. As a result, we observe higher energy consumption with `int8`-often 2–3× more than float32-despite moving fewer bytes (Figure 8b).

Modern GPUs also transfer memory in fixed-width chunks (e.g., 32–64 bytes), so 4-bit formats do not reduce memory bandwidth proportionally. Combined with memory misalignment and suboptimal coalescing, this results in negligible or even negative energy gains from quantization in the decode phase. In fact, we find that `int4` performs similarly to float32, reinforcing the notion that in memory-bound phases with high kernel fragmentation, reducing numerical precision is insufficient to meaningfully reduce energy use.

In summary: numerical precision reduction yields the most benefit in the *prefill* phase of large models, where compute dominates. In contrast, the *decode* phase remains memory-

limited, and aggressive quantization (e.g., int8 or int4) may incur overheads that outweigh theoretical savings.

4.3 Batch Size Effects on Energy Efficiency

Batching is one of the most effective levers for improving throughput and reducing per-request overhead in LLM inference. By processing multiple sequences in parallel, batching amortizes fixed costs such as memory transfers and kernel launch overheads. However, its impact on energy consumption depends on the inference phase (prefill vs decode), the compute regime (compute- vs memory-bound), and the presence of padding. In this section, we analyze how GPU energy scales with batch size for **LLaMA 3.1–8B (float32)**, using the `transformers` library in static batching mode. We separate the analysis into two perspectives:

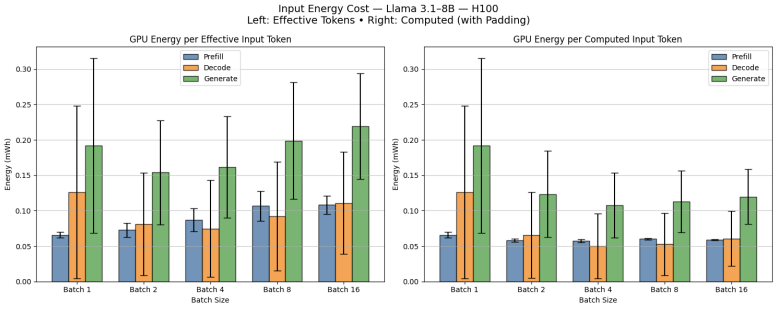
- Energy per *input token*, distinguishing between **effective** (excluding padding) and **computed** (including padding) tokens;
- Energy per *output token*, where effective = computed since completed sequences are dropped automatically.

Input token normalization: trade-offs between padding and parallelism. To understand how batch size affects different phases of inference, we first normalize energy by the number of *input tokens*.

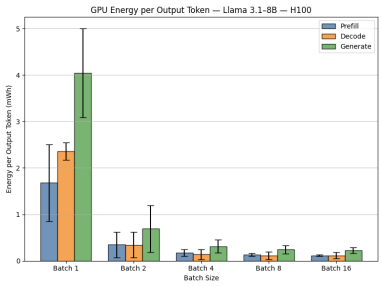
On the left of Figure 9a, the energy per *effective input token* in the prefill phase increases steadily with batch size due to padding. As sequences are padded to match the longest one, the compute-bound prefill phase performs extra work on padded tokens, leading to inflated energy per useful token.

In the decode phase, we observe a U-shaped curve: batching improves memory reuse and reduces launch overheads, but larger batches increase the number of prompt tokens per sequence, which in turn increases the cost of each attention step. The optimal batch size for decode is reached at $b = 4$ in our setup. This U-shape carries over to the total generate phase, which sees minimal energy per effective input token at $b = 2$, a compromise between prefill waste and decode gains. At $b = 16$, energy per token increases by nearly 25% compared to this optimal point.

When energy is normalized by *computed input tokens* (right of Figure 9a), a different picture emerges. Prefill energy per token remains constant, as expected for a compute-bound workload where energy scales linearly with FLOPs. Decode energy per computed token decreases with batch size, but the gains plateau around $b = 4$ as the marginal benefits of parallelizing attention computations diminish for the sequence lengths considered. The generate phase follows the same trend, reaching about 65% of the energy per token observed at $b = 1$.



(a) GPU energy per **input token**. Left: Effective tokens (excluding padding); Right: Computed tokens (including padding).



(b) GPU energy per **output token** (effective = computed).

Figure 9: GPU energy consumption per token on LLaMA 3.1–8B. (a) Input-side energy depends on token type and padding; (b) Output-side energy remains consistent across requests.

Output token normalization: efficient batching across all phases. In Figure 9b, we normalize energy by the number of *output tokens*. Here, all tokens are **effective** because transformers automatically drop completed sequences from the batch, avoiding padding overheads.

We observe consistent improvements across all phases. Energy per output token decreases rapidly with batch size and follows a roughly logarithmic trend. The memory-bound *decode* phase benefits the most: matrix multiplications over cached keys and values dominate its cost, and batching enables better amortization of memory transfers. Prefill, though compute-bound, also appears more efficient in this metric, since its fixed cost is shared across a larger number of generated tokens. Lastly, full generation shows the same trend, confirming that larger batches lead to longer kernels and reduced idle times, further improving energy efficiency.

Conclusion. Batching improves energy efficiency across all inference phases, but through different mechanisms. In the *prefill* phase, gains are limited by padding, which inflates compute without contributing useful work. In the *decode* phase, batching brings strong benefits up to $b = 4$, after which parallelism yields diminishing returns. Normalizing by output tokens confirms consistent efficiency gains due to reduced overheads and longer, better-amortized kernels. Overall, optimal batch size depends on the chosen normalization and reflects a trade-off between parallelism and padding waste.

4.4 Energy Efficiency with TGI and Arrival Shaping

To simulate production-like deployment scenarios, we ran LLaMA 3.1–8B and 70B in `bfloat16` using Hugging Face's `text-generation-inference` (TGI) server (v3.3.4). TGI enables continous batching and integrates multiple inference optimizations such as more

kernel fusion. This section investigates how usage patterns and in particular, request arrival timing affect - batching quality and energy efficiency.

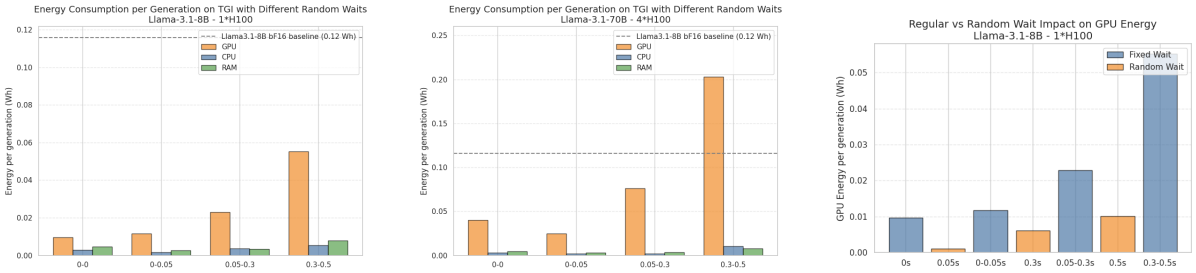
4.4.1 Methodology

We evaluated the per-request energy consumption under different inter-arrival patterns:

- **Random delays**: each request i is sent at $t_i = i \cdot \Delta$, with $\Delta \sim \mathcal{U}(0, 1)$ seconds.
- **Fixed intervals**: regular delays between requests (e.g., every 50ms, 300ms, or 500ms).

In both cases, we sent 10,000 generation requests to the TGI server. Energy consumption was tracked via nvml on the GPU host, and averaged over all requests.

4.4.2 LLaMA 8B: Arrival Shaping Unlocks Large Gains



(a) LLaMA 8B: mean energy per request (GPU, CPU, RAM) under random arrival.

(b) LLaMA 70B: same setup as in (a), scaled up to larger model size.

(c) LLaMA 8B: energy per request under fixed vs. random inter-arrival delays.

Figure 10: Impact of inter-arrival delay and model size on energy per request. (a) and (b): Mean energy for LLaMA 8B and 70B under random delays. (c): Comparison of fixed vs. random delays at 8B scale.

For LLaMA 3.1–8B, switching from the standard `transformers` library (with sequential request handling) to Hugging Face's `text-generation-inference` (TGI) server (with burst-mode batching) reduces the mean energy per request from 1.2×10^{-1} **Wh** to 9.6×10^{-3} **Wh** (Figure 10a). This 12.5× improvement highlights the impact of continuous batching and backend optimizations in TGI (see next subsection 4.4.4).

Further improvements are possible with fixed inter-arrival delays. As shown in Figure 10c, using a constant spacing of 50 ms reduces energy to as low as 1.1×10^{-3} **Wh per request**, corresponding to a 100× energy reduction relative to the naive baseline (LLaMA 8B-BF16 using the standard `transformers` backend) - achieved purely via improved batch consistency and GPU utilization.

4.4.3 LLaMA 70B: Scaling Benefits Hold at Large Scale

We repeated the experiment on LLaMA 3.1–70B (4×H100s), keeping the same generation settings. Despite the 10× increase in model size and the multi-GPU context, TGI achieved a per-request energy consumption as low as 2.4×10^{-2} **Wh** - significantly lower than the naive baseline for 8B-BF16 (1.2×10^{-1} **Wh**) (Figure 10b). This confirms that dynamic batching and traffic shaping scale effectively to large models and hardware setups (see next subsection 4.4.4).

4.4.4 Interpretation and Mechanisms

Two main mechanisms explain TGI's strong performance:

- **Continuous batching**: Incoming requests are incrementally batched at the token level as they arrive. Feedforward operations (e.g., MLP, QKV projections) are executed jointly across all active sequences, while attention is batched via paged mechanisms that group memory accesses efficiently across requests. This allows dynamic, low-latency batching without waiting for full prompts.
- **Kernel fusion and caching**: Fused operations (e.g., QKV projections, FFN layers) reduce intermediate memory writes and improve cache locality, further lowering DRAM usage and power draw.

Arrival shaping directly affects both mechanisms. Regular spacing ensures a steady stream of aligned requests, minimizing idle GPU time and improving the average batch size. Random delays still help by introducing jitter, but fixed spacing offers the most consistent utilization.

Summary. TGI combines efficient kernel execution with continuous batching strategies that adapt to incoming traffic. By shaping request arrivals - even with lightweight delay patterns - one can drastically improve batching quality and reduce energy consumption. These results suggest that **user-side scheduling and backend inference optimizations are jointly critical** to making LLM deployment more sustainable.

4.5 Macro Impact Estimate

To contextualize our results, we estimate the energy footprint of serving LLaMA 8B at scale. In our baseline setup (`float32`, no batching), the mean GPU energy per request is 1.2×10^{-1} **Wh** (Figure 2a). At 10^6 requests per day, this yields:

$$\text{Total\_energy} = 10^6 \times 1.2 \times 10^{-1}\ \text{Wh} = 1.2 \times 10^2\ \text{kWh/day}$$

This is equivalent to the daily electricity use of **over 10 French households**[4].

With optimized serving - using `bfloat16`, TGI, and regular arrival intervals - the mean energy drops to 1.1×10^{-3} **Wh/request**, yielding:

$$\text{Total\_energy} = 10^6 \times 1.1 \times 10^{-3}\ \text{Wh} = 1.1 \times 10^0\ \text{kWh/day}$$

This corresponds to a $> 100\times$ reduction, achieved solely through system-level improvements. These results emphasize that sustainable LLM deployment depends not only on model size or architecture, but also on scheduling and infrastructure.

4.6 Conclusion and Takeaways

Energy efficiency in LLM inference is not solely dictated by model architecture or size. Instead, our experiments reveal a complex interplay between numerical precision, batch shaping, and serving configuration - each of which can dramatically affect latency and power draw.

- **Precision matters- but only in compute-bound regimes.** Lower-precision formats (e.g., `bfloat16`, `int8`) yield significant speedups and energy savings during prefill, particularly for large models. However, in memory-bound phases like decoding, quantization often fails to improve - and may even worsen - efficiency due to overheads like dequantization.
- **Batching is critical to efficiency.** Both static and dynamic batching reduce energy per token by amortizing compute and memory overheads. However, prefill is sensitive to padding inefficiencies, requiring careful shaping (e.g., bucketing) to avoid regressions.
- **Serving infrastructure shapes sustainability.** Our experiments with TGI demonstrate that the *how* of inference - i.e., the scheduling of requests - can impact energy consumption by up to two orders of magnitude, even with the same model and hardware.
- **Energy profiling should be phase-aware.** Decode and prefill exhibit fundamentally different compute characteristics, and should be measured and optimized separately. Reporting aggregate energy alone may obscure key bottlenecks or inefficiencies.

Taken together, our findings argue for a more holistic view of inference efficiency - one that includes not just model optimization, but also system design and traffic shaping. As

[4]Based on an average of 4,255 kWh/year per household in France, i.e., ∼11.7 kWh/day. Source: `https://www.fournisseurs-electricite.com/compteur/consommation-electrique/moyenne`

LLMs continue to scale and proliferate, such systemic improvements will be critical to making their deployment environmentally sustainable.

5 Case Study III: Video Killed the Energy Budget: Characterizing the Latency and Power Regimes of Open Text-to-Video Models

Text-to-video (T2V) generation represents one of the most computationally demanding applications of generative AI. Producing even a few seconds of coherent video requires large diffusion transformers, dozens of denoising steps, and high spatial and temporal resolutions, resulting in long runtimes and substantial energy consumption.

In this study, we develop a compute-bound analytical model of inference for WAN2.1-T2V and validate its predictions through controlled benchmarks. We then extend our analysis to six popular open-source T2V systems, comparing their latency and energy profiles under default settings. Our results reveal quadratic scaling with spatial and temporal dimensions, making video generation an extreme case of inference inefficiency and a critical target for future optimization.

5.1 Theoretical Model of Latency and Energy

To ground our analysis, we focus on the **WAN2.1-T2V-1.3B** model [44], which serves as our reference architecture. WAN2.1 is representative of modern latent text-to-video diffusion systems: a pretrained text encoder provides conditioning, a timestep embedding MLP injects the diffusion step index, a large DiT (Diffusion Transformer) performs the bulk of spatio-temporal denoising, and a VAE decoder maps latent tensors back to pixel space. This structure is shown in Figure 11. The same framework can be applied to other recent models with minor adjustments. WAN2.1 is also the most downloaded text-to-video model on the Hugging Face Hub at the time of writing, motivating its selection for an in-depth study.

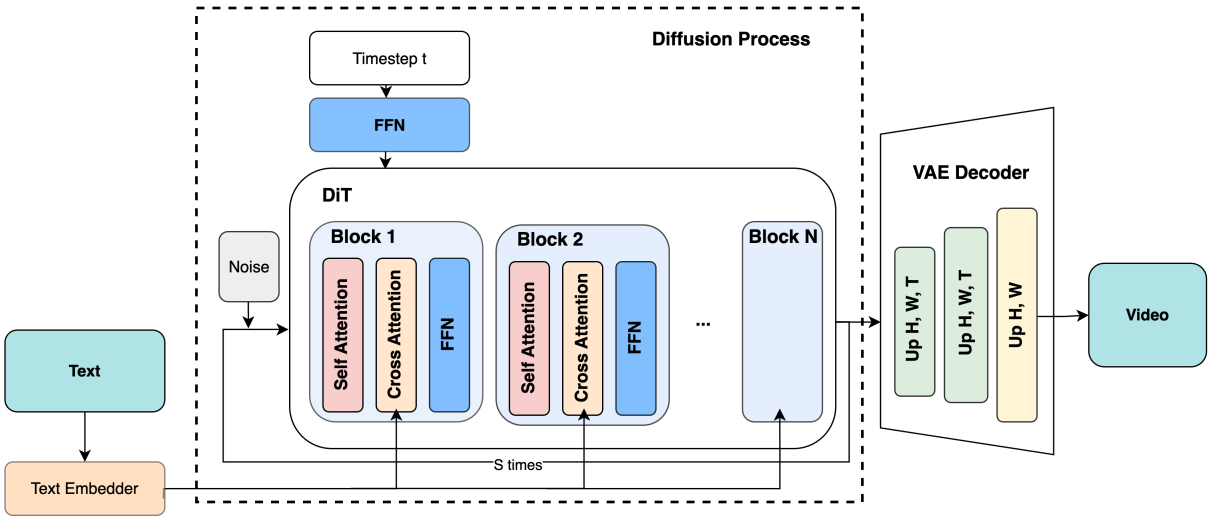


Figure 11: Simplified architecture of WAN2.1-T2V-1.3B.

We are then able to derive a compute-bound analytical model of WAN2.1 inference, decomposing FLOPs by operator and predicting latency and energy as explicit functions of resolution (H, W), number of frames T, and denoising steps S.

5.1.1 Compute vs. Memory-Bound Regimes

On modern GPUs such as the NVIDIA H100, inference kernels can be either:

- **Compute-bound**, when execution is limited by arithmetic throughput (FLOP/s).
- **Memory-bound**, when limited by memory bandwidth.

Profiling shows that the main operators of WAN2.1 inference (self-attention, cross-attention, MLPs, VAE convolutions) are predominantly compute-bound. GPU utilization remains saturated, and power traces indicate negligible CPU-induced idle time. We therefore adopt a compute-bound model, following the classic roofline formulation [45], where latency is proportional to total FLOPs divided by sustained throughput. This approximation is consistent with prior studies of large-scale transformer workloads [46, 47, 48, 49, 50].

5.1.2 Notation and Constants

We follow the HPC convention where one multiply-add corresponds to two FLOPs. Throughout, $H \times W$ denotes the spatial resolution, T the number of frames, S the number of denoising steps, N the number of DiT layers, d the hidden size, f the MLP expansion factor, m the text conditioning length, g the number of classifier-free guidance (CFG) passes, and ℓ the latent token length seen by the DiT. A complete list of symbols, constants, and hardware parameters is provided in Appendix D.

The DiT token length ℓ grows with the spatial (H, W) and temporal (T) dimensions of the latent grid:

$$\ell = \left(1 + \frac{T}{4}\right) \frac{H}{16} \frac{W}{16}.$$

5.1.3 Operation-Level FLOP Breakdown

The total FLOPs per video generation can be decomposed into contributions from the text encoder, timestep MLP, the diffusion transformer (DiT), and the VAE decoder, see table 1. A full derivation of these FLOP formulas is provided in Appendix D, where we detail each operator (self-attention, cross-attention, MLP, VAE, text encoder, timestep MLP).

Table 1: FLOP cost of WAN2.1-T2V-1.3B components. Top: once per video. Bottom: per denoising step (to be multiplied by gS). Symbols are defined inline in Section 5.1, with the complete list deferred to Appendix D.

Component	**FLOPs**	**Notation**
	Once per video	
Text encoder (T5)	$p_{\text{text}}\, L_{\text{text}} \Big(8md_{\text{text}}^2 + 4m^2 d_{\text{text}} + 4f_{\text{text}} md_{\text{text}}^2 \Big)$	F_{text}
VAE decoder convolutions	$\sum_{j=1}^{N_{\text{dec,conv}}} 2\, k_t^{(j)} k_h^{(j)} k_w^{(j)}\, C_{\text{in}}^{(j)} C_{\text{out}}^{(j)}\, T^{(j)} H^{(j)} W^{(j)}$	$F_{\text{VAE,conv}}$
VAE decoder 2D "middle" attention	$T_* \Big(8\, C_*^2 L_* + 4\, L_*^2 C_* \Big)$	$F_{\text{VAE,mid-attn}}$
	Per denoising step (multiply by gS)	
DiT		
Self-attention (N layers)	$N \left(8\ell d^2 + 4\ell^2 d \right)$	F_{self}
Cross-attention (N layers)	$N \left(4\ell d^2 + 4md^2 + 4\ell md \right)$	F_{cross}
MLP (N layers)	$N \left(4f\ell d^2 \right)$	F_{mlp}
Timestep MLP (shared across layers)	$2\, d_\tau\, d \;+\; 14\, d^2$	F_τ

5.1.4 Total FLOPs

The total FLOPs for generating a video of spatial size $H \times W$, T frames, and S steps is:

$$F_{\text{total}} = F_{\text{text}} + F_{\text{VAE,conv}} + F_{\text{VAE,mid-attn}} + Sg \cdot \Big(F_{\text{self}} + F_{\text{cross}} + F_{\text{mlp}} + F_\tau \Big).$$

We define μ as the ratio between sustained and peak throughput:

$$\mu = \frac{F_{\text{total}} / D_{\text{measured}}}{\Theta_{\text{peak}}}.$$

Assuming compute-bound execution with empirical efficiency μ, and letting Θ_{peak} denote the GPU's theoretical peak throughput in dense BF16, the total latency D_{total} of generating a video can be approximated as:

$$D_{\text{total}} \approx \frac{F_{\text{total}}}{\mu\, \Theta_{\text{peak}}}.$$

In practice, the H100 provides a dense BF16 peak of $\Theta_{\text{peak}} = 989\,\text{TFLOP/s}$ (NVIDIA datasheet), but this level is unattainable. The empirical efficiency μ thus acts as a correction factor, reflecting both hardware under-utilization (tile misalignment, kernel overheads, memory-bound ops) and approximations of our latency model. For WAN2.1 – after performing the experiments explained in section 5.2 – we obtain $\mu \approx 0.456$, consistent with sustained FLOP utilization of 30–63% reported for large-scale transformer inference on H100s [48, 49, 50]. We calibrated μ by linear regression of measured latencies against theoretical FLOPs across our experiments, which yielded $\mu = 0.456$ with negligible

overhead and $R^2 = 0.998$.

5.1.5 Energy Model

Since sustained GPU power remains close to $P_{\max}$ during inference, the total energy consumed E_{total}:

$$E_{\text{total}} \approx P_{\max} \cdot D_{\text{total}}.$$

Thus, energy and latency scale proportionally.

5.1.6 Predicted Scaling Regimes

From these equations, we can anticipate distinct computational regimes:

- **Quadratic scaling in spatial and temporal dimensions.** Since the DiT token length ℓ grows linearly with H, W, and T, the self- and cross-attention terms contribute $\mathcal{O}(\ell^2)$ FLOPs, leading to quadratic growth in latency and energy as resolution or frame count increases.
- **Linear scaling in denoising steps.** Each step applies the same sequence of N transformer layers, so the ideal cost scales as $\mathcal{O}(S)$.
- **Negligible contributions from auxiliary components.** The text encoder is run once per video, and the timestep MLP adds only a small overhead per step. Likewise, the VAE decoder scales linearly with voxel count $T \times H \times W$ and is quickly dominated by the quadratic DiT cost.

In summary, the theoretical model predicts that WAN2.1 inference is **transformer-dominated and compute-bound**, with quadratic regimes in spatial and temporal dimensions, linear dependence on denoising steps, and minor overhead from conditioning networks. These predictions will be validated against empirical measurements in Section 5.3.

5.2 Methodology

Our methodology combines two complementary perspectives. First, we perform controlled micro-benchmarks on **WAN2.1-T2V-1.3B**, our reference model, to validate the scaling regimes predicted by the theoretical model (Section 5.1). Second, we benchmark a diverse set of recent open-source text-to-video models under default settings, to situate WAN2.1 within the broader ecosystem.

5.2.1 Hardware and Measurement Protocol

All experiments were conducted on a dedicated NVIDIA H100 SXM GPU (80GB HBM3) paired with an 8-core AMD EPYC 7R13 CPU, with no co-scheduled jobs. We measured GPU and CPU energy using `CodeCarbon` [35], which interfaces with NVML and pyRAPL, and estimated RAM energy using CodeCarbon's default heuristic[5].

To reduce noise, each measurement included two warmup iterations, followed by five repeated runs. Inference used the Hugging Face `Diffusers` library [51] with default generation parameters.

5.2.2 Controlled Scaling Experiments on WAN2.1-T2V-1.3B

To validate the theoretical model, we systematically varied the three key structural parameters: resolution, number of frames, and denoising steps. Since the text encoder always pads or truncates prompts to a fixed length of 512 tokens, the specific choice of prompt does not affect runtime. We therefore fixed a single prompt and applied the same warmup-and-repetition protocol as above to isolate structural scaling laws.

- **Spatial resolution:** from 256×256 to 3520×1980, both dimensions divisible by 8 (model constraint). Frames and steps fixed.
- **Temporal length (frames):** from 4 to 100 in increments of 4 (model constraint). Resolution and steps fixed.
- **Denoising steps:** from 1 to 200. Resolution and frames fixed.

For each configuration we logged total latency (seconds) and energy for each hardware component (GPU / CPU / RAM).

5.2.3 Cross-Model Benchmark

To provide a bird's-eye view of energy and latency costs across current systems, we selected a diverse set of models spanning different architectures and parameter scales (Table 2), focusing on those that are among the most downloaded and trending on the Hugging Face Hub at the time of writing.

For this benchmark, we generated **50 different prompts** per model. Each prompt was measured with the protocol above (2 warmups, 5 runs), yielding robust averages and standard deviations that capture both runtime noise and input variability.

- **AnimateDiff** [52](License) - lightweight motion-layer diffusion.
- **CogVideoX-2b/5b** [53] (License) - cascaded base + refiner stages.

[5] `https://mlco2.github.io/codecarbon/methodology.html#ram`

- **LTX-Video-0.9.7-dev** [54](License) - autoregressive temporal modeling.
- **Mochi-1-preview** [55](License) - large-scale diffusion optimized for motion realism.
- **WAN2.1-T2V (1.3B and 14B)** [44](License) - high-resolution latent diffusion with DiT backbone.

Table 2: Default generation settings for each model (taken from Hugging Face model cards).

Model	Steps	Resolution (HxW)	Frames	FPS
AnimateDiff	4	512×512	16	10
CogVideoX-2b	50	480×720	49	8
CogVideoX-5b	50	480×720	49	8
LTX-Video	40	512×704	121	24
Mochi-1-preview	64	480×848	84	30
WAN2.1-T2V-1.3B	50	720×1280	81	15
WAN2.1-T2V-14B	50	720×1280	81	15

We did not assess perceptual quality to isolate compute behavior; instead, these experiments confront the predicted quadratic and linear regimes (Section 5.1) with actual scaling laws and scheduler-induced deviations. All code, prompts, and configurations are available in an anonymized repository at GitHub repo, and all generated videos are released on the Hugging Face Hub via an organization page.

5.3 Empirical Findings

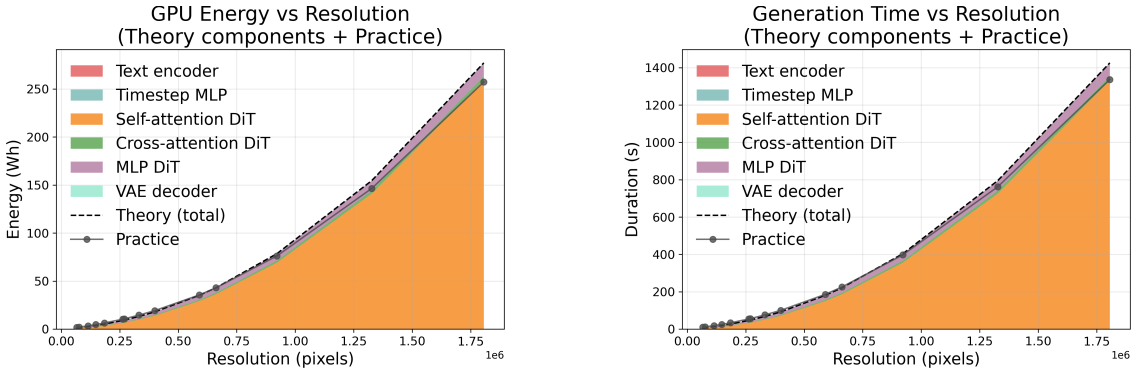
We now compare the theoretical predictions of Section 5.1 with empirical measurements – first by conducting a fine-grained validation on **WAN2.1-T2V-1.3B** and comparing measured energy and latency against theoretical curves as resolution, temporal length, and denoising steps vary. We then situate these results in the broader context of other open-source video generation models.

5.3.1 Validation on WAN2.1-T2V-1.3B

In this section we focus exclusively on *GPU energy and latency*, since GPU accounts for 80–90% of the total consumption and dominates inference cost. Figures show theoretical predictions (stacked areas by operator: self-attention, cross-attention, MLP, VAE, text encoder, timestep MLP) with empirical measurements overlaid as points with error bars.

Spatial Resolution Increasing the resolution from 256×256 to 3520×1980 (frames - 81 and steps - 50 fixed) *causes both latency and energy to grow quadratically.* Theoretical predictions (stacked by operator) and empirical measurements are compared in Figure 12.

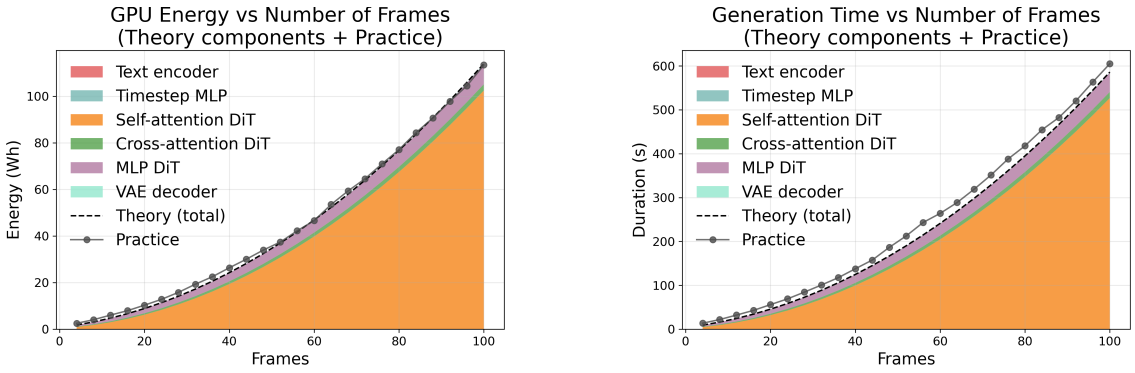
The agreement remains strong across the entire range, with modest deviations at high resolutions (see Table 3). The VAE contribution remains minor compared to the DiT blocks.



(a) GPU energy vs. spatial resolution (b) Latency vs. spatial resolution

Figure 12: Empirical results (points) vs. theoretical predictions (stacked areas per operator) as a function of resolution. Both energy and latency follow the predicted quadratic regime.

Temporal Length (Frames) Varying the number of frames from 4 to 100 (resolution - 720×1280 and steps - 50 fixed) also induces *quadratic growth* in both latency and energy, as shown in Figure 13. This behavior directly follows from the quadratic dependence of attention on the token count ℓ. The model closely tracks empirical results, with errors reported in Table 3.

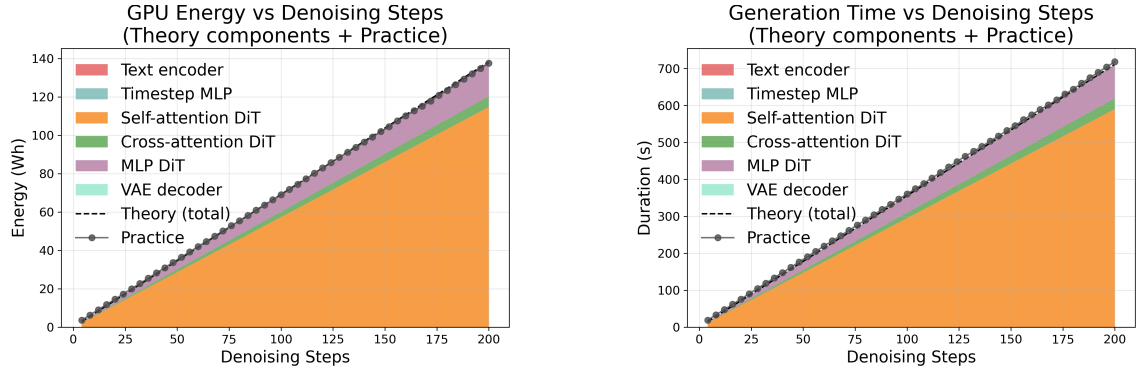


(a) GPU energy vs. number of frames (b) Latency vs. number of frames

Figure 13: Empirical results (points) vs. theoretical predictions (stacked areas per operator) as a function of temporal length. Both metrics follow the quadratic regime predicted by the model.

Denoising Steps In contrast to resolution and frame count (resolution - 720×1280 and frames - 81 fixed), scaling with the number of denoising steps is *perfectly linear*, exactly as

predicted by the theoretical model. Each additional step applies the same N transformer layers, leading to a cost that grows proportionally with S. Figure 14 shows near-perfect alignment between predictions and measurements, with errors below 2% (Table 3).



(a) GPU energy vs. denoising steps

(b) Latency vs. denoising steps

Figure 14: Empirical results (points) vs. theoretical predictions (stacked areas per operator) as a function of denoising steps. Both energy and latency scale linearly with S, in near-perfect agreement with the compute-bound model.

Table 3: Mean percentage error (MPE) between theoretical predictions and empirical measurements.

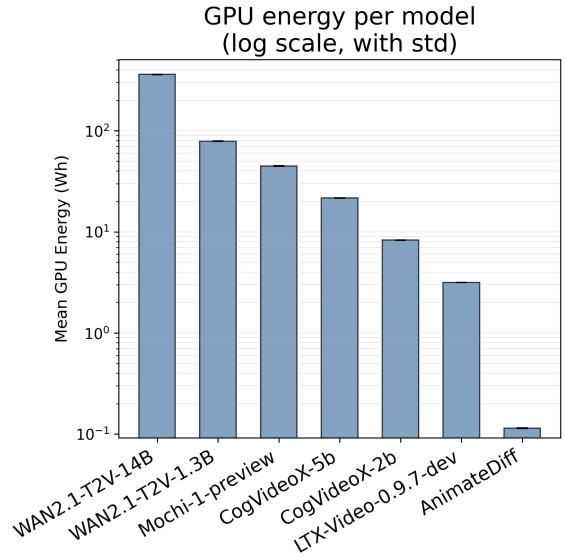
	Energy	**Latency**
Resolution scaling	11.6%	14.0%
Temporal length	6.6%	10.5%
Denoising steps	1.9%	1.9%

5.3.2 Cross-Model Comparison

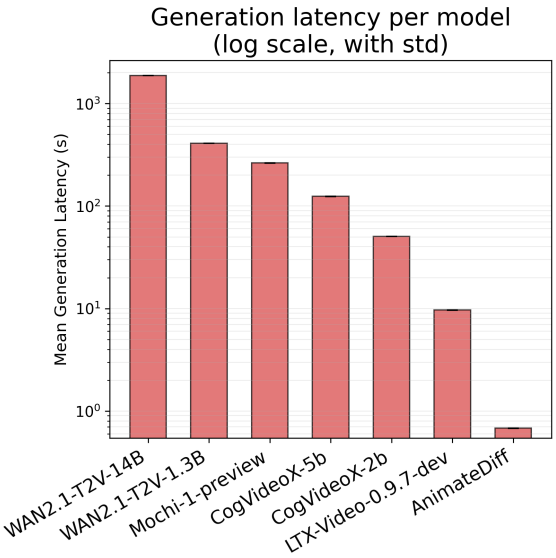
Finally, we compare average GPU energy consumption, latency, and component-wise energy shares across seven open-source text-to-video models under their default generation settings (Figure 15).

We observe orders-of-magnitude disparities: **AnimateDiff** requires only **0.14 Wh** in total, while **WAN2.1-T2V-14B** consumes over **415 Wh**, a factor of nearly 3000×. Latency follows a similar trend, with lightweight models producing clips in less than a second, while large-scale architectures such as WAN2.1-14B or Mochi require several minutes of inference. These differences stem from:

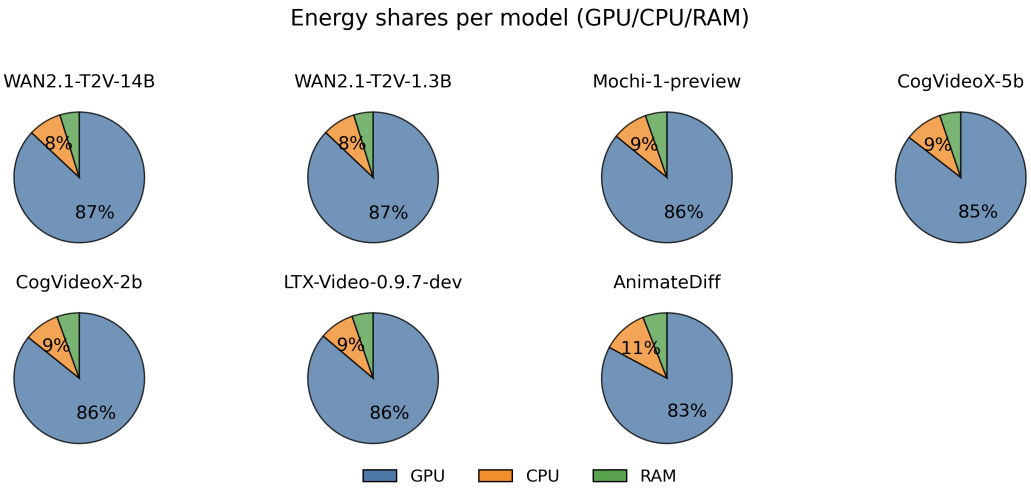
- **Model size:** larger models (WAN2.1-14B, Mochi) process more parameters per step.
- **Sampling steps:** AnimateDiff runs in 4 steps vs. 60–64 for others.



(a) GPU energy per model for one video



(b) Generation latency per model for one video



(c) Energy shares (GPU/CPU/RAM)

Figure 15: Cross-model comparison of energy and latency. Top: GPU energy and latency (log scale, with std). Bottom: relative contributions of GPU, CPU, and RAM.

- **Video length:** frame count and FPS vary significantly.
- **Architectural complexity:** cascaded pipelines (CogVideoX) require multiple stages.

As shown in the bottom panel, GPU consistently dominates energy consumption (>80%) across all models, confirming a compute-bound regime with high GPU utilization. CPU and RAM contributions remain secondary, though slightly more pronounced in cascaded or multi-stage pipelines.

Table 4: Cross-model average latency and energy consumption (default settings). All values are reported as mean ± std.

Model	**Latency (s)**	**GPU (Wh)**	**CPU (Wh)**	**RAM (Wh)**
WAN2.1-T2V-14B	1875 ± 2.1	359.7 ± 0.5	35.6 ± 4.0	19.8 ± 0.02
WAN2.1-T2V-1.3B	410 ± 0.5	78.8 ± 0.1	7.4 ± 0.4	4.3 ± 0.01
Mochi-1-preview	263 ± 0.5	44.7 ± 0.2	4.6 ± 0.01	2.8 ± 0.01
CogVideoX-5B	124 ± 0.4	21.6 ± 0.05	2.4 ± 0.03	1.3 ± 0.004
CogVideoX-2B	50.6 ± 0.2	8.3 ± 0.03	0.84 ± 0.04	0.53 ± 0.002
LTX-Video-0.9.7-dev	9.7 ± 0.01	3.16 ± 0.006	0.32 ± 0.002	0.19 ± 0.001
AnimateDiff	0.68 ± 0.002	0.115 ± 0.001	0.016 ± 0.0001	0.008 ± 0.00003

5.4 Discussion

Our results confirm that WAN2.1 inference operates in a **compute-bound regime**, where latency and energy scale quadratically with spatial (H, W) and temporal (T) dimensions, and linearly with denoising steps (S). The close match between theory and measurement validates the analytical model and provides clear guidance for practitioners.

Implications for efficiency. Quadratic scaling in H, W, and T means that even modest increases in resolution or video length incur steep costs: doubling any of these dimensions in isolation yields $\sim 4\times$ more compute, while scaling multiple dimensions compounds multiplicatively (e.g., H and W doubled $\rightarrow 16\times$). Thus, *output size control* is a powerful lever: reducing spatial or temporal length often saves more than architectural changes. In practice, offering presets (e.g., "low resolution, low frames" vs. "high fidelity") balances user needs with energy cost.

Validated linear regime in steps. In contrast, denoising steps scale linearly, with measured costs matching theoretical predictions once empirical efficiency μ is applied. This makes S a reliable knob for latency–quality trade-offs: halving steps roughly halves both latency and energy.

Opportunities for model-level improvements. The public Hugging Face implementation of WAN2.1 lacks inference-time optimizations, but the original paper suggests effective techniques: (i) *diffusion caching*, reusing redundant attention/CFG activations for up to $1.62\times$ savings, and (ii) *quantization*, using FP8/INT8 mixed precision for $\sim 1.27\times$ speedup without loss. Other avenues include step pruning, low-rank attention, and kernel fusion to better exploit GPU tensor cores.

Broader implications. Video diffusion is far more costly than text or image generation. Normalized per output, Luccioni et al. [9] report average costs of ∼0.002 Wh for text classification, 0.047 Wh for text generation, and 2.9 Wh for image generation. By comparison, generating a single short video with WAN2.1–T2V–1.3B consumes nearly ∼90 Wh. This places video diffusion roughly 30× more costly than image generation, 2,000× than text generation, and 45,000× than text classification. At scale, the quadratic growth in (H, W, T) implies rapidly increasing hardware and environmental costs, highlighting the need for hardware-aware optimizations and sustainable model design.

5.5 Limitations and Conclusion

Limitations. Our analysis provides a detailed characterization of WAN2.1–1.3B using the open-source Hugging Face codebase. As such, it does not capture potential improvements from internal optimizations such as diffusion caching, quantization, or kernel fusion. The theoretical model also assumes uniform attention cost and ignores memory hierarchy effects, which may cause deviations for small inputs or extreme aspect ratios.

Energy measurements were conducted on a single hardware platform (NVIDIA H100 SXM) and may not generalize to other accelerators or low-power deployments (e.g., L4 or consumer GPUs). We deliberately excluded perceptual quality from our scope, leaving open the question of energy–fidelity tradeoffs. Finally, many production T2V systems (e.g., Veo) also generate audio, whose contribution to energy cost remains unexplored.

Conclusion. We presented a systematic study of latency and energy consumption in text-to-video generation. Through fine-grained experiments on WAN2.1, we validated a simple analytical model that predicts quadratic scaling with spatial and temporal dimensions, and linear scaling with denoising steps. Cross-model benchmarks confirmed that this compute-bound regime extends broadly across recent open-source systems, with orders-of-magnitude disparities in cost depending on model size, sampling strategy, and video length.

These findings highlight both the structural inefficiency of current video diffusion pipelines and the urgent need for efficiency-oriented design. Promising avenues include diffusion caching, low-precision inference, step pruning, and improved attention mechanisms. We hope this work serves as both a benchmark reference and a modeling framework to guide future research on sustainable generative video systems.

6 Discussion and Limitations

6.1 Cross-Case Discussion

The three case studies presented in this report address different levels of the inference stack - from user interactions to system deployment and model architecture - yet together they outline a consistent picture of where inefficiencies arise and how they can be mitigated.

From micro-interactions to macro costs. The politeness study illustrates how even trivial conversational habits, such as appending a "thank you," trigger full inference passes with measurable energy footprints. While negligible in isolation, these costs accumulate at scale, given the billions of requests processed daily by LLMs. This highlights the importance of understanding usage patterns and designing prompts, interfaces, or model behaviors that minimize unnecessary computation without compromising user experience.

System-level levers. The analysis of quantization, batching, and request scheduling shows that system-level choices often have a greater impact on energy efficiency than the architecture itself. Quantization delivers strong gains only in compute-bound regimes, whereas batching reduces per-request costs by amortizing overheads, and traffic shaping can yield two orders-of-magnitude efficiency improvements in realistic serving scenarios. These findings emphasize that sustainable deployment cannot be achieved solely by improving models; it requires joint optimization of the serving stack and workload management.

Architectural scaling in video models. The study of text-to-video generation exposes the structural inefficiency of current multimodal systems. Quadratic scaling in spatial and temporal dimensions leads to extreme energy costs, orders of magnitude higher than text or image generation. This serves as a boundary case for inference sustainability, underscoring the need for new architectural innovations such as diffusion caching, step pruning, or low-rank approximations, as well as user-facing controls over output size and fidelity.

Methodological implications. Across all case studies, we found that phase-level decomposition (prefill vs. decode) and hardware-aware modeling were essential to interpreting empirical measurements. Latency models that distinguish compute - from memory-bound regimes align closely with observed energy trends, suggesting that theoretical frameworks can complement measurement in predicting efficiency across tasks, models, and infrastructures.

6.2 Limitations

While the three case studies in this report provide novel insights into the latency and energy behavior of generative AI models, several limitations constrain the generality and scope of our findings. These limitations fall into four main categories: hardware coverage, methodological scope, measurement fidelity, and external validity.

Hardware coverage. All experiments were conducted primarily on NVIDIA H100 GPUs, with limited exploration of alternative accelerators. Although the H100 represents the state of the art in datacenter deployment, it differs significantly from other hardware commonly used for inference, including earlier NVIDIA generations (A100, L4, consumer RTX), AMD GPUs, Google TPUs, and specialized inference accelerators (AWS Inferentia, Cerebras). Each of these platforms has distinct architectural trade-offs in terms of compute throughput, memory bandwidth, cache hierarchy, and power management. As a result, the phase-level behaviors and efficiency trends we report may not generalize directly. Extending these benchmarks across hardware is crucial to fully capture the diversity of real-world deployments.

Methodological scope. Our analysis focused on isolated inference workloads, with batch size, prompt length, and output length controlled systematically. In practice, production serving environments introduce additional complexities, including multi-tenant scheduling, distributed inference across multiple GPUs, and network I/O. These factors can significantly alter latency and energy profiles. For example, in multi-GPU settings, interconnect bandwidth and synchronization overheads may dominate, while network traffic can become the primary contributor to energy costs in cloud deployments. By focusing on controlled single-node benchmarks, we gain precision but lose some ecological validity.

Measurement fidelity. Although we employed established tools such as CodeCarbon, NVML, and pyRAPL, energy measurement remains imperfect. GPU power telemetry can suffer from sampling delays, averaging effects, and discrepancies between instantaneous and reported values. RAM energy, in particular, was estimated via CodeCarbon's heuristic rather than direct measurement, limiting the accuracy of absolute values. Furthermore, we measured hardware energy consumption in Wh, without including embodied emissions from hardware manufacturing or datacenter overheads such as cooling and redundancy. As such, our numbers reflect direct operational energy but underestimate the full environmental impact of generative inference.

Model and task diversity. We concentrated on a limited set of open-source models: the LLaMA 3.1 and Qwen families for LLMs, Mistral-7B, and WAN2.1 plus five additional

systems for video generation. While representative, this selection excludes many important classes of models, such as large proprietary LLMs (GPT-4, Claude, Gemini), multimodal foundation models, and lightweight architectures optimized for edge deployment. Likewise, our case studies targeted conversational prompts and text-to-video generation; other tasks such as retrieval-augmented generation, reasoning, or long-context summarization may exhibit different scaling laws and energy characteristics.

Quality-efficiency trade-offs. In all experiments, we deliberately excluded evaluations of model quality, focusing solely on latency and energy. However, in real deployments, efficiency cannot be considered in isolation: reductions in energy may come at the cost of degraded accuracy, fluency, or perceptual fidelity. For instance, quantization may introduce numerical errors, batching may increase response latency, and step pruning in diffusion models may reduce visual quality. Without jointly measuring quality, it is difficult to fully characterize the trade-off space in which practitioners must operate.

Scope of carbon and cost accounting. Our results are reported in watt-hours, which provide a direct measure of operational energy but not of higher-level externalities. Translating these numbers into carbon emissions requires assumptions about datacenter energy mix, cooling efficiency, and geographic deployment. Likewise, mapping energy to financial cost depends on electricity pricing, hardware amortization, and operational expenses. We did not attempt these translations systematically, which limits the ability to directly connect our measurements to environmental and economic outcomes.

Simplifications in theoretical modeling. The latency and energy models developed in this work adopt several simplifications. We assumed compute-bound execution for video generation and ignored kernel launch overheads or cache effects in LLMs. In practice, asynchronous compute–memory overlap, inter-kernel scheduling delays, and caching behavior can significantly affect runtime and energy. While we calibrated empirical efficiency factors (μ) to absorb these effects, this reduces interpretability and may hide second-order behaviors. Future work could integrate detailed kernel-level profiling to refine these models.

User-level scope. The politeness case study used short, controlled prompts as a reproducible proxy for micro-interactions. While this offers clarity, it may not reflect the full variety of conversational usage, where context length, dialogue structure, and multilingual interactions play a significant role. User behavior can amplify or mitigate inefficiencies in ways not captured by our controlled setup. Broader datasets of real-world conversations would provide stronger evidence of aggregate impact.

Summary. Overall, these limitations do not undermine the validity of our main findings, but they constrain their generality and scope. The results should be interpreted as controlled benchmarks and modeling exercises, rather than direct estimates of real-world deployment costs. Addressing these limitations - through broader hardware coverage, richer measurement tools, inclusion of quality trade-offs, and system-level deployments - represents an important avenue for future work.

7 Conclusion and Perspectives

This internship contributes a multi-level investigation of the latency and energy costs of generative AI inference, combining theoretical analysis with empirical benchmarking on state-of-the-art GPUs. The key insights are:

- Energy costs emerge not only from large-scale training but also from everyday inference, including ubiquitous micro-interactions.
- System-level design choices such as precision, batching, and scheduling can reshape energy efficiency by orders of magnitude, often more than architectural factors.
- Video generation exemplifies the structural limits of current generative pipelines, with quadratic scaling in resolution and frame count making sustainability a pressing concern.

Beyond their scientific contribution, these results were integrated into Hugging Face's open-source ecosystem, fostering transparency, reproducibility, and practical uptake by the community.

Looking forward, several directions appear promising. Expanding measurements to diverse hardware accelerators (e.g., AMD GPUs, TPUs, low-power devices) would strengthen generalizability. Integrating carbon accounting and cost metrics would connect raw energy measurements to real-world environmental and economic impacts. Finally, coupling efficiency analysis with quality evaluation would illuminate the trade-offs between sustainability and user-facing performance.

In short, this work demonstrates that efficiency must be addressed across all levels of the inference stack - from user prompts to serving infrastructure and model architecture. By doing so, we can begin to design generative AI systems that are not only powerful and versatile, but also sustainable and responsible.

References

[1] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in NLP. *CoRR*, abs/1906.02243, 2019. URL `http://arxiv.org/abs/1906.02243`.

[2] David Patterson, Joseph Gonzalez, Quoc Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David So, Maud Texier, and Jeff Dean. Carbon emissions and large neural network training, 2021. URL `https://arxiv.org/abs/2104.10350`.

[3] Peter Henderson, Jieru Hu, Joshua Romoff, Emma Brunskill, Dan Jurafsky, and Joelle Pineau. Towards the systematic reporting of the energy and carbon footprints of machine learning. *CoRR*, abs/2002.05651, 2020. URL `https://arxiv.org/abs/2002.05651`.

[4] Udit Gupta, Young Geun Kim, Sylvia Lee, Jordan Tse, Hsien-Hsin S Lee, Gu-Yeon Wei, David Brooks, and Carole-Jean Wu. Chasing carbon: The elusive environmental footprint of computing. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 854–867. IEEE, 2021.

[5] Carole-Jean Wu, Ramya Raghavendra, Udit Gupta, Bilge Acun, Newsha Ardalani, Kiwan Maeng, Gloria Chang, Fiona Aga, Jinshi Huang, Charles Bai, et al. Sustainable AI: Environmental implications, challenges and opportunities. *Proceedings of machine learning and systems*, 4:795–813, 2022.

[6] Alexandra Sasha Luccioni, Sylvain Viguier, and Anne-Laure Ligozat. Estimating the carbon footprint of bloom, a 176b parameter language model, 2022. URL `https://arxiv.org/abs/2211.02001`.

[7] Sasha Luccioni, Boris Gamazaychikov, Sara Hooker, Regis Pierrard, Emma Strubell, Yacine Jernite, and Carole-Jean Wu. Light bulbs have energy ratings—so why can't ai chatbots? *Nature*, 632(8026):736–738, 2024.

[8] Jared Fernandez, Clara Na, Vashisth Tiwari, Yonatan Bisk, Sasha Luccioni, and Emma Strubell. Energy considerations of large language model inference and efficiency optimizations, 2025. URL `https://arxiv.org/abs/2504.17674`.

[9] Sasha Luccioni, Yacine Jernite, and Emma Strubell. Power hungry processing: Watts driving the cost of ai deployment? In *The 2024 ACM Conference on Fairness, Accountability, and Transparency*, FAccT '24, page 85–99. ACM, June 2024. doi: 10.1145/3630106.3658542. URL `http://dx.doi.org/10.1145/3630106.3658542`.

[10] Siddharth Samsi, Dan Zhao, Joseph McDonald, Baolin Li, Adam Michaleas, Michael Jones, William Bergeron, Jeremy Kepner, Devesh Tiwari, and Vijay Gadepally. From

words to watts: Benchmarking the energy costs of large language model inference, 2023. URL https://arxiv.org/abs/2310.03003.

[11] Brad Everman, Trevor Villwock, Dayuan Chen, Noe Soto, Oliver Zhang, and Ziliang Zong. Evaluating the carbon impact of large language models at the inference stage. pages 150–157, 11 2023. doi: 10.1109/IPCCC59175.2023.10253886.

[12] Marta Adamska, Daria Smirnova, Hamid Nasiri, Zhengxin Yu, and Peter Garraghan. Green prompting. 2025. URL https://arxiv.org/abs/2503.10666.

[13] Soham Poddar, Paramita Koley, Janardan Misra, Sanjay Podder, Navveen Balani, Niloy Ganguly, and Saptarshi Ghosh. Brevity is the soul of sustainability: Characterizing llm response lengths, 2025. URL https://arxiv.org/abs/2506.08686.

[14] Kuofeng Gao, Yang Bai, Jindong Gu, Shu-Tao Xia, Philip Torr, Zhifeng Li, and Wei Liu. Inducing high energy-latency of large vision-language models with verbose images, 2024. URL https://arxiv.org/abs/2401.11170.

[15] Grant Wilkins, Srinivasan Keshav, and Richard Mortier. Offline energy-optimal llm serving: Workload-based energy models for llm inference on heterogeneous systems, 2024. URL https://arxiv.org/abs/2407.04014.

[16] Jared Fernandez, Clara Na, Vashisth Tiwari, Yonatan Bisk, Sasha Luccioni, and Emma Strubell. Energy considerations of large language model inference and efficiency optimizations, 2025. URL https://arxiv.org/abs/2504.17674.

[17] Roy Schwartz, Jesse Dodge, Noah A. Smith, and Oren Etzioni. Green ai, 2019. URL https://arxiv.org/abs/1907.10597.

[18] Arya Tschand, Arun Tejusve Raghunath Rajan, Sachin Idgunji, Anirban Ghosh, Jeremy Holleman, Csaba Kiraly, Pawan Ambalkar, Ritika Borkar, Ramesh Chukka, Trevor Cockrell, Oliver Curtis, Grigori Fursin, Miro Hodak, Hiwot Kassa, Anton Lokhmotov, Dejan Miskovic, Yuechao Pan, Manu Prasad Manmathan, Liz Raymond, Tom St. John, Arjun Suresh, Rowan Taubitz, Sean Zhan, Scott Wasson, David Kanter, and Vijay Janapa Reddi. Mlperf power: Benchmarking the energy efficiency of machine learning systems from microwatts to megawatts for sustainable ai, 2025. URL https://arxiv.org/abs/2410.12032.

[19] Mostafa Dehghani, Anurag Arnab, Lucas Beyer, Ashish Vaswani, and Yi Tay. The efficiency misnomer. *CoRR*, abs/2110.12894, 2021. URL https://arxiv.org/abs/2110.12894.

[20] Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. Awq: Activation-aware weight quantization for llm compression and acceleration, 2024. URL https://arxiv.org/abs/2306.00978.

[21] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers, 2023. URL https://arxiv.org/abs/2210.17323.

[22] Paulius Micikevicius, Dusan Stosic, Neil Burgess, Marius Cornea, Pradeep Dubey, Richard Grisenthwaite, Sangwon Ha, Alexander Heinecke, Patrick Judd, John Kamalu, Naveen Mellempudi, Stuart Oberman, Mohammad Shoeybi, Michael Siu, and Hao Wu. Fp8 formats for deep learning, 2022. URL https://arxiv.org/abs/2209.05433.

[23] Yujun Lin, Haotian Tang, Shang Yang, Zhekai Zhang, Guangxuan Xiao, Chuang Gan, and Song Han. Qserve: W4a8kv4 quantization and system co-design for efficient llm serving, 2025. URL https://arxiv.org/abs/2405.04532.

[24] Jiayi Liu, Tinghan Yang, and Jennifer Neville. Cliqueparcel: An approach for batching llm prompts that jointly optimizes efficiency and faithfulness, 2024. URL https://arxiv.org/abs/2402.14833.

[25] Hugging Face. Text generation inference. https://github.com/huggingface/text-generation-inference, 2022. Version consulted: v3.3.4.

[26] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention, 2023. URL https://arxiv.org/abs/2309.06180.

[27] Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. Orca: A distributed serving system for Transformer-Based generative models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 521–538, Carlsbad, CA, July 2022. USENIX Association. ISBN 978-1-939133-28-1. URL https://www.usenix.org/conference/osdi22/presentation/yu.

[28] Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*, 2023.

[29] Pin-Lun Hsu, Yun Dai, Vignesh Kothapalli, Qingquan Song, Shao Tang, Siyu Zhu, Steven Shimizu, Shivam Sahni, Haowen Ning, and Yanning Chen. Liger kernel: Efficient triton kernels for llm training, 2025. URL https://arxiv.org/abs/2410.10989.

[30] Dujian Ding, Ankur Mallick, Chi Wang, Robert Sim, Subhabrata Mukherjee, Victor Ruhle, Laks V. S. Lakshmanan, and Ahmed Hassan Awadallah. Hybrid llm: Cost-efficient and quality-aware query routing, 2024. URL https://arxiv.org/abs/2404.14618.

[31] Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding, 2023. URL https://arxiv.org/abs/2211.17192.

[32] Xiaoxiang Shi, Colin Cai, and Junjia Du. Proactive intra-gpu disaggregation of prefill and decode in llm serving, 2025. URL https://arxiv.org/abs/2507.06608.

[33] Baolin Li, Yankai Jiang, and Devesh Tiwari. Carbon in motion: Characterizing open-sora on the sustainability of generative ai for video generation. *ACM SIGENERGY Energy Informatics Review*, 4(5):160–165, 2024.

[34] Zangwei Zheng, Xiangyu Peng, Tianji Yang, Chenhui Shen, Shenggui Li, Hongxin Liu, Yukun Zhou, Tianyi Li, and Yang You. Open-sora: Democratizing efficient video production for all. *arXiv preprint arXiv:2412.20404*, 2024.

[35] Benoit Courty, Victor Schmidt, Sasha Luccioni, Goyal-Kamal, MarionCoutarel, Boris Feld, Jérémy Lecourt, LiamConnell, Amine Saboni, Inimaz, supatomic, Mathilde Léval, Luis Blanche, Alexis Cruveiller, ouminasara, Franklin Zhao, Aditya Joshi, Alexis Bogroff, Hugues de Lavoreille, Niko Laskaris, Edoardo Abati, Douglas Blank, Ziyao Wang, Armin Catovic, Marc Alencon, Michał Stęchły, Christian Bauer, Lucas Otávio N. de Araújo, JPW, and MinervaBooks. mlco2/codecarbon: v2.4.1, May 2024. URL https://doi.org/10.5281/zenodo.11171501.

[36] pyRAPL contributors. pyrapl: Energy profiling for cpu for python. https://github.com/powerapi-ng/pyRAPL, 2020.

[37] Ning Ding, Yulin Chen, Bokai Xu, Yujia Qin, Zhi Zheng, Shengding Hu, Zhiyuan Liu, Maosong Sun, and Bowen Zhou. Enhancing chat language models by scaling high-quality instructional conversations, 2023.

[38] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pages 38–45, 2020.

[39] Soham Poddar, Paramita Koley, Janardan Misra, Sanjay Podder, Navveen Balani, Niloy Ganguly, and Saptarshi Ghosh. Brevity is the soul of sustainability: Characterizing llm response lengths, 2025. URL https://arxiv.org/abs/2506.08686.

[40] Yue Zhang, Ming Zhang, Haipeng Yuan, Shichun Liu, Yongyao Shi, Tao Gui, Qi Zhang, and Xuanjing Huang. Llmeval: A preliminary study on how to evaluate large language models, 2023. URL https://arxiv.org/abs/2312.07398.

[41] bitsandbytes contributors. bitsandbytes: Accessible large language models via k-bit quantization for pytorch. https://github.com/bitsandbytes-foundation/bitsandbytes, 2020.

[42] Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. Llm.int8(): 8-bit matrix multiplication for transformers at scale, 2022. URL https://arxiv.org/abs/2208.07339.

[43] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms, 2023. URL https://arxiv.org/abs/2305.14314.

[44] Team Wan, Ang Wang, Baole Ai, Bin Wen, Chaojie Mao, Chen-Wei Xie, Di Chen, Feiwu Yu, Haiming Zhao, Jianxiao Yang, Jianyuan Zeng, Jiayu Wang, Jingfeng Zhang, Jingren Zhou, Jinkai Wang, Jixuan Chen, Kai Zhu, Kang Zhao, Keyu Yan, Lianghua Huang, Mengyang Feng, Ningyi Zhang, Pandeng Li, Pingyu Wu, Ruihang Chu, Ruili Feng, Shiwei Zhang, Siyang Sun, Tao Fang, Tianxing Wang, Tianyi Gui, Tingyu Weng, Tong Shen, Wei Lin, Wei Wang, Wei Wang, Wenmeng Zhou, Wente Wang, Wenting Shen, Wenyuan Yu, Xianzhong Shi, Xiaoming Huang, Xin Xu, Yan Kou, Yangyu Lv, Yifei Li, Yijing Liu, Yiming Wang, Yingya Zhang, Yitong Huang, Yong Li, You Wu, Yu Liu, Yulin Pan, Yun Zheng, Yuntao Hong, Yupeng Shi, Yutong Feng, Zeyinzi Jiang, Zhen Han, Zhi-Fan Wu, and Ziyu Liu. Wan: Open and advanced large-scale video generative models, 2025. URL https://arxiv.org/abs/2503.20314.

[45] Samuel Williams, Andrew Waterman, and David Patterson. Roofline: an insightful visual performance model for multicore architectures. *Commun. ACM*, 52(4):65–76, April 2009. ISSN 0001-0782. doi: 10.1145/1498765.1498785. URL https://doi.org/10.1145/1498765.1498785.

[46] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *CoRR*, abs/1909.08053, 2019. URL http://arxiv.org/abs/1909.08053.

[47] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, Amar Phanishayee, and Matei Zaharia. Efficient large-scale language model training on GPU clusters. *CoRR*, abs/2104.04473, 2021. URL https://arxiv.org/abs/2104.04473.

[48] Johannes Hagemann, Samuel Weinbach, Konstantin Dobler, Maximilian Schall, and Gerard de Melo. Efficient parallelization layouts for large-scale distributed model training, 2024. URL https://arxiv.org/abs/2311.05610.

[49] Ziheng Jiang, Haibin Lin, Yinmin Zhong, Qi Huang, Yangrui Chen, Zhi Zhang, Yanghua Peng, Xiang Li, Cong Xie, Shibiao Nong, Yulu Jia, Sun He, Hongmin Chen, Zhihao Bai, Qi Hou, Shipeng Yan, Ding Zhou, Yiyao Sheng, Zhuo Jiang, Haohan Xu, Haoran Wei, Zhang Zhang, Pengfei Nie, Leqi Zou, Sida Zhao, Liang Xiang, Zherui Liu, Zhe Li, Xiaoying Jia, Jianxi Ye, Xin Jin, and Xin Liu. Megascale: Scaling large language model training to more than 10,000 gpus, 2024. URL https://arxiv.org/abs/2402.15627.

[50] Jessica Pavani, Rosangela Helena Loschi, and Fernando Andres Quintana. Modeling temporal dependence in a sequence of spatial random partitions driven by spanning tree: an application to mosquito-borne diseases, 2025. URL https://arxiv.org/abs/2501.04601.

[51] Patrick von Platen, Suraj Patil, Anton Lozhkov, Pedro Cuenca, Nathan Lambert, Kashif Rasul, Mishig Davaadorj, Dhruv Nair, Sayak Paul, William Berman, Yiyi Xu, Steven Liu, and Thomas Wolf. Diffusers: State-of-the-art diffusion models. https://github.com/huggingface/diffusers, 2022.

[52] Yuwei Guo, Ceyuan Yang, Anyi Rao, Zhengyang Liang, Yaohui Wang, Yu Qiao, Maneesh Agrawala, Dahua Lin, and Bo Dai. Animatediff: Animate your personalized text-to-image diffusion models without specific tuning, 2024. URL https://arxiv.org/abs/2307.04725.

[53] Zhuoyi Yang, Jiayan Teng, Wendi Zheng, Ming Ding, Shiyu Huang, Jiazheng Xu, Yuanming Yang, Wenyi Hong, Xiaohan Zhang, Guanyu Feng, Da Yin, Yuxuan Zhang, Weihan Wang, Yean Cheng, Bin Xu, Xiaotao Gu, Yuxiao Dong, and Jie Tang. Cogvideox: Text-to-video diffusion models with an expert transformer, 2025. URL https://arxiv.org/abs/2408.06072.

[54] Yoav HaCohen, Nisan Chiprut, Benny Brazowski, Daniel Shalem, Dudu Moshe, Eitan Richardson, Eran Levin, Guy Shiran, Nir Zabari, Ori Gordon, Poriya Panet, Sapir Weissbuch, Victor Kulikov, Yaki Bitterman, Zeev Melumian, and Ofir Bibi. Ltx-video: Realtime video latent diffusion, 2024. URL https://arxiv.org/abs/2501.00103.

[55] Genmo Team. Mochi 1. https://github.com/genmoai/models, 2024.

A Theoretical Latency Analysis of LLaMA 3.1 8B (FP32) on NVIDIA H100 SXM

Modeling assumptions. In this appendix, we provide a theoretical latency analysis of LLaMA 3.1 8B executed in FP32 on an NVIDIA H100 GPU. We adopt a simplified performance model in which compute and memory operations are assumed to run asynchronously. Each operation is characterized either as compute-bound or memory-bound depending on its FLOP count F_o and data transfer volume D_o, relative to the hardware peaks $F_{\max}$ (floating-point throughput) and $B_{\max}$ (memory bandwidth). We neglect kernel launch overhead, thread block underutilization, memory misalignment, and other low-level effects, which we subsume into empirical correction factors μ_{comp} and μ_{mem} calibrated *a posteriori*. The model assumes the absence of fused kernels and relies on HBM3 bandwidth as a proxy for effective cache performance.

A.1 Hardware and Model Constants for Llama-3.1 B on NVIDIA H100

Compute time and memory time. For each operation o with F_o floating-point operations and D_o bytes of data transfer, the wall-clock execution time on the GPU is bounded by either the compute throughput or the memory bandwidth:

$$t_o^{\text{cmp}} = \frac{F_o}{F_{\text{eff}}}, \quad t_o^{\text{mem}} = \frac{D_o}{B_{\text{eff}}}, \quad t_o = \max\Big(t_o^{\text{cmp}}, t_o^{\text{mem}}\Big). \tag{1}$$

This fundamental relation underpins the latency calculations in Tables 7 and 8.

Model hyperparameters. Table 6 lists the main model-level symbols used throughout this analysis and their default values for Llama-2 8B in FP32.

A.2 Prefill Phase (Prompt Processing)

During the prefill phase, the model processes the input prompt of length s using N stacked Transformer blocks. Each block performs a sequence of elementary operations listed in Table 7, including normalization, projections, attention, and feed-forward layers. The table reports the symbolic FLOP count F_o, the data volume D_o, and the critical sequence length $s^\star$ at which the operation becomes memory-bound (see Eq. (1)).

A.2.1 Closed-Form Latency (dominant terms)

To keep the latency model interpretable and focused on the critical path, we retain only the four operations that dominate wall-clock time in practice during the prefill phase of

Table 5: Effective ceilings adopted in the FP32 analysis (H100 SXM 80 GB, ECC off, June 2025 official specs).

Quantity	Symbol	Value
Peak FP32 throughput (SM)	$F_{\max}$	6.7×10^{13} FLOP s^{-1}
Raw HBM3 bandwidth	$B_{\max}$	3.35×10^{12} B s^{-1}
Effective compute throughput	F_{eff}	$\mu_{\text{comp}} \cdot F_{\max}$
Effective memory bandwidth	B_{eff}	$\mu_{\text{mem}} \cdot B_{\max}$
Compute efficiency factor	μ_{comp}	0.7
Memory efficiency factor	μ_{mem}	0.28

Table 6: Model-level symbols and default values for LLAMA-3.1 8B (FP32).

Description	**Symbol**	**Value**
Number of Transformer blocks	N	32
Vocabulary size	V	128 256
Word size (FP32)	b_{w}	4 bytes
Prompt length	s	1–128 000
Number of generated tokens	g	1–128 000
Batch size	b	1
Query projection (Q)		
Number of heads	n_q	32
Dimension per head	d	128
Total hidden size	$h_q = n_q d$	4 096
Key/Value projections (K,V)		
Number of KV heads (GQA)	n_{kv}	8
Dimension per head	d	128
Total KV size	$h_{kv} = n_{kv} d$	1 024
Feed-forward network (FFN)		
Hidden expansion size	h_{ff}	14 336

Table 7: Operation-wise cost during prefill for LLaMA-3.1 8B (FP32). The table lists the number of FLOPs and bytes transferred per operation, along with the switching point $s^\star$ beyond which memory bandwidth becomes the limiting factor.

Operation and Formula	**FLOP** F_o	**Bytes** D_o	**Switching length** $s^\star$
Executed once per sequence			
Token + position lookup	0	$bsh_q b_\mathrm{w}$	M
Executed N times per block			
$\mathrm{LayerNorm}_1$ (RMSNorm)	$5bsh_q$	$2bsh_q b_\mathrm{w}$	M
Q projection (xW_Q, $W_Q \in \mathbb{R}^{h_q \times h_q}$)	$2bsh_q^2$	$(2bsh_q + h_q^2)b_\mathrm{w}$	$\mathrm{M} \leq 40.8 \leq C$
K projection (xW_K, $W_K \in \mathbb{R}^{h_q \times h_{kv}}$)	$2bsh_q h_{kv}$	$(bs(h_q + h_{kv}) + h_q h_{kv})b_\mathrm{w}$	$\mathrm{M} \leq 42.5 \leq C$
V projection (xW_V, $W_V \in \mathbb{R}^{h_q \times h_{kv}}$)	$2bsh_q h_{kv}$	$(bs(h_q + h_{kv}) + h_q h_{kv})b_\mathrm{w}$	$\mathrm{M} \leq 42.5 \leq C$
RoPE Q	$5bsh_q$	$2bs(h_q + d)b_\mathrm{w}$	M
RoPE K	$5bsh_{kv}$	$2bs(h_{kv} + d)b_\mathrm{w}$	M
Flash Attention ($P \cdot V$, with $P = \mathrm{softmax}(QK^\top)$)	$bs^2(4h_q + 5n_q)$	$(4bsh_q + bsh_{kv} + bn_q s^2)b_\mathrm{w}$	$\mathrm{M} \leq 99.59 \leq C$
Output proj. W_O ($xW_O \in \mathbb{R}^{h_q \times h_q}$)	$2bsh_q^2$	$(2bsh_q + h_q^2)b_\mathrm{w}$	$\mathrm{M} \leq 40.80 \leq C$
$\mathrm{Residual\ add}_1$	bsh_q	$2bsh_q b_\mathrm{w}$	M
$\mathrm{LayerNorm}_2$	$5bsh_q$	$2bsh_q b_\mathrm{w}$	M
FFN up proj. (xW_up, $W \in \mathbb{R}^{h_q \times h_\mathrm{ff}}$)	$2bsh_q h_\mathrm{ff}$	$(bs(h_q + h_\mathrm{ff}) + h_q h_\mathrm{ff})b_\mathrm{w}$	$\mathrm{M} \leq 40.51 \leq C$
SiLU activation	$4bsh_\mathrm{ff}$	$2bsh_\mathrm{ff} b_\mathrm{w}$	M
FFN gate proj. (xW_gate, $W \in \mathbb{R}^{h_q \times h_\mathrm{ff}}$)	$2bsh_q h_\mathrm{ff}$	$(bs(h_q + h_\mathrm{ff}) + h_q h_\mathrm{ff})b_\mathrm{w}$	$\mathrm{M} \leq 40.51 \leq C$
Element-wise mult (gate $\cdot$ up)	bsh_ff	$2bsh_\mathrm{ff} b_\mathrm{w}$	M
FFN down proj. (xW_down, $W \in \mathbb{R}^{h_\mathrm{ff} \times h_q}$)	$2bsh_q h_\mathrm{ff}$	$(bs(h_q + h_\mathrm{ff}) + h_q h_\mathrm{ff})b_\mathrm{w}$	$\mathrm{M} \leq 40.51 \leq C$
$\mathrm{Residual\ add}_2$	bsh_q	$2bsh_q b_\mathrm{w}$	M
After the N blocks			
Final LayerNorm	$5bsh_q$	$2bsh_q b_\mathrm{w}$	M
LM head (xW_LM, $W \in \mathbb{R}^{h_q \times V}$)	$2bh_q V$	$(bh_q + Vh_q + b)b_\mathrm{w}$	M
Softmax on logits	$5bV$	$2bVb_\mathrm{w}$	M
Top-k / Argmax / Sampling (sort, mask, sample)	$\sim 0.1bV$	bVb_w	M
Post-processing (bitwise ops, comparisons, scatter)	negligible	$\sim bVb_\mathrm{w}$	M

LLaMA-3.1:

- **Q/K/V projections:** Three matrix multiplications of shape $[bs, h_q] \cdot [h_q, h_q \text{ or } h_{kv}]$, contributing $2bs(h_q^2 + 2h_q h_{kv})$ FLOPs and significant memory bandwidth due to read/write of large activations.
- **Flash Attention:** A fused kernel computing $QK^\top$, softmax, and $P \cdot V$ in one pass, with total cost $\sim 9bs^2 h_q$ FLOPs and complex memory access patterns. This term dominates for long prompts due to quadratic scaling in s.
- **FFN up/down projections:** Two large matrix multiplications $[bs, h_q] \cdot [h_q, h_{\text{ff}}]$ and $[bs, h_{\text{ff}}] \cdot [h_{\text{ff}}, h_q]$, yielding $4bsh_q h_{\text{ff}}$ FLOPs, which are compute-bound for typical hidden sizes.
- **LM head:** The final projection of shape $[bs, h_q] \cdot [h_q, V]$ with cost $2bsh_q V$, particularly expensive for large vocabulary sizes V.

These four operations account for the majority of latency observed during inference on modern GPUs, especially for medium to large batch sizes and prompt lengths.

For each operation o we denote by t_o^{cmp} the compute-bound time and by t_o^{mem} the memory-bound time, and we write $t_o(s) = \max\Big(t_o^{\text{cmp}}(s),\, t_o^{\text{mem}}(s)\Big)$ as in Eq. (1). With the symbols introduced in Tables 5–6, the per-token latencies of the dominant operations are given by:

$$t_{\text{qkv}}(s) = 2\max\!\left(\frac{2bs\,h_q\,h_{kv}}{F_{\text{eff}}},\ \frac{(bs\,(h_q+h_{kv})+h_q\,h_{kv})\,b_{\text{w}}}{B_{\text{eff}}}\right) + \max\!\left(\frac{2bs\,h_q^2}{F_{\text{eff}}},\ \frac{(2bs\,h_q+h_q^2)\,b_{\text{w}}}{B_{\text{eff}}}\right), \tag{2}$$

$$t_{\text{flash}}(s) = \max\!\left(\frac{bs^2(4h_q+5n_q)}{F_{\text{eff}}},\ \frac{(4bsh_q+bsh_{kv}+bn_q s^2)b_{\text{w}}}{B_{\text{eff}}}\right), \tag{3}$$

$$t_{\text{ffn}}(s) = 3\max\!\left(\frac{2bs\,h_q h_{\text{ff}}}{F_{\text{eff}}},\ \frac{bs(h_q+h_{\text{ff}})+h_q h_{\text{ff}})\,b_{\text{w}}}{B_{\text{eff}}}\right), \tag{4}$$

$$t_{\text{lm}}(s) = \max\!\left(\frac{2b\,h_q V}{F_{\text{eff}}},\ \frac{(b\,h_q+Vh_q+b)\,b_{\text{w}}}{B_{\text{eff}}}\right). \tag{5}$$

Putting everything together, the closed-form prefill latency for a prompt of length s processed by N Transformer blocks is

$$\boxed{t_{\text{prefill}}(s) \;=\; N\Big[t_{\text{qkv}}(s) + t_{\text{flash}}(s) + t_{\text{ffn}}(s)\Big] \;+\; t_{\text{lm}}(s).} \tag{6}$$

All lighter operations (token lookup, normalisations, residual adds, GELU, *etc.*) have been omitted because they contribute at most a few percent to the total runtime under the hardware ceilings of Table 5.

Asymptotic regimes in s

Evaluating Eqs. (7)–(6) with the default constants from Table 6 reveals four qualitatively distinct regimes:

1. **Constant regime ($0 \le s \lesssim 100$)**
   $\boxed{t_{\text{prefill}} \approx \text{const}} = \frac{h_q(2Nh_{kv}+Nh_q+3Nh_{\text{ff}}+V)b_w}{B_{\text{eff}}} \approx 2.63 \times 10^{-2}$ s
   The memory-bound constant term, independent of s, dominates. As a result, t_{prefill} is effectively flat.

2. **Large-context regime ($s \gtrsim 100$)**

   - *Linear sub-regime* ($100 < s \ll 2{,}000$):
     $\boxed{t_{\text{prefill}} \approx \gamma s}$
     The small memory-bound terms quickly become negligible. All dominant terms are compute-bound, and latency scales linearly with s.
   - *Transition regime* ($2{,}000 \lesssim s \lesssim 30{,}000$):
     $\boxed{t_{\text{prefill}} \approx \gamma s + \eta s^2}$
     The quadratic attention term becomes significant, and the latency shows a mixed linear–quadratic behavior.
   - *Quadratic regime* ($s \gtrsim 30{,}000$):
     $\boxed{t_{\text{prefill}} \approx \eta s^2}$
     The quadratic term outweighs the linear term by a factor ≥ 10, making the total latency effectively quadratic.

With constants:

$$\gamma = \frac{bh_q(2Nh_q + 6Nh_{\text{ff}} + 4Nh_{kv})}{F_{\text{eff}}} \approx 2.40{\times}10^{-4}\,\text{s/token}, \quad \eta = \frac{4bNh_q}{F_{\text{eff}}} \approx 1.68{\times}10^{-8}\,\text{s/token}^2$$

In summary, the latency curve is:

- *constant* for very short prompts ($s \lesssim 100$),
- *linear* over most practical sequence lengths,
- and *quadratic* only in ultra-long-context scenarios, not reached with our Llama 3.1 8B.

A.3 Decode Phase (Auto-Regressive)

During the *decode* phase the model generates a sequence of g new tokens *one step at a time*. At step $t \in \{1, \dots, g\}$ the current context length is

$$\ell_t \;=\; s + t - 1,$$

i.e. the s prompt tokens plus the $t-1$ tokens already produced. All keys and values computed at previous steps are kept in an in-memory *KV-cache*. Therefore only the *query* for the current token is freshly projected, but it must attend to *all* ℓ_t cached keys and values.

A.3.1 Operation-wise cost per generated token

Table 8 lists the elementary operations executed for *one* generated token when the context length is ℓ, together with their symbolic floating-point cost F_o, the data volume D_o, and the crossover length $\ell^\star$ at which the operation switches from memory-bound to compute-bound according to Eq. (1). Except for the dot-product attention, all operations are independent of ℓ.

A.3.2 Closed-form latency for one token

With the same notation as in Sec. A.2.1, the dominant per-token latencies are

$$t_{\text{qkv}}(s) = 2\max\!\left(\frac{2b\,h_q\,h_{kv}}{F_{\text{eff}}},\ \frac{(b\,(h_q+h_{kv})+h_q\,h_{kv})\,b_{\text{w}}}{B_{\text{eff}}}\right) + \max\!\left(\frac{2b\,h_q^2}{F_{\text{eff}}},\ \frac{(2b\,h_q+h_q^2)\,b_{\text{w}}}{B_{\text{eff}}}\right), \tag{7}$$

$$t_{\text{flash}}(s) = \max\!\left(\frac{b\ell(4h_q+5n_q)}{F_{\text{eff}}},\ \frac{(4bh_q+bh_{kv}\ell+bn_q\ell)b_{\text{w}}}{B_{\text{eff}}}\right), \tag{8}$$

$$t_{\text{ffn}}(s) = 3\max\!\left(\frac{2b\,h_q h_{\text{ff}}}{F_{\text{eff}}},\ \frac{b(h_q+h_{\text{ff}})+h_q h_{\text{ff}})\,b_{\text{w}}}{B_{\text{eff}}}\right), \tag{9}$$

$$t_{\text{lm}}(s) = \max\!\left(\frac{2b\,h_q V}{F_{\text{eff}}},\ \frac{(b\,h_q+Vh_q+b)\,b_{\text{w}}}{B_{\text{eff}}}\right). \tag{10}$$

In the case of the **H100** GPU and the **Llama 3.1 8B** model (FP32), **all operations remain memory-bound** across the board.

A.3.3 End-to-end latency for g generated tokens

Let $\ell_t = s + t - 1$ be the context length at step t. The total decode latency is the sum of the g step-latencies:

$$\boxed{t_{\text{decode}}(s,g) \;=\; \sum_{\ell_t=s+1}^{s+g-1}\Big[N\Big(t_{\text{proj}} + t_{\text{ffn}} + t_{\text{attn}}(\ell_t)\Big) + t_{\text{lm}}\Big].} \tag{11}$$

Since $t_{\text{proj}}, t_{\text{ffn}}, t_{\text{lm}}$ do not depend on ℓ_t, they contribute a purely *linear* term, whereas $t_{\text{attn}}(\ell_t)$ contains all dependence on ℓ_t. We start the summation at $\ell_t = s + 1$ (i.e. $t = 2$) since the generation of the first token ($t = 1$) is typically included in the prefill latency.

$$
\begin{aligned}
t_{\text{decode}}(s, g) &= A(g-1) + B(g-1)(g+2s) \\
&= \boxed{\delta(s)g + \epsilon g^2} \quad \text{(quadratic in } g\text{)} \\
&= \boxed{\theta(g) + \lambda(g)s} \quad \text{(linear in } s\text{)} \\
A &= \frac{((12Nb + V)h_q + 2Nbh_{kv} + Nbh_{ff} + Nh_q^2 + 3Nh_q h_{ff} + 2Nh_{kv}h_q)b_w}{B_{\text{eff}}} = 6.40 \times 10^{-3} FLOPs \\
B &= \frac{Nb(n_q + h_{kv})b_w}{B_{\text{eff}}} = 3.91 \times 10^{-7} s
\end{aligned}
\tag{12}
$$

With the default parameters for **H100** and **Llama 3.1 8B**:

$$\delta(s) = (A + 2sB - B) \approx 6.40 \times 10^{-3} + 7.83 \times 10^{-7} \cdot s \ \text{ [s]}$$

$$\epsilon = B \approx 3.91 \times 10^{-7} \ \text{ [s/token}^2\text{]}$$

$$\theta(g) = (A + g)(g - 1) \approx 8.21 \times 10^{1}(g - 1) + 3.91 \times 10^{-7} g^2 \ \text{ [s]}$$

$$\lambda(g) = 2B(g - 1) \approx 7.83 \times 10^{-7}(g{-}1) \ \text{ [s/token]}$$

Asymptotic regimes in g and s

1. Fixed prompt (s), vary generation (g). The linear–quadratic knee

$$g_\star(s) = \frac{A}{B} + 2s \approx 7.93{\times}10^4 + 2s$$

The quadratic term becomes non-negligible for large generation lengths g; a longer prompt s delays the onset of this quadratic regime, but increases the overall latency.

2. Fixed generation (g), vary prompt (s). Splitting t_{decode} into a constant and an s-linear part, their crossover (the constant becoming less important than the linear term):

$$s_\star \approx \frac{A}{2B} + \frac{g}{2} \approx 3.96{\times}10^4 + 0.5g$$

For small prompt lengths, the constant term clearly dominates; for very larger prompts, the linear term becomes significant. A larger generation length g delays the appearance of this linear regime.

In summary, the decode latency behaves as:

- *linear* in both s and g across nearly all practical use cases,
- and *quadratic* in g only for extremely long generations ($g \gtrsim 10^8$), with the prompt length s further delaying this regime.

B Latency Comparison by Numerical Precision

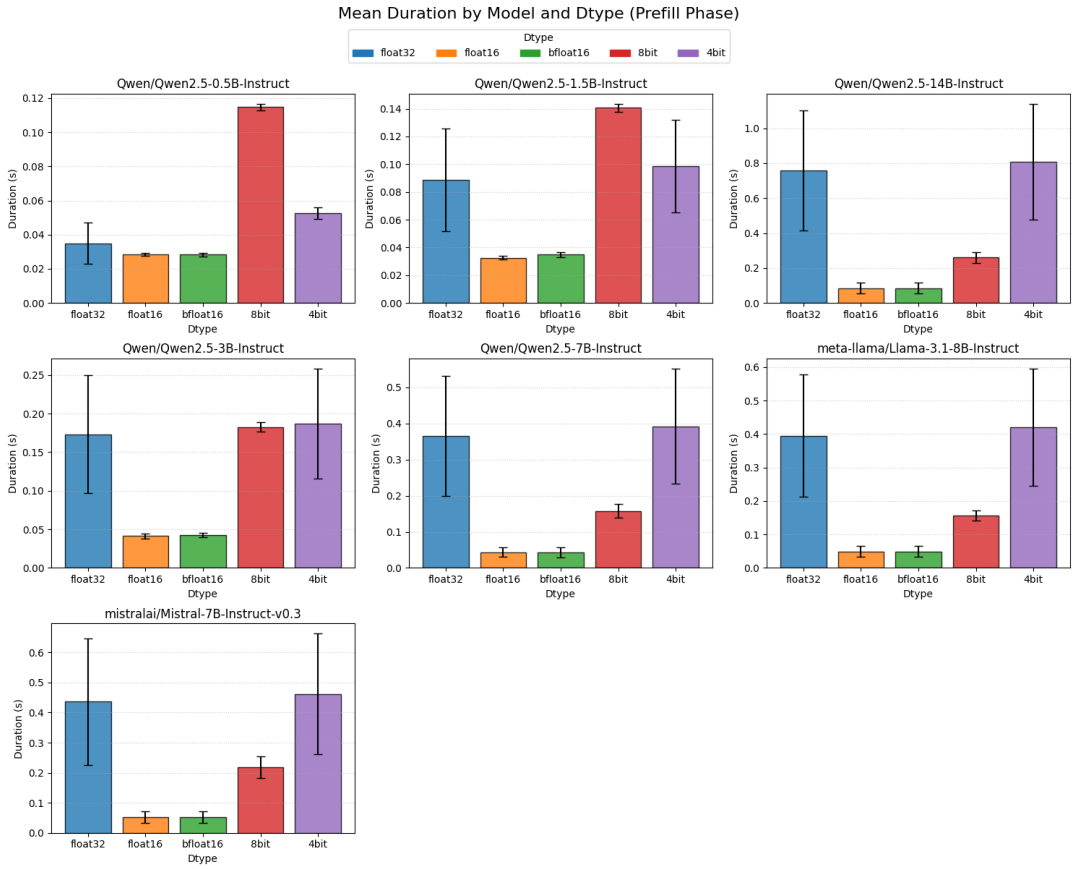


Figure 16: Mean latency per request (with variance across runs) for different models and data types during the **prefill** phase. Lower-precision formats generally reduce latency, with diminishing returns for already small models.

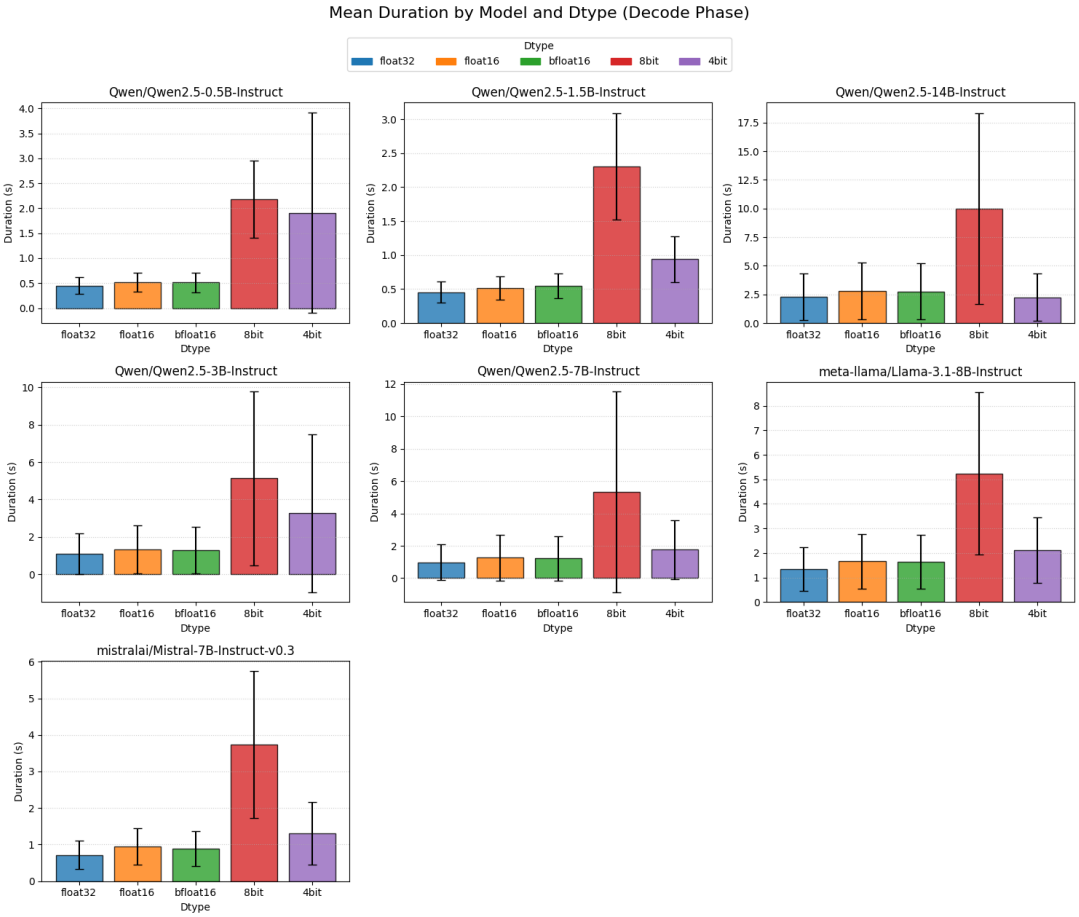


Figure 17: Mean latency per **generated token** (with variance across runs) for different models and data types during the **decode** phase. Memory-bound regimes lead to latency plateaus despite lower precision.

C Latency Comparison by Batch Size

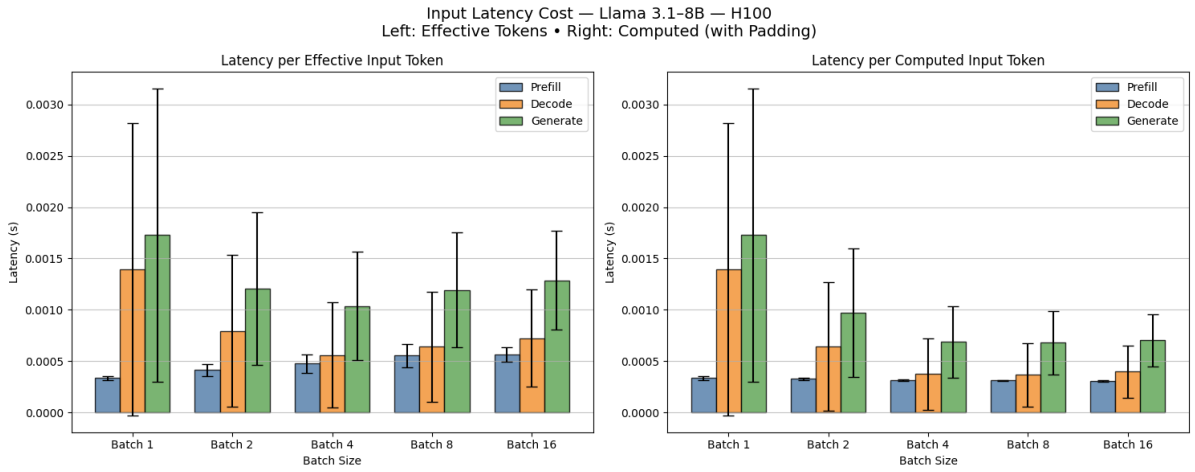


Figure 18: Latency per input token on LLaMA 3.1 8B across batch sizes. **Left:** Effective tokens only (excluding padding). **Right:** Computed tokens including padding overhead. Increasing batch size improves compute amortization but introduces padding-induced inefficiencies.

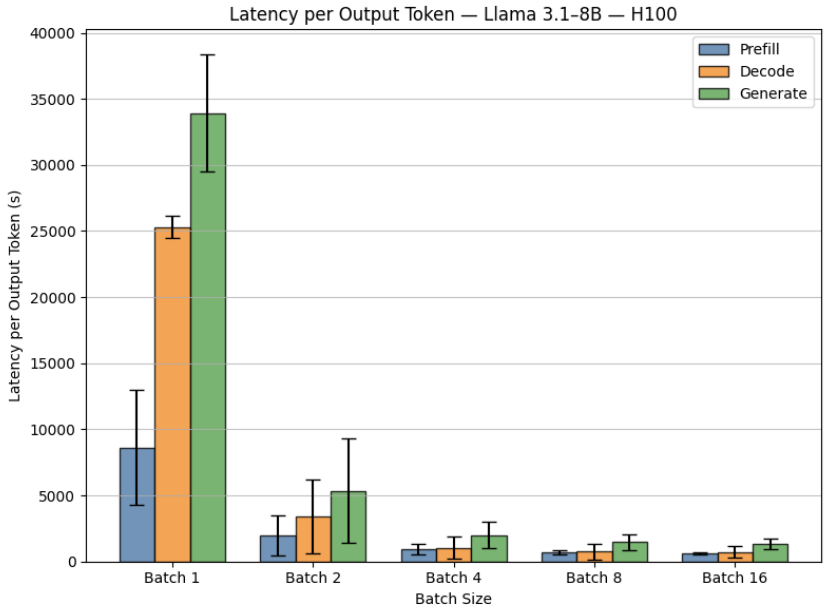


Figure 19: Latency per output token (effective = computed) across batch sizes. Gains plateau at moderate batch sizes due to limits in parallelism and autoregressive nature of decoding.

D Detailed FLOP Derivations and Scaling Laws for Wan 2.1

Conventions. We follow the HPC convention where one multiply–add equals two FLOPs. Matrix multiplications of shape $(a \times b) \cdot (b \times c)$ therefore cost $2abc$ FLOPs. Bias additions, activations, layer norms, and softmax are lower order and omitted unless stated. All results below apply per forward pass.

D.1 Latent Tokenization and Shapes

Let the video have T frames and spatial size $H \times W$ in pixels. The VAE downsamples temporally by a factor v_t and spatially by v_s, and the DiT operates on spatial patches of size $p_h \times p_w$ in the latent grid. The token length ℓ seen by the DiT is

$$\ell \;=\; \left(1 + \frac{T}{v_t}\right) \frac{H}{v_s\, p_h} \, \frac{W}{v_s\, p_w} \,. \tag{13}$$

In WAN2.1 we use $(v_t, v_s, p_h, p_w) = (4, 8, 2, 2)$, hence the shorthand $\ell = (1 + \frac{T}{4})\frac{H}{16}\frac{W}{16}$ used in the main text.

D.2 Self-Attention in the DiT

Let d be the model width and h the number of heads (with $d_h = d/h$). For a sequence of length ℓ:

$$\begin{aligned} \text{Q,K,V projections:} \quad & 3 \times 2\,\ell d^2 \;=\; 6\,\ell d^2 \\ \text{Attention logits } (QK^\top): \quad & 2\,\ell^2 d \\ \text{Weighted sum } (AV): \quad & 2\,\ell^2 d \\ \text{Output projection:} \quad & 2\,\ell d^2 \,. \end{aligned} \tag{14}$$

Summing on all N DiT layers yields

$$F_{\text{self}} \;=\; N \times (8\,\ell d^2 \;+\; 4\,\ell^2 d) \,. \tag{15}$$

(The head count h cancels out, since $h \cdot d_h = d$.)

D.3 Cross-Attention (Video → Text)

Let m be the number of text tokens and d the shared width. Assuming no KV cache (K,V recomputed each denoising step as it is done in the current official implementation) and one cross-attention block per DiT layer:

$$\begin{aligned} \text{Query from video:} \quad & 2\,\ell d^2 \\ \text{Keys/values from text:} \quad & 4\,md^2 \quad \text{(K and V)} \\ \text{Attention products:} \quad & 2\,\ell md \;+\; 2\,\ell md \;=\; 4\,\ell md \\ \text{Output projection:} \quad & 2\,\ell d^2 \,. \end{aligned} \tag{16}$$

Hence over the N layers

$$F_{\text{cross}} \;=\; N \times (4\,\ell d^2 \;+\; 4\,md^2 \;+\; 4\,\ell md)\,. \tag{17}$$

With KV caching, the $4md^2$ term becomes once-per-video while the $4\ell md$ products remain per step. *With windowed or factorized attention*, ℓ or m may be replaced by the effective window size.

D.4 Transformer MLP

With expansion factor f and sequence length ℓ, a two-layer MLP $d \to fd \to d$ costs over all DiT layers

$$F_{\text{mlp}} \;=\; N \times 4f\,\ell d^2\,. \tag{18}$$

D.5 Stacking Across S Steps, and CFG

Let g denote the number of conditional forward passes (CGF) per denoising step ($g = 2$ under classifier-free guidance). Combining (15)–(18), the DiT cost is

$$F_{\text{DiT}}(T, H, W; S, N, d, f, m, g) \;=\; g\,S \;\times \Big(F_{\text{self}} + F_{\text{cross}} + F_{\text{mlp}}\Big)\,, \tag{19}$$

with ℓ given by (13).

D.6 Text Encoder

For a L_{text}-layer encoder (e.g., T5/CLIP-like) with width d_{text}, expansion f_{text}, and m tokens:

$$\begin{aligned} \text{Self-attn per layer:} &\quad 8\,md_{\text{text}}^2 + 4\,m^2 d_{\text{text}} \\ \text{FFN per layer:} &\quad 4f_{\text{text}}\,md_{\text{text}}^2\,. \end{aligned} \tag{20}$$

For p_{text} forward passes per video (e.g., $p_{\text{text}} = 2$ for conditional and unconditional prompts),

$$F_{\text{text}} \;=\; p_{\text{text}}\,L_{\text{text}}\Big(8\,md_{\text{text}}^2 + 4\,m^2 d_{\text{text}} + 4f_{\text{text}}\,md_{\text{text}}^2\Big). \tag{21}$$

This term is once-per-video, independent of S.

D.7 Timestep Embedding MLP

Mapping a scalar diffusion step to a d-dim vector and injecting it into each block via a small MLP with hidden width d_τ:

$$F_\tau \;=\; gS(2\,d_\tau\,d \;+\; 14\,d^2). \tag{22}$$

D.8 VAE: Convolutions and Middle Attention

We account for the VAE cost as the sum of (i) all convolutional layers along the decoder and (ii) a 2D self-attention "middle" block evaluated independently per time slice.

Convolutional layers. For a 3D convolution with kernel $(k_t^{(j)}, k_h^{(j)}, k_w^{(j)})$, channels $C_{\text{in}}^{(j)} \to C_{\text{out}}^{(j)}$ and output size $T^{(j)} \times H^{(j)} \times W^{(j)}$, the cost is

$$F_{\text{conv3d}}^{(j)} \;=\; 2\,k_t^{(j)} k_h^{(j)} k_w^{(j)}\, C_{\text{in}}^{(j)} C_{\text{out}}^{(j)}\, T^{(j)} H^{(j)} W^{(j)}\,. \tag{23}$$

Summing over the decoder path gives $F_{\text{VAE,conv}} = \sum_{j=1}^{N_{\text{dec,conv}}} F_{\text{conv3d}}^{(j)}$, with concrete per-layer shapes provided in Table 10. WAN-2.1 VAE include a 2D self-attention middle block evaluated independently on each time slice ($L_* = H_* W_*$, channel width C_*):

$$F_{\text{VAE,mid-attn}} \;=\; T_*\Big(8\,C_*^2 L_* \;+\; 4\,L_*^2 C_*\Big). \tag{24}$$

Middle self-attention (2D, per time slice). Let C_* be the channel width at the middle resolution, and T_*, H_*, W_* the temporal/spatial sizes (thus $L_* = H_* W_*$ tokens per time slice). Using the derivation in Appendix D.2, the middle attention cost is

$$F_{\text{VAE,mid-attn}} \;=\; T_*\Big(8\,C_*^2 L_* \;+\; 4\,L_*^2 C_*\Big), \tag{25}$$

where the final $2C_*^2 L_*$ term arises from the output projection and is included in the $8C_*^2 L_*$ term above.

WAN2.1 decoder instantiation (values). In WAN2.1, the VAE decoder starts from a latent grid $(T_0, H_0, W_0) = \big(\lceil T/4 \rceil,\, H/8,\, W/8\big)$ with z=16 channels. A causal 3×3×3 convolution expands this to 384 channels, followed by a "middle" block consisting of two residual 3×3×3 convolutions and a 2D self-attention layer applied independently per time slice. The decoder then progressively upsamples: two *temporal+spatial* upsamplings (doubling T, H, W and halving channels), followed by one purely *spatial* upsampling (doubling H, W and halving channels). Residual blocks (three per stage) refine features at each resolution, and a final 3×3×3 convolution produces the RGB output at (T, H, W).

Table 10 summarizes the dominant operators for FLOP accounting. Applying Eq. (23) across these layers yields $F_{\text{VAE,conv}}$, while Eq. (25) gives the middle-attention cost.

D.9 Total FLOPs and Leading-Order Scaling

We finally obtain

$$F_{\text{total}}(H, W, T, S) \;=\; F_{\text{text}} \;+\; F_{\text{VAE,conv}} \;+\; F_{\text{VAE,mid-attn}} \;+\; F_{\tau} \;+\; F_{\text{DiT}}\,, \tag{26}$$

with components given by (21), (23), (25), (22), and (19). Since ℓ grows linearly with H, W, and T (Eq. 13), the $\ell^2 d$ and $\ell m d$ terms in F_{DiT} dominate for typical settings ($\ell \gg m$), yielding quadratic growth in H, W, and T, and linear growth in S.

Scope and caveats. (i) FlashAttention and fused kernels reduce memory traffic and constants but do not change FLOP counts. (ii) KV caching changes only the cross-attention $4md^2$ term from per-step to once-per-video. (iii) Windowed or factorized attention replaces ℓ (or m) by an effective window size, altering quadratic scaling. (iv) If activations or norms become bandwidth-bound, the proportionality between FLOPs and latency weakens; our WAN2.1 measurements on H100 indicated compute-bound behavior over the operating points considered.

Table 8: Operation-wise cost during decode for LLAMA-3.1 8B (FP32). FLOP/byte counts are for a *single* generated token; ℓ is the current context length ($\ell = s + t - 1$).

Operation and Formula	F_o **(FLOP)**	D_o **(bytes)**	$\ell^\star$
Executed once per generated token			
Token + position lookup	0	$bh_q b_\mathrm{w}$	memory-bound
Executed N times per block			
$\mathrm{LayerNorm}_1$	$5bh_q$	$2bh_q b_\mathrm{w}$	memory-bound
Q projection (xW_Q)	$2bh_q^2$	$(2bh_q + h_q^2)b_\mathrm{w}$	memory-bound
K projection (xW_K)	$2bh_q h_{kv}$	$(b(h_q + h_{kv}) + h_q h_{kv})b_\mathrm{w}$	memory-bound
V projection (xW_V)	$2bh_q h_{kv}$	$(b(h_q + h_{kv}) + h_q h_{kv})b_\mathrm{w}$	memory-bound
RoPEQ	$5bh_q$	$2b(h_q + d)b_\mathrm{w}$	memory-bound
RoPEK	$5bh_{kv}$	$5b(h_{kv} + d)b_\mathrm{w}$	memory-bound
Flash Attention ($P = \mathrm{softmax}(QK^\top), P \cdot V$)	$b\ell(4h_q + 5n_q)$	$(4bh_q + b\ell h_{kv} + bn_q\ell)b_\mathrm{w}$	compute $\leq \ell^\star \leq$ memory
Output proj. W_O	$2bh_q^2$	$(2bh_q + h_q^2)b_\mathrm{w}$	memory-bound
Residual add_1	bh_q	$2bh_q b_\mathrm{w}$	memory-bound
$\mathrm{LayerNorm}_2$	$5bh_q$	$2bh_q b_\mathrm{w}$	memory-bound
FFN up proj. (xW_up)	$2bh_q h_\mathrm{ff}$	$(b(h_q + h_\mathrm{ff}) + h_q h_\mathrm{ff})b_\mathrm{w}$	memory-bound
SiLU activation	$4bh_\mathrm{ff}$	$2bh_\mathrm{ff} b_\mathrm{w}$	memory-bound
FFN gate proj. (xW_gate)	$2bh_q h_\mathrm{ff}$	$(b(h_q + h_\mathrm{ff}) + h_q h_\mathrm{ff})b_\mathrm{w}$	memory-bound
Element-wise mult (gate $\cdot$ up)	bh_ff	$2bh_\mathrm{ff} b_\mathrm{w}$	memory-bound
FFN down proj. (xW_down)	$2bh_q h_\mathrm{ff}$	$(b(h_q + h_\mathrm{ff}) + h_q h_\mathrm{ff})b_\mathrm{w}$	memory-bound
Residual add_2	bh_q	$2bh_q b_\mathrm{w}$	memory-bound
After the N blocks			
Final LayerNorm	$5bh_q$	$4bh_q b_\mathrm{w}$	memory-bound
LM head (xW_LM)	$2bh_q V$	$(bh_q + Vh_q + b)b_\mathrm{w}$	memory-bound
Softmax on logits	$5bV$	$2bVb_\mathrm{w}$	memory-bound
Top-k / Argmax / Sampling	$\sim 0.1bV$	bVb_w	memory-bound
Post-processing (bitwise ops, comparisons, scatter)	negligible	$\sim bVb_\mathrm{w}$	memory-bound

Table 9: Complete set of WAN2.1-T2V-1.3B hyperparameters and constants. This table provides the full notation, including VAE layer-wise symbols (instantiated explicitly in Appendix D.8).

Symbol	Value	Meaning
	Global video parameters	
T	variable	Number of frames
$H \times W$	variable	Input spatial resolution
S	variable	Number of denoising steps
g	2	CFG passes per step (cond + uncond)
v_t, v_s	$4, 8$	Temporal and spatial downsampling factors of the VAE
p_h, p_w	$2, 2$	Spatial patch size in the DiT latent grid
	Diffusion Transformer (DiT)	
N	32	Number of DiT layers
d	2048	Hidden size
f	4	MLP expansion factor ($8192 = 4d$)
ℓ	$(1 + \frac{T}{4})\frac{H}{16}\frac{W}{16}$	Token length of latent grid
	Text encoder (T5-XXL)	
m	512	Output tokens per video (conditioning length)
p_{text}	2	Calls per video (cond + uncond)
d_{text}	4096	Hidden size
L_{text}	24	Encoder layers
f_{text}	2.5	MLP expansion factor
	Timestep embedding	
d_τ	256	Hidden width of timestep MLP
	VAE (layer-wise; values in App. D.8)	
j	$1, \ldots, N_{\text{dec,conv}}$	Layer index along the VAE **decoder** path
$N_{\text{dec,conv}}$	11	Number of 3D conv layers in the VAE decoder
$k_t^{(j)}, k_h^{(j)}, k_w^{(j)}$	–	3D kernel sizes of decoder layer j
$C_{\text{in}}^{(j)}, C_{\text{out}}^{(j)}$	–	In/out channels at decoder layer j
$T^{(j)}, H^{(j)}, W^{(j)}$	–	Output grid sizes at decoder layer j
C_*	384	Channel width at middle attention block
T_*, H_*, W_*	$\lceil T/4 \rceil, H/8, W/8$	Grid sizes at middle resolution
L_*	$H_* W_*$	Spatial token length per frame (2D middle attention)
	Hardware / efficiency constants	
μ	0.456	Empirical efficiency (fraction of Θ_{peak})
Θ_{peak}	$989{\times}10^{15}$ FLOP/s	Peak GPU throughput (H100)
$P_{\max}$	700 W	Sustained GPU power
D_{total}	$F_{\text{total}}/(\mu\Theta_{\text{peak}})$	Total latency

Table 10: VAE decoder: representative dominant operators for FLOP accounting (layer j). It mirrors the encoder; z=16, C_*=384, middle resolution $(\lceil T/4 \rceil, H/8, W/8)$.

Stage j	Op type	Kernel (k_t, k_h, k_w)	$C_{\text{in}}^{(l)} \to C_{\text{out}}^{(l)}$	$T^{(l)}$	$H^{(l)}$	$W^{(l)}$
D0	conv3d	$(3,3,3)$	$z \to 384$	$\lceil T/4 \rceil$	$H/8$	$W/8$
Middle (RBs)	conv3d	$(3,3,3)$	$384 \to 384$	$\lceil T/4 \rceil$	$H/8$	$W/8$
Middle (attn 2D)	attn-2D	–	$384 \to 384$	$\lceil T/4 \rceil$	$H/8$	$W/8$
D1 (RBs)	conv3d	$(3,3,3)$	$384 \to 384$	$\lceil T/4 \rceil$	$H/8$	$W/8$
Up (time)	conv3d (time)	$(3,1,1)$	$384 \to 2\times384$	$\lceil T/2 \rceil$	$H/8$	$W/8$
Up (space)	conv2d (space)	$(1,3,3)$	$384 \to 192$	$\lceil T/2 \rceil$	$H/4$	$W/4$
D2 (RBs)	conv3d	$(3,3,3)$	$192 \to 384$	$\lceil T/2 \rceil$	$H/4$	$W/4$
Up (time)	conv3d (time)	$(3,1,1)$	$384 \to 2\times384$	T	$H/4$	$W/4$
Up (space)	conv2d (space)	$(1,3,3)$	$384 \to 192$	T	$H/2$	$W/2$
D3 (RBs)	conv3d	$(3,3,3)$	$192 \to 192$	T	$H/2$	$W/2$
Up (space)	conv2d (space)	$(1,3,3)$	$192 \to 96$	T	H	W
Head	conv3d	$(3,3,3)$	$96 \to 3$	T	H	W