Energy Scaling Laws for Diffusion Models: Quantifying Compute and Carbon Emissions in Image Generation

Aniketh Iyengar[1] **Jiaqi Han**[1] **Boris Ruf**[2*]
Vincent Grari[2] **Marcin Detyniecki**[2] **Stefano Ermon**[1]
[1]Stanford University [2]AXA AI Research

Abstract

The rapidly growing computational demands of diffusion models for image generation have raised significant concerns about energy consumption and environmental impact. While existing approaches to energy optimization focus on architectural improvements or hardware acceleration, there is a lack of principled methods to predict energy consumption across different model configurations and hardware setups. We propose an adaptation of Kaplan scaling laws to predict GPU energy consumption for diffusion models based on computational complexity (FLOPs). Our approach decomposes diffusion model inference into text encoding, iterative denoising, and decoding components, with the hypothesis that denoising operations dominate energy consumption due to their repeated execution across multiple inference steps. We conduct comprehensive experiments across four state-of-the-art diffusion models (Stable Diffusion 2, Stable Diffusion 3.5, Flux, and Qwen) on three GPU architectures (NVIDIA A100, A4000, A6000), spanning various inference configurations including resolution (256^2-1024^2), precision (fp16/fp32), step counts (10-50), and classifier-free guidance settings. Our energy scaling law achieves high predictive accuracy within individual architectures ($R^2 > 0.9$) and exhibits strong cross-architecture generalization, maintaining high rank correlations across models and enabling reliable energy estimation for unseen model–hardware combinations. These results validate the compute-bound nature of diffusion inference and provide a foundation for sustainable AI deployment planning and carbon footprint estimation.

1 Introduction

The rapid advancement of generative AI, particularly diffusion models [30, 9, 28] for high-quality image synthesis, has transformed creative applications and scientific visualization. Models like Stable Diffusion [25], DALL-E [24], and recent innovations such as Flux [5] and Gemini 2.5 Flash Image "Nano Banana" [7] can generate photorealistic images from text prompts with unprecedented quality. However, this capability comes at substantial computational cost—generating a single high-resolution image can require billions of floating-point operations across dozens of denoising steps.

As these models are deployed at scale, their energy consumption has become a critical concern. Recent studies estimate that training large language models can consume as much electricity as hundreds of homes over several months [31], and inference costs can be equally significant when scaled to billions of users. For diffusion models, the energy challenge is compounded by their iterative nature—unlike single forward-pass models, diffusion models require multiple denoising steps, each involving full network evaluation [9, 29].

*Corresponding author: boris.ruf@axa.com

Preprint.

Despite the critical importance of energy efficiency, current approaches to estimating diffusion model energy consumption rely primarily on empirical measurements or crude approximations. The research landscape presents significant gaps: limited research on diffusion models compared to language models [31, 17], lack of predictive models that enable planning rather than reactive measurement, limited generalization across model-hardware combinations, absence of theoretical frameworks relating computational complexity to energy usage, and hardware heterogeneity that complicates cross-platform prediction.

This work addresses these challenges by developing the first principled approach to predicting diffusion model energy consumption based on computational complexity. Our key contributions are: (1) adaptation of Kaplan scaling laws [13] to formulate energy scaling relationships for diffusion models, (2) comprehensive FLOP decomposition for four major diffusion architectures, (3) cross-architecture validation demonstrating generalization across different model families, (4) hardware-aware modeling incorporating precision and GPU architecture effects, and (5) practical framework for sustainable AI deployment planning.

2 Related Work

Energy Efficiency in Deep Learning. The environmental impact of machine learning has gained increasing attention. Strubell et al. [31] demonstrated that training large transformer models can produce as much CO_2 as several cars over their lifetimes. Henderson et al. [8] established frameworks for systematic carbon footprint reporting. Tools like CarbonTracker [2] and CodeCarbon [3] provide energy monitoring, though they focus on measurement rather than prediction. Recent work by Luccioni et al. [17, 18] analyzed energy consumption for large models but remains reactive rather than predictive.

Scaling Laws for Neural Networks. Kaplan et al. [13] established foundational scaling laws for transformer-based language models, demonstrating that performance scales predictably with parameters, dataset size, and compute budget. The Chinchilla paper [11] refined these laws for optimal resource allocation. However, existing scaling laws focus on model performance rather than inference energy consumption, and primarily address auto-regressive language models rather than diffusion-based generative models.

Diffusion Model Efficiency. Computational efficiency research has focused on reducing latency rather than energy consumption. DDIM [9] introduced deterministic sampling to reduce denoising steps. Classifier-free guidance [10] improved quality but doubled computational requirements. Latent diffusion models [25] reduce cost by operating in compressed spaces, but energy analysis has been limited to crude FLOP counting. Progressive distillation [27, 20] and few-step samplers [15, 16, 33] represent promising efficiency directions, but energy benefits vary across hardware platforms.

Hardware-Aware Machine Learning. GPU architecture significantly impacts energy efficiency. Mixed-precision training [21] can reduce energy consumption while maintaining quality. Tensor core acceleration provides efficiency improvements that depend on workload characteristics [19, 12]. Our scaling law approach enables energy prediction across hardware platforms without exhaustive measurement.

3 Background and Theoretical Foundation

3.1 Diffusion Model Architecture

Diffusion models generate images through a learned denoising process that iteratively refines random noise into coherent images. Modern text-to-image diffusion models typically consist of three main components: text encoding, iterative denoising, and image decoding.

Text Encoding. Input text prompts are processed by transformer-based encoders (e.g., CLIP [22], T5 [23]) to produce contextual embeddings that guide the generation process. This step occurs once per inference and typically represents a small fraction of total computation.

Iterative Denoising. The core computational process involves repeatedly applying a neural network to progressively denoise a latent representation. While many methods employ a U-Net [26] or standard Transformer backbone [32], state-of-the-art diffusion models typically use variants of the

MMDiT architecture [6]. Each denoising iteration requires a full forward pass through the network, with typical inference comprising 10–50 such steps. When classifier-free guidance is applied, the computational cost effectively doubles—whether implemented as an expanded batch dimension or two sequential forward passes—since both conditional and unconditional evaluations are required [2].

Image Decoding. The final denoised latent representation is decoded into pixel space using a learned decoder network (e.g., VAE decoder). Like text encoding, this occurs once per inference.

The iterative nature of denoising makes it the dominant computational component, typically accounting for more than 90% of inference FLOPs, as shown by our decomposition analysis in Fig. 8.

3.2 Kaplan Scaling Laws

The original Kaplan scaling laws [13] demonstrate that model performance scales predictably with computational resources according to power-law relationships: $L \propto N^{-\alpha_N}$, $L \propto C^{-\alpha_C}$, and $L \propto D^{-\alpha_D}$, where L denotes loss, N is the number of parameters, C represents compute (in FLOPs), and D is the dataset size. The scaling exponents α typically fall within the range 0.05–0.095. These relationships provide a principled framework for resource allocation in large-scale language model development.

Our work adapts this methodology to energy consumption, hypothesizing that energy usage follows power-law relationships with computational complexity: $E \propto \text{FLOPs}^{\alpha}$, albeit with different exponents and additional hardware-dependent factors. Kaplan et al. also introduced an analytical approximation for the FLOPs required per forward pass in transformer-based architectures, which we adopt in our analysis.

3.3 GPU Energy Consumption

GPU energy consumption depends on multiple factors including computational workload, memory access patterns, and hardware utilization. Modern GPUs are designed for high throughput parallel computation, with energy efficiency varying significantly based on workload characteristics.

Computational vs. Memory Bound. Operations that fully utilize compute units (e.g., matrix multiplications and convolutions) generally achieve higher energy efficiency than memory-bound operations such as element-wise computations or data transfers. Diffusion model inference is predominantly compute-bound, driven by large matrix operations in attention and convolution layers. Nonetheless, optimizing low-level kernel implementations to improve memory access patterns and reduce data movement can further enhance inference speed and lower overall energy consumption.

Precision Effects. Lower precision arithmetic (fp16, bfloat16) can reduce both computation time and energy consumption. Modern GPUs include specialized tensor cores that provide significant efficiency improvements for mixed-precision workloads, though the benefits depend on specific model architectures and implementations.

Hardware Architecture. Different GPU architectures exhibit varying energy efficiency characteristics. Newer architectures generally provide better performance-per-watt ratios, but absolute energy consumption depends on workload size and utilization patterns.

Understanding these factors is crucial for developing accurate energy prediction models that generalize across different hardware platforms and computational workloads.

3.4 FLOP Calculation for Diffusion Models

For diffusion models, we decompose the total computational cost as:

$$\text{FLOPs}_{\text{total}} = \text{FLOPs}_{\text{text}} + N_{\text{steps}} \times \text{FLOPs}_{\text{denoise}} + \text{FLOPs}_{\text{decode}} \tag{1}$$

This decomposition mirrors the structure of diffusion inference: text encoding and image decoding occur once per prompt, while denoising is repeated over N_{steps} iterations. Accordingly, we scale $\text{FLOPs}_{\text{denoise}}$ by N_{steps} to capture the dominant iterative cost.

[2] Some models implement amortized classifier-free guidance using a learned token. Here, we instead consider the true classifier-free guidance case from a computational footprint perspective.

Since the majority of FLOPs arise from denoising, the total compute for a denoising module with N non-embedding parameters, attention dimension d_{attn}, n_{layers} layers, and sequence length L can be approximated using the Kaplan formulation [13]:

$$\text{FLOPs}_{\text{denoise}} \approx 2N + 2Ld_{\text{attn}}n_{\text{layers}} \tag{2}$$

For diffusion models using U-Net architectures, FLOP calculation requires detailed analysis of convolution operations, attention mechanisms, and skip connections. We develop model-specific FLOP estimation functions that account for these architectural differences while maintaining the theoretical foundation established by Kaplan et al.[13]. **Given our findings in Figure 8, which highlights that the denoising FLOP count dominates the total in all models, we directly approximate** $\text{FLOPs}_{\text{total}} \approx N_{steps}N_{prompts}\text{FLOPs}_{\text{denoise}}$ **in all of our below experiments.**

4 Methodology

4.1 Energy Scaling Law Formulation

We propose an energy scaling law inspired by Kaplan's approach, relating energy consumption E to computational complexity (FLOPs) and key hardware and model configuration factors. Classifier-free guidance, floating-point precision, GPU architecture, and image resolution are incorporated as additive terms in a log-linear regression model, with FLOPs serving as the primary predictor of compute-bound energy usage.

Taking the logarithm yields a linear relationship suitable for regression:

$$\begin{aligned}\log(E) = \log(A) + \alpha \log(\text{FLOPs} \times 2^{\mathbb{I}_{\text{cfg}}}) \\ + \beta_{\text{dtype}}\mathbb{I}_{\text{dtype}} + \beta_{\text{gpu}}\mathbb{I}_{\text{gpu}} + \beta_{\text{res}} \log\left(\frac{H \times W}{256}\right)\end{aligned} \tag{3}$$

where E is the total energy consumption across the N prompts, $\mathbb{I}_{\text{cfg}}$ is the classifier-free guidance indicator, $\mathbb{I}_{\text{dtype}}, \mathbb{I}_{\text{gpu}}$ are hardware configuration indicators, H, W are image height and width, and $A, \alpha, \beta_{\text{dtype}}, \beta_{\text{gpu}}, \beta_{\text{res}}$ are learned parameters.

4.2 Feature Engineering

To implement our energy scaling law in practice, we express it in the log-linear form of Equation 3, which allows direct estimation through linear regression. The feature vector below represents the input to our linear regression model, where each component corresponds to a term in Equation 3. Hardware and configuration indicators are encoded as one-hot variables, transforming categorical attributes (GPU type, precision) into numerical features suitable for regression. The feature vector consists of:

4.3 Feature Engineering

To implement our energy scaling law in practice, we construct a feature vector $\mathbf{x}$ that serves as the input to our linear regression model. Each component in $\mathbf{x}$ corresponds directly to a term in Equation 3, enabling direct parameter estimation through ordinary least squares. Hardware and configuration indicators are encoded as one-hot variables, transforming categorical attributes (GPU type, precision) into numerical features suitable for regression. The feature vector consists of:

$$\mathbf{x} = [1, \log(\text{FLOPs}_{\text{cfg}}), \mathbb{I}_{\text{fp32}}, \mathbb{I}_{\text{A4000}}, \mathbb{I}_{\text{A6000}}, \log(H \times W/256)]^T \tag{4}$$

Here, the intercept term (1) captures base energy consumption; $\text{FLOPs}_{\text{cfg}} = \text{FLOPs} \times 2^{\mathbb{I}_{\text{cfg}}}$ accounts for the doubling of denoising FLOPs when classifier-free guidance is enabled; $\mathbb{I}_{\text{fp32}}$ is a binary indicator for 32-bit precision (with 16-bit as the baseline); $\mathbb{I}_{\text{A4000}}$ and $\mathbb{I}_{\text{A6000}}$ are one-hot indicators for GPU architecture (with A100 as the baseline); and the resolution bias term, $\log(H \times W/256)$, accounts

for efficiency variations beyond pure FLOP scaling. In implementation, we define $\mathbb{I}_{\text{dtype}} := \mathbb{I}_{\text{fp32}}$ and $\mathbb{I}_{\text{gpu}} := [\mathbb{I}_{\text{A4000}}, \mathbb{I}_{\text{A6000}}]$ to match the theoretical formulation in Equation 3.

The resolution bias term accounts for hardware-specific efficiency effects across tensor sizes. We hypothesize that energy efficiency varies with resolution due to differences in memory bandwidth utilization, cache behavior, and kernel-level optimizations not reflected in FLOP counts alone.

4.4 Experimental Design

Our experimental methodology follows rigorous practices to ensure reliable, generalizable results:

Energy Measurement. We use CodeCarbon's EmissionsTracker [3] to monitor GPU power consumption at 1Hz sampling rate throughout inference. CodeCarbon internally uses NVIDIA Management Library (NVML) to query GPU power consumption. Total energy is computed as the integral of power consumption over time, with baseline idle power subtracted to isolate inference-specific consumption.

Statistical Validation. Each experimental configuration is run with a fixed random seed for reproducibility. We assess model performance using statistical measures including R^2, mean absolute error (MAE), and Spearman rank & Pearson correlation coefficients to evaluate scaling law accuracy and generalization capability.

Cross-Validation Strategy. We employ two complementary validation approaches to assess generalization:

1. **Within-Architecture:** We apply 2-fold cross-validation on data from a single model-GPU combination (e.g., Flux on A100). This assesses scaling law stability when training and testing conditions are matched. Results are reported in Figure 1.
2. **Cross-Architecture:** We train on one or more model-GPU combinations and test on held-out models or hardware platforms. For example, training on Flux+SD3.5 data and testing on Qwen evaluates cross-model generalization (Figure 3), while training on A100 data and testing on A6000 evaluates cross-GPU generalization (Figure 2). This strategy validates that our scaling laws capture architecture-agnostic energy-complexity relationships.

Hardware Configuration. Experiments are conducted using CUDA device isolation with individual GPU assignment per experiment to minimize resource contention. Energy measurements are performed during dedicated inference runs with consistent GPU synchronization to ensure power measurement accuracy.

5 Experimental Setup

5.1 Model Selection and FLOP Estimation

We evaluate four representative diffusion models spanning distinct architectural families and compute FLOPs using model-specific formulas derived from architectural analysis (see Tables 1, 2, Figure 7).

Stable Diffusion 3.5-Large (SD3.5): 8B-parameter MMDiT with 38 layers and dual CLIP+T5 text encoders. FLOPs reflect quadratic attention scaling and joint text–image token processing. CFG is applied via a 2× batch expansion.

Flux.1 [dev]: 12B-parameter hybrid MMDiT (19 layers + 38 single transformer) trained with rectified flow rather than standard diffusion. FLOP estimation adapts transformer attention for flow-matching dynamics. CFG incurs a 2× full model pass.

Qwen-Image: 20B-parameter, 60-layer MMDiT with 16×16 visual patches and T5-based text conditioning. FLOP calculation follows the same MMDiT formulation used for Flux and SD3.5. CFG is implemented as a 2× forward pass.

Stable Diffusion 2 (SD2): 865M-parameter U-Net baseline. FLOPs are decomposed across ResNet blocks, cross-attention layers, and skip connections. CFG is applied via batch-level duplication.

This selection spans 865M–20B parameters, convolutional vs. transformer architectures, diffusion vs. flow-matching objectives, and fixed- vs. variable-length text conditioning — enabling a robust evaluation of scaling law generalization across paradigms.

5.2 Hardware Configuration

Our experiments are conducted on three NVIDIA GPU architectures representing different performance and efficiency characteristics:

NVIDIA A100: A data center GPU with 80 GB of HBM2e memory, 6,912 CUDA cores, and 432 third-generation Tensor Cores. The A100 delivers state-of-the-art mixed-precision performance and serves as our primary experimental platform. Its peak theoretical throughput is 312 TFLOPS (bfloat16 / Tensor Core precision).

NVIDIA RTX A4000: A professional workstation GPU with 16GB GDDR6 memory, 6144 CUDA cores, and 192 Tensor cores. This GPU represents mid-range deployment scenarios common in professional and edge applications. Peak theoretical performance: 19.2 TFLOPS (fp32).

NVIDIA RTX A6000 ADA: A high-end workstation GPU with 48 GB of GDDR6 ECC memory, 18,176 CUDA cores, and 568 Tensor Cores. This represents premium workstation deployment with substantial memory capacity and cutting-edge Ada Lovelace architecture. Peak theoretical performance: 91.1 TFLOPS (fp32).

5.3 Hyperparameter Space

Our experiments span a comprehensive hyperparameter space designed to capture the full range of practical deployment scenarios:

- **Inference steps**: $\{10, 20, 30, 40, 50\}$ - covering fast sampling to high-quality generation.
- **Image resolutions**: $\{256^2, 512^2, 768^2, 1024^2\}$ - from preview to high-resolution output.
- **Precision**: $\{\text{float16}, \text{float32}\}$ - comparing memory and energy trade-offs.
- **Total queries**: $\{25, 50, 100\}$ prompts - varying iterative inference throughput settings.
- **Classifier-free guidance**: $\{\text{enabled}, \text{disabled}\}$ - quality vs. efficiency trade-off.

This design yields 240 potential experimental configurations per model-GPU combination. However, practical limitations constrain our data collection: complete experimental sweeps are only achieved for A100 hardware, representing our most reliable platform. For other configurations (A4000/A6000 GPUs), we collect representative samples with varying levels of coverage across models. Additionally, Qwen's large memory requirements (7B parameters) prevent fp32 inference on our hardware, limiting this model to fp16 precision experiments. We prioritize data collection to ensure adequate coverage for cross-platform scaling validation while acknowledging these experimental constraints.

5.4 Dataset and Energy Units

We utilize a random subset of the COCO 2017 dataset [14] for our experiments. Energy consumption is recorded in kilowatt-hours (kWh) using our tracker. For consistency in regression analyses, we model the natural logarithm of total energy spent to generate the N prompts, in units $\ln(\text{kWh})$. To convert to joules, we apply the relation $J = \exp(\ln \text{kWh}) \times 3.6 \times 10^6$.

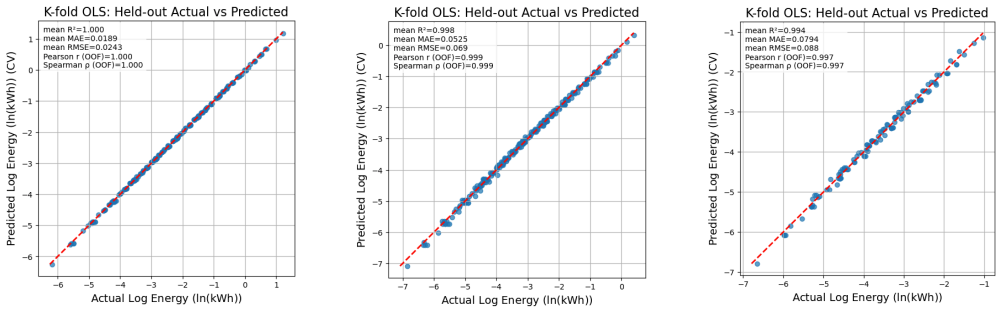
6 Results and Discussion

We highlight our main results in this section, while providing additional contextualization of the predicted energy values and the contributions of individual hyperparameters in Appendix A.4.

6.1 Individual Model Energy Scaling

Figure 1 presents energy scaling validation for individual diffusion models on NVIDIA A100 hardware. All three models demonstrate strong adherence to our scaling law formulation with $R^2 > 0.92$, with learned parameters summarized in Figure 1d.

Flux achieves $R^2 = 1.0$ with scaling exponent $\alpha = 0.989$, very close to the theoretical compute-bound ideal of 1.0. The precision coefficient $\beta_{\text{dtype}} = 2.04$ indicates approximately e^2 energy increase for fp32 vs fp16 inference. We can see that our energy range generally falls in $8.9 \times 10^3 - 9.8 \times 10^6 J$.



(a) Flux (b) Stable Diffusion 3.5 (c) Qwen

	Flux	**SD 3.5**	**Qwen**
$\log(A)$	-20.61	-19.95	-18.64
α	0.989	0.969	0.992
β_{dtype}	2.04	1.90	0
β_{res}	-0.043	-0.054	-0.306

(d) Learned parameters

Figure 1: Individual model energy scaling validation on NVIDIA A100 GPU. Diagnostic plots showing actual versus predicted energy consumption for (a) Flux, (b) Stable Diffusion 3.5, and (c) Qwen diffusion models. (d) Shows the learned scaling parameters with exponents α approaching the theoretical compute-bound ideal of 1.0. All models exhibit near-linear FLOP-energy relationships, confirming the compute-bound nature of diffusion inference. Note: β_{gpu1}=β_{gpu2}=0 since only A100 data is used; Qwen β_{dtype}=0 due to fp16-only training.

Stable Diffusion 3.5 shows similar performance with $R^2 = 0.998$ and $\alpha = 0.969$, demonstrating consistent scaling behavior despite different architectural details. We can see that our energy range generally falls in $3.3 \times 10^3 - 9.8 \times 10^6 J$.

Qwen presents $R^2 = 0.994$ and $\alpha = 0.992$, however heavily using the β_{res} bias, potentially reflecting memory bandwidth limitations in its 60-layer architecture. We can see that our energy range generally falls in $3.3 \times 10^3 - 1.32 \times 10^6 J$, but only for float16 inferences.

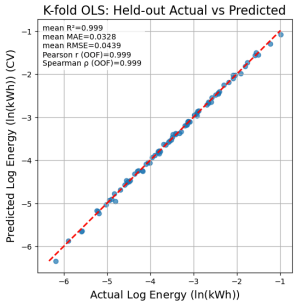
As a case study, Table 6 reports energy usage for Qwen across hyperparameter configurations on an A100 GPU in the 100-prompt setting. Consumption spans three orders of magnitude—from 1.83×10^4 J (0.051 Wh) per image under a minimal configuration (10 steps, 256^2, fp16, no CFG) to 1.29×10^6 J (3.58 Wh) for high-quality generation (50 steps, 1024^2, fp16, CFG). These values exceed typical large language model inference costs: a single diffusion image can consume up to $10\times$ the energy of an average ChatGPT query (0.34 Wh) [1] or median Gemini request (0.24 Wh) [4].

Furthermore, the negative resolution-bias coefficient ($\beta_{\text{res}} = -0.028$ to -0.206) indicates that this bias diminishes as resolution increases. This suggests that GPU utilization becomes more efficient at larger tensor sizes—or, conversely, that fixed-overhead operations constitute a relatively larger share of the energy cost at lower resolutions.

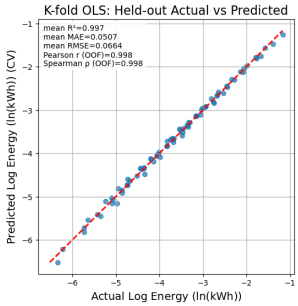
6.2 Cross-GPU Hardware Validation

Figure 2 demonstrates scaling law robustness across GPU architectures. The fundamental FLOPs scaling exponents remain stable ($\alpha = 0.997, 0.989$), while GPU-specific coefficients capture hardware differences: A6000 shows energy overhead ($\beta_{\text{gpu2}} = 0.450, 0.308$) compared to A100 baseline. Qwen is excluded from cross-GPU validation due to memory constraints on the A6000 platform.

This hardware consistency validates our approach for deployment planning—organizations can predict energy consumption across different GPU platforms without extensive empirical testing for each configuration.



(a) Flux



(b) Stable Diffusion 3.5

	Flux	**SD 3.5**
$\log(A)$	-20.85	-20.44
α	0.997	0.989
$\beta_{\textbf{gpu2}}$	0.450	0.308
$\beta_{\textbf{res}}$	-0.027	-0.037

(c) Learned parameters

Figure 2: Diagnostic plots show actual versus predicted energy consumption for (a) Flux and (b) Stable Diffusion 3.5 models, using data from NVIDIA A100 and A6000 GPUs. Panel (c) presents the learned scaling parameters, highlighting stable exponents (α) across GPUs and GPU-specific coefficients. Note: The Flux plot includes no-CFG, float16, and non-50-prompt runs, while the SD 3.5 plot includes only CFG, float16, and non-50-prompt runs.

6.3 Cross-Model Generalization

The most significant finding is demonstrated in Figure 3: scaling laws learned from certain diffusion models successfully predict energy consumption for entirely different diffusion models. This cross-model universality suggests that energy scaling laws capture fundamental computational principles rather than model-specific optimizations, enabling transferability from open-source models to proprietary closed systems, with profound implications for sustainable AI development.

Three cross-validation scenarios were used to assess generalization: (a) Qwen + SD 3.5 $\rightarrow$ Flux, (b) Flux + SD 3.5 $\rightarrow$ Qwen, and (c) Flux + Qwen $\rightarrow$ SD 3.5. Strong agreement between training and testing performance indicates that energy efficiency is governed primarily by computational complexity rather than architectural hyperparameters. The slight deviation observed in the Qwen cases stems from not fitting the β_{res} parameter. We find that the off-diagonal points correspond to underestimated FLOPs for 256×256 images. At this smaller image resolution, we hypothesize hat FLOPs alone no longer significantly dominate the full energy costs, relative to memory operations and bandwidth constraints. Consequently, the bounding characteristics of FLOPs become insufficient to explain the observed energy profile in large models. The β_{res} parameter likely adjusts for this when fitted on the Qwen data.

6.4 Cross-Architecture (& GPU) Validation

We further validate the transferability of our scaling laws across fundamentally different architectural paradigms—convolutional U-Nets (Stable Diffusion 2) and transformer-based MMDiT models. Evaluations conducted on A100 and A6000 show that training on MMDiT models (Flux, SD 3.5, Qwen) and testing on the U-Net architecture achieves robust relative ranking performance ($R, R_s > 0.9$), although decrease in precision.

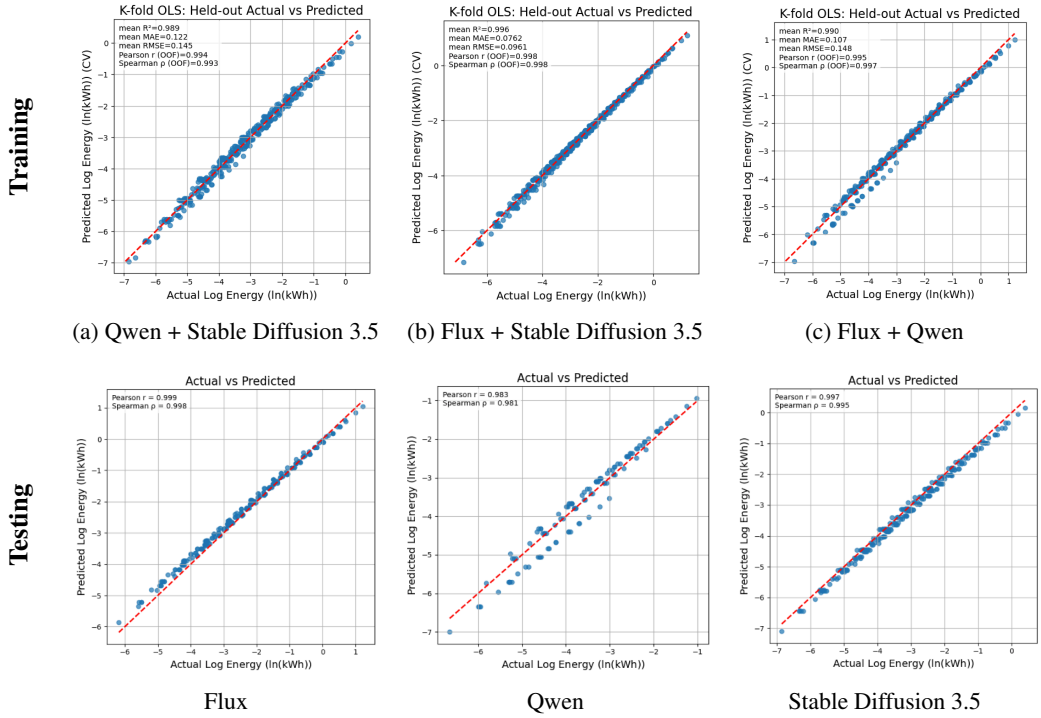
This cross-architecture generalization demonstrates that FLOP-based scaling laws capture computational universals rather than architecture-specific optimizations, enabling unified energy prediction across diverse model families from convolutional to transformer designs (detailed results in Appendix A.1).

6.5 Practical Implications and Applications

Our findings support several immediate applications for sustainable AI deployment:

Deployment and Hardware Planning. Near-linear FLOPs scaling ($\alpha \approx 1$) enables accurate pre-deployment energy estimates, allowing practitioners to compare GPU configurations, balance cost–efficiency trade-offs, and select deployment sites to minimize environmental impact.

Carbon-Aware and Algorithmic Optimization. Consistent hyperparameter effects allow systems to dynamically adjust precision, resolution, or step count based on grid carbon intensity, while our



(a) Qwen + Stable Diffusion 3.5 (b) Flux + Stable Diffusion 3.5 (c) Flux + Qwen

Flux Qwen Stable Diffusion 3.5

Figure 3: The top row shows training on model pairs: (a) Qwen + SD 3.5, (b) Flux + SD 3.5, and (c) Flux + Qwen. The bottom row presents the corresponding tests on the held-out models: Flux, Qwen, and SD 3.5, respectively. Consistent diagnostic patterns across all training–test pairs demonstrate the robustness of our FLOP-based scaling methodology for cross-model energy prediction. All plots here use results from the NVIDIA A100, which yielded the most comprehensive hyperparameter search.

framework also quantifies the sustainability gains from algorithmic improvements such as reduced inference steps or more efficient sampling.

Standardization and Reporting. The proposed methodology offers a standardized approach for estimating and reporting energy consumption, supporting regulatory compliance and reproducible energy accounting across models and hardware platforms.

6.6 Limitations and Future Directions

While our results show strong predictive performance, several limitations remain:

Model and Hardware Scope. We evaluate four models and three NVIDIA GPU types. New diffusion architectures and alternative hardware (e.g., AMD, specialized accelerators) may exhibit distinct scaling, though cross-architecture validation indicates reasonable robustness.

Dynamic Effects. GPU frequency scaling and thermal throttling may affect real-world energy use beyond what static FLOP-based models capture, despite our controlled experimental setup.

Pipeline Coverage. Our analysis isolates the core model inference cost and does not account for auxiliary pre/post-processing or multi-model orchestration, which are typically minor contributors to overall energy. In real-world deployment, inference is also often served as part of a pipeline of user queries, which we approximate via iterative prompt evaluation, though this is not executed within a fully productionized serving stack.

Future work will extend our analysis to video diffusion models and those with larger scale, real-time carbon-aware optimization, and validation on broader hardware including emerging AI accelerators.

7 Conclusion

This work presents the first principled framework for predicting diffusion model energy consumption from computational complexity. By extending Kaplan-style scaling laws to energy prediction, we establish both theoretical foundations and practical tools for sustainable AI deployment.

Our main contributions are: (1) a theoretical formulation of energy scaling laws incorporating computational, hardware, and resolution effects; (2) comprehensive validation across four model architectures and three GPU platforms; (3) demonstration of cross-architecture generalization enabling prediction for unseen model–hardware pairs; and (4) practical applications for deployment planning and carbon accounting. We find that diffusion model energy consumption can span three orders of magnitude (e.g., for Qwen: 0.051–3.58 Wh per image), with a single high-quality image consuming up to 10× the energy of a typical large language model query—underscoring the need for energy-aware deployment strategies.

Observed near-linear scaling confirms that diffusion inference is largely compute-bound, while robust cross-architecture transfer shows that these laws capture fundamental, architecture-agnostic principles. The framework achieves strong predictive accuracy ($R^2 > 0.9$ within-architecture, ($R, R_s > 0.9$) cross-architecture) and yields interpretable insights into hardware efficiency.

Beyond research, our approach supports sustainable AI practices through deployment planning, carbon-aware optimization, and standardized energy reporting. More broadly, it illustrates how scaling-law methodologies can generalize across models and hardware to guide systematic environmental optimization in machine learning.

References

[1] Sam Altman. The Gentle Singularity. `https://blog.samaltman.com/the-gentle-singularity`. [Accessed 18-10-2025].

[2] Lasse F Wolff Anthony, Benjamin Kanding, and Raghavendra Selvan. Carbontracker: Tracking and predicting the carbon footprint of training deep learning models. *arXiv preprint arXiv:2007.03051*, 2020.

[3] 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.

[4] Cooper Elsworth, Keguo Huang, David Patterson, Ian Schneider, Robert Sedivy, Savannah Goodman, Ben Townsend, Parthasarathy Ranganathan, Jeff Dean, Amin Vahdat, et al. Measuring the environmental impact of delivering AI at Google Scale. *arXiv preprint arXiv:2508.15734*, 2025.

[5] Patrick Esser, Sumith Kulal, Andreas Blattmann, Rahim Entezari, Jonas Müller, Harry Saini, Yam Levi, Dominik Lorenz, Axel Sauer, Frederic Boesel, et al. Scaling rectified flow transformers for high-resolution image synthesis. *arXiv preprint arXiv:2403.03206*, 2024.

[6] Patrick Esser, Sumith Kulal, Andreas Blattmann, Rahim Entezari, Jonas Müller, Harry Saini, Yam Levi, Dominik Lorenz, Axel Sauer, Frederic Boesel, Dustin Podell, Tim Dockhorn, Zion English, Kyle Lacey, Alex Goodwin, Yannik Marek, and Robin Rombach. Scaling rectified flow transformers for high-resolution image synthesis, 2024.

[7] Alisa Fortin, Guillaume Vernade, Kat Kampf, and Ammaar Reshi. Introducing gemini 2.5 flash image, our state-of-the-art image model. Google Developers Blog, August 2025.

[8] 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. *Journal of Machine Learning Research*, 21(248):1–43, 2020.

[9] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020.

[10] Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. *arXiv preprint arXiv:2207.12598*, 2022.

[11] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Oriol Vinyals, Jack W. Rae, and Laurent Sifre. Training compute-optimal large language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2022. Curran Associates Inc.

[12] Zhe Jia, Marco Maggioni, Benjamin Staiger, and Daniele Paolo Scarpazza. Dissecting the NVIDIA volta GPU architecture via microbenchmarking. *arXiv preprint arXiv:1804.06826*, 2018.

[13] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

[14] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2015.

[15] Cheng Lu, Yuhao Zhou, Fan Bao, Jianfei Chen, Chongxuan Li, and Jun Zhu. Dpm-solver: A fast ode solver for diffusion probabilistic model sampling in around 10 steps. *Advances in Neural Information Processing Systems*, 35:5775–5787, 2022.

[16] Cheng Lu, Yuhao Zhou, Fan Bao, Jianfei Chen, Chongxuan Li, and Jun Zhu. Dpm-solver++: Fast solver for guided sampling of diffusion probabilistic models. *Machine Intelligence Research*, pages 1–22, 2025.

[17] Alexandra Sasha Luccioni, Sylvain Viguier, and Anne-Laure Ligozat. Estimating the carbon footprint of bloom, a 176b parameter language model. *arXiv preprint arXiv:2211.02001*, 2022.

[18] Alexandra Sasha Luccioni, Sylvain Viguier, and Anne-Laure Ligozat. Power hungry processing: Watts driving the cost of AI deployment? *FAccT '23: Proceedings of the 2023 ACM Conference on Fairness, Accountability, and Transparency*, pages 85–99, 2023.

[19] Stefano Markidis, Steven Wei Der Chien, Erwin Laure, Ivy Bo Peng, and Jeffrey S Vetter. NVIDIA tensor core programmability, performance & precision. In *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 522–531. IEEE, 2018.

[20] Chenlin Meng, Robin Rombach, Ruiqi Gao, Diederik Kingma, Stefano Ermon, Jonathan Ho, and Tim Salimans. On distillation of guided diffusion models. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14297–14306, 2023.

[21] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, et al. Mixed precision training. *arXiv preprint arXiv:1710.03740*, 2017.

[22] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision, 2021.

[23] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer, 2023.

[24] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. In *International conference on machine learning*, pages 8821–8831. Pmlr, 2021.

[25] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10684–10695, 2022.

[26] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.

[27] Tim Salimans and Jonathan Ho. Progressive distillation for fast sampling of diffusion models. *arXiv preprint arXiv:2202.00512*, 2022.

[28] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International conference on machine learning*, pages 2256–2265. PMLR, 2015.

[29] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502*, 2020.

[30] Yang et al. Song. Score-based generative modeling through stochastic differential equations. *arXiv preprint arXiv:2011.13456*, 2020.

[31] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in NLP. *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3645–3650, 2019.

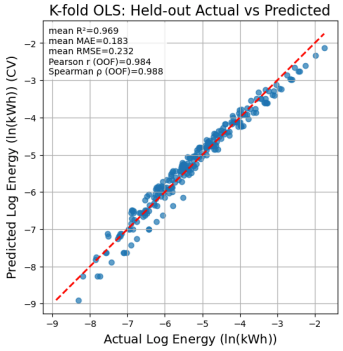
[32] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

[33] Qinsheng Zhang, Molei Tao, and Yongxin Chen. gddim: Generalized denoising diffusion implicit models. *arXiv preprint arXiv:2206.05564*, 2022.

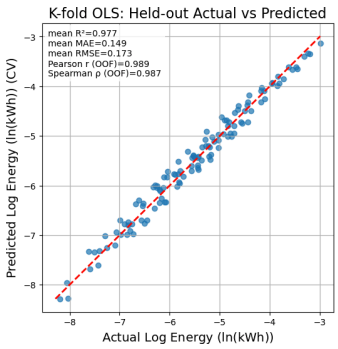
A Additional Validation Results

A.1 Individual U-Net Architecture Validation

Figure 4 provides detailed individual model validation for Stable Diffusion 2's U-Net architecture across different GPU platforms. Despite fundamental architectural differences from transformer-based models, SD2 demonstrates robust scaling behavior consistent with MMDiT architectures, with $R^2 > 0.9$ for both single-GPU and cross-GPU validation scenarios. See 7 for the SD2 FLOP details.



(a) Stable Diffusion 2 (NVIDIA A100)



(b) Stable Diffusion 2 (cross-GPU)

Figure 4: Diagnostic plots show (a) Stable Diffusion 2 on NVIDIA A100, illustrating U-Net scaling behavior, and (b) cross-GPU validation across A100, A4000, and A6000 platforms. Consistent scaling patterns confirm that our FLOP-based energy prediction generalizes beyond transformer-based models to convolutional architectures. Note: Cross-GPU results include CFG, float16, and non-50-prompt runs.

A.2 U-Net to Transformer Generalization

Figure 5 shows that cross-architecture experiments accurately predict U-Net energy consumption using scaling laws learned from transformer-based MMDiT models, and vice versa. Figure 6 further demonstrates this generalization across A100 and A6000 GPUs.

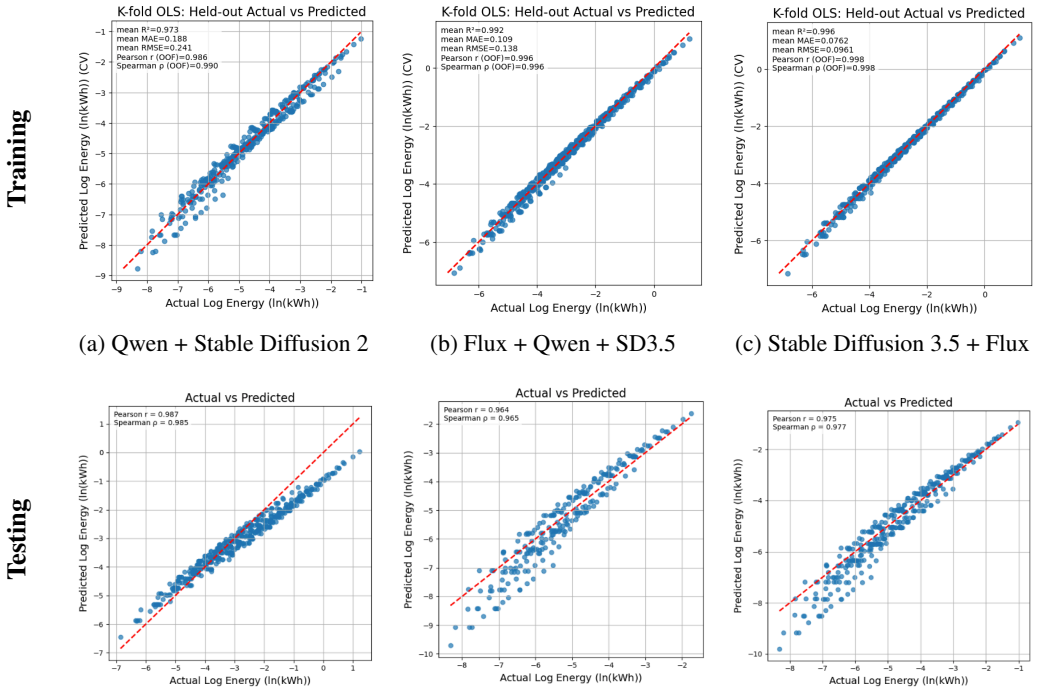
Despite the architectural gulf between U-Net and transformer designs, our scaling laws capture fundamental energy-complexity relationships independent of specific architectural paradigms, successfully bridging traditional convolutional U-Net designs and modern transformer-based MMDiT approaches. This indicates broad applicability across diffusion model paradigms and suggests our methodology could extend to other generative model families.

A.3 FLOPs Computations

Tables 1, 2, 7 detail how the FLOPs are computed per model class. Again, while we detail the computation of the encoder and decoder flops, this is largely not used in our analysis due to the bounding behavior of the denoising computation.

A.4 Hyperparameter Energy Contributions

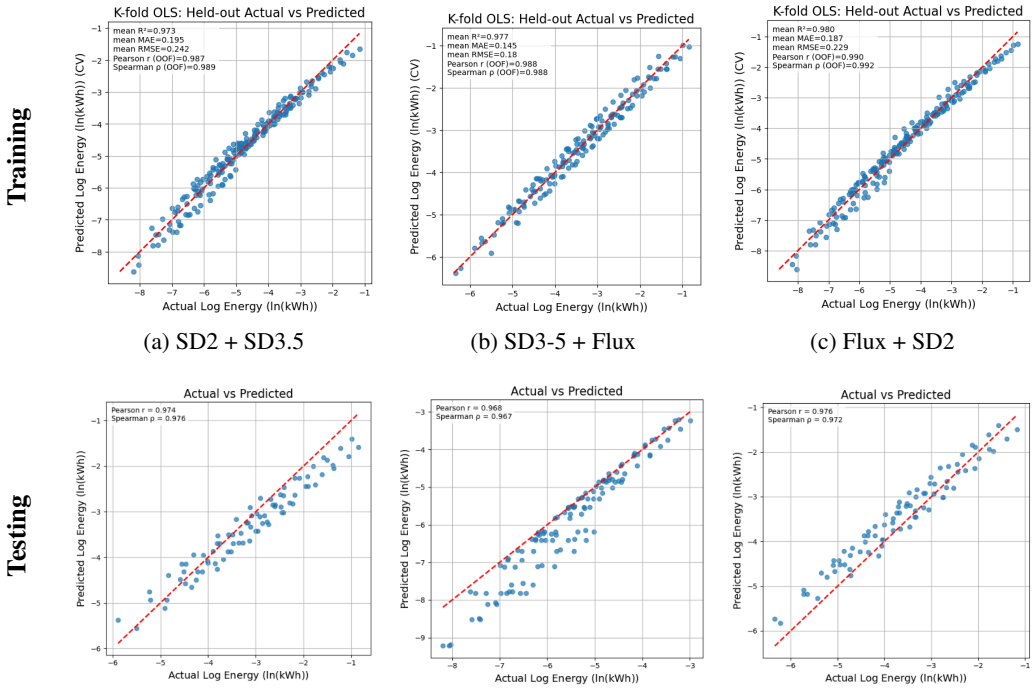
In Figures 9, 10, 11, and 12, we compare how energy consumption scales across models as a function of key hyperparameters, normalized to the lowest-energy configuration in the 100-prompt A100 setting: 256×256 resolution, float16 precision, no CFG, and 10 diffusion steps. The highest-energy setting corresponds to 1024×1024 resolution, float32 precision, CFG enabled, and 50 steps. Across all models, float32 incurs significantly higher energy cost than float16, and energy increases sharply with image resolution — especially when paired with more denoising steps. These observations mirror recent efforts toward step-efficient and distillation-based generation strategies to reduce inference cost at scale. Exact A100 energy values for this setting are reported in Tables 3, 4, 5, and 6. Other GPU results can be found in our code.



(a) Qwen + Stable Diffusion 2 (b) Flux + Qwen + SD3.5 (c) Stable Diffusion 3.5 + Flux

Flux + Stable Diffusion 3.5 Stable Diffusion 2 Stable Diffusion 2 + Qwen

Figure 5: Cross-architecture experiments demonstrate generalization between U-Net and MMDiT architectures on the A100. Top row shows training: (a) Qwen+SD2 (MMDiT+U-Net), (b) Flux+Qwen+SD3.5 (all MMDiT), (c) SD3.5+Flux (both MMDiT). Bottom row shows corresponding testing: Flux+SD3.5 (MMDiT), SD2 (U-Net), SD2+Qwen (U-Net+MMDiT). The consistent scaling patterns validate that our FLOP-based methodology captures fundamental energy-complexity relationships independent of specific architectural paradigms, successfully bridging traditional convolutional U-Net designs and modern transformer-based MMDiT approaches.



(a) SD2 + SD3.5 (b) SD3-5 + Flux (c) Flux + SD2

Flux Stable Diffusion 2 Stable Diffusion 3.5

Figure 6: Cross-architecture+gpu experiments demonstrate generalization between U-Net and MMDiT architectures across the A100 and A6000. Top row shows training: (a) SD3.5+SD2 (MMDiT+U-Net), (b) SD3.5+FLUX (all MMDiT), (c) Flux+SD2 (MMDiT+U-Net). Bottom row shows corresponding testing: Flux (MMDiT), SD2 (U-Net), SD3.5 (MMDiT). The consistent scaling patterns validate that our FLOP-based methodology captures fundamental energy-complexity relationships independent of specific architectural paradigms and across GPU types. Note: This evaluation uses the cross-GPU hyperparameter subsets we have noted throughout the paper.

Table 1: Summary of FLOP formulas used for different model components. Multiply–add pairs count as 2 FLOPs. Totals are for a single forward pass.

Function	Formula
Convolution	$\text{FLOPs}_{\text{conv}}(\text{kernel} = k, C_{\text{in}}, C_{\text{out}}, H, W) = 2HWk^2 C_{\text{in}} C_{\text{out}}$
Transformer	$\text{FLOPs}_{\text{Tr}}(d_{\text{model}}, d_{\text{ff}}, d_{\text{attn}}, n_{\text{layer}}, n_{\text{ctx}})$ $N = 2d_{\text{model}} n_{\text{layer}}(2d_{\text{attn}} + d_{\text{ff}}), \quad c_{\text{fwd}} = 2N + 2n_{\text{layer}} n_{\text{ctx}} d_{\text{attn}}$ $\text{FLOPs}_{\text{Tr}} = n_{\text{ctx}} \cdot c_{\text{fwd}}$
MMDiT	$\text{FLOPs}_{\text{MMDiT}}(d_{\text{model}}, d_{\text{ff}}, d_{\text{attn}}, n_{\text{layer}}, n_{\text{ctx}})$ $N = 4d_{\text{model}} n_{\text{layer}}(2d_{\text{attn}} + d_{\text{ff}}), \quad c_{\text{fwd}} = N + 2n_{\text{layer}} n_{\text{ctx}} d_{\text{attn}}$ $\text{FLOPs}_{\text{MMDiT}} = n_{\text{ctx}} \cdot c_{\text{fwd}}$
Cross-Attn Tr.	$\text{FLOPs}_{\text{CrossTr}}(d_q, d_k, d_{\text{ff}}, d_{\text{attn}}, n_{\text{layer}}, n_q, n_k)$ $N = 2n_{\text{layer}}[d_q(d_{\text{attn}} + d_{\text{ff}}) + d_k d_{\text{attn}}], \quad c_{\text{fwd}} = 2N + 2n_{\text{layer}} n_k d_{\text{attn}}$ $\text{FLOPs}_{\text{CrossTr}} = n_q \cdot c_{\text{fwd}}$
ResNetBlock	$\text{FLOPs}_{\text{Res}}(\text{kernel} = k, C_{\text{in}}, C_{\text{out}}, H, W)$ $= \text{FLOPs}_{\text{conv}}(k, C_{\text{in}}, C_{\text{out}}, H, W) + \text{FLOPs}_{\text{conv}}(k, C_{\text{out}}, C_{\text{out}}, H, W)$
Decoder	$\text{FLOPs}_{\text{Dec}}(H, W)$ where $H_0 = H/8$, $W_0 = W/8$ $= \text{FLOPs}_{\text{conv}}(k{=}3, C_{\text{in}}{=}16, C_{\text{out}}{=}512, H_0, W_0) + 2\text{FLOPs}_{\text{Res}}(k{=}3, 512, 512, H_0, W_0)$ $+ \text{FLOPs}_{\text{Tr}}(d_{\text{model}}{=}512, d_{\text{ff}}{=}256, d_{\text{attn}}{=}512, n_{\text{layer}}{=}1, n_{\text{ctx}}{=}H_0 W_0)$ $+ 3\text{FLOPs}_{\text{Res}}(3, 512, 512, H_0, W_0) + \text{FLOPs}_{\text{conv}}(3, 512, 512, 2H_0, 2W_0)$ $+ 3\text{FLOPs}_{\text{Res}}(3, 512, 512, 2H_0, 2W_0) + \text{FLOPs}_{\text{conv}}(3, 512, 512, 4H_0, 4W_0)$ $+ \text{FLOPs}_{\text{Res}}(3, 512, 256, 4H_0, 4W_0) + 2\text{FLOPs}_{\text{Res}}(3, 256, 256, 4H_0, 4W_0)$ $+ \text{FLOPs}_{\text{conv}}(3, 256, 256, 8H_0, 8W_0) + \text{FLOPs}_{\text{Res}}(3, 256, 128, 8H_0, 8W_0)$ $+ 2\text{FLOPs}_{\text{Res}}(3, 128, 128, 8H_0, 8W_0) + \text{FLOPs}_{\text{conv}}(3, 128, 3, 8H_0, 8W_0)$

Table 2: Per-model FLOP composition (single forward pass). Multiply–add pairs count as 2 FLOPs. Totals are expressed in GFLOPs as defined in Table 1. Model-specific latent and embedding dimensions are substituted directly into the general formulas.

Model	Context / Geometry	FLOP formula (GFLOPs)
FLUX	$n_{\text{ctx}} = \frac{HW}{16^2} + 512$	Latent patch tokens ($H/16 \times W/16$) plus 512 fixed text tokens. Hidden width d_{model}=3072, d_{ff}=12288, d_{attn}=3072. Uses 19 MMDiT layers and 38 transformer layers. Text encoders: OpenAI CLIP and T5. The T5 bias term accounts for GLU activation blocks. *Core (diffusion):* $\text{GFLOPs}_{\text{MMDiT}}(3072, 12288, 3072, 19, n_{\text{ctx}})$ + $\text{GFLOPs}_{\text{Tr}}(3072, 12288, 3072, 38, n_{\text{ctx}})$ *Text / embedding:* $\text{GFLOPs}_{\text{Tr}}(768, 3072, 768, 12, 77)$ + $\text{GFLOPs}_{\text{Tr}}(4096, 10240, 4096, 24, 512) + 24{\cdot}10240{\cdot}4097/10^9$ *Decoder:* $\text{GFLOPs}_{\text{Dec}}(H, W)$
Qwen (image)	$n_{\text{ctx}} = \frac{HW}{16^2} + 12$	Image latents ($H/16 \times W/16$) plus variable text tokens (averaged to 12 from dataset). Hidden width d_{model}=3072, d_{ff}=12288, d_{attn}=3072, n_{layer}=60. Uses Qwen2.5-VL encoder with MQA (multi-query attention) correction. *Core (diffusion):* $\text{GFLOPs}_{\text{MMDiT}}(3072, 12288, 3072, 60, n_{\text{ctx}})$ *Text / embedding:* $\text{GFLOPs}_{\text{Tr}}(3584, 18944, 3584, 28, 12) + 28{\cdot}12{\cdot}2(2{\cdot}3584(512 - 3584))/10^9$ *Decoder:* $\text{GFLOPs}_{\text{Dec}}(H, W)$
SD 3.5	$n_{\text{ctx}} = \frac{HW}{16^2} + 333$	Latent grid ($H/16 \times W/16$) plus 333 fixed conditioning tokens. Hidden width d_{model}=2432, d_{ff}=9478, d_{attn}=2432, n_{layer}=38. Uses three text encoders: two CLIP variants and T5. *Core (diffusion):* $\text{GFLOPs}_{\text{MMDiT}}(2432, 9478, 2432, 38, n_{\text{ctx}})$ *Text / embedding:* $\text{GFLOPs}_{\text{Tr}}(768, 3072, 768, 12, 77)$ + $\text{GFLOPs}_{\text{Tr}}(1280, 5120, 1280, 32, 77)$ + $\text{GFLOPs}_{\text{Tr}}(4096, 10240, 4096, 24, 256) + 24{\cdot}10240{\cdot}4097/10^9$ *Decoder:* $\text{GFLOPs}_{\text{Dec}}(H, W)$

Figure 7: **Stable Diffusion 2 FLOP Derivation (per diffusion step)**

Conventions. Let $h_0 = H/8$, $w_0 = W/8$ be the initial latent dimensions. A multiply–add pair counts as 2 FLOPs. Totals are reported as **GFLOPs** via GFLOPs $=$ FLOPs$/10^9$. All expressions below are single forward-pass costs. The bespoke numbers added are biases that account for discrepancies between our atom-formulas and the actual architecture seen in SD2 (although generally negligible).

Overall Structure.

$$\text{FLOPs}_{\text{SD2}}(H,W) = \text{FLOPs}^{\text{in}} + \sum_{i=1}^{4} \text{FLOPs}_i^{\text{down}} + \text{FLOPs}^{\text{mid}} + \sum_{j=1}^{4} \text{FLOPs}_j^{\text{up}} + \text{FLOPs}^{\text{out}}$$

Stage	Resolution	FLOP Components
ConvIn	$h_0 \times w_0$	$\text{FLOPs}_{\text{conv}}(3, 4, 320, h_0, w_0)$
Down 1	$h_0 \times w_0 \to h_0/2 \times w_0/2$	$2 \cdot \text{FLOPs}_{\text{Res}}(3, 320, 320, h_0, w_0)$ $+ 2 \cdot \text{FLOPs}_{\text{Tr}}(320, 0, 320, 1, h_0 w_0)$ $+ 2 \cdot \text{FLOPs}_{\text{CrossTr}}(320, 1024, 1920, 320, 1, h_0 w_0, 77)$ $+ 4 h_0 w_0 \cdot 320^2$ (proj in/out) $+ \text{FLOPs}_{\text{conv}}(3, 320, 320, h_0/2, w_0/2)$
Down 2	$h_0/2 \times w_0/2 \to h_0/4 \times w_0/4$	$\text{FLOPs}_{\text{Res}}(3, 320, 640, h_0/2, w_0/2) + \text{FLOPs}_{\text{Res}}(3, 640, 640, h_0/2, w_0/2)$ $+ 2 \cdot \text{FLOPs}_{\text{Tr}}(640, 0, 640, 1, h_0 w_0/4)$ $+ 2 \cdot \text{FLOPs}_{\text{CrossTr}}(640, 1024, 3840, 640, 1, h_0 w_0/4, 77)$ $+ h_0 w_0 \cdot 640^2$ (proj in/out) $+ \text{FLOPs}_{\text{conv}}(3, 640, 640, h_0/4, w_0/4)$
Down 3	$h_0/4 \times w_0/4 \to h_0/8 \times w_0/8$	$\text{FLOPs}_{\text{Res}}(3, 640, 1280, h_0/4, w_0/4) + \text{FLOPs}_{\text{Res}}(3, 1280, 1280, h_0/4, w_0/4)$ $+ 2 \cdot \text{FLOPs}_{\text{Tr}}(1280, 0, 1280, 1, h_0 w_0/16)$ $+ 2 \cdot \text{FLOPs}_{\text{CrossTr}}(1280, 1024, 7680, 1280, 1, h_0 w_0/16, 77)$ $+ \frac{h_0 w_0}{4} \cdot 1280^2$ (proj in/out) $+ \text{FLOPs}_{\text{conv}}(3, 1280, 1280, h_0/8, w_0/8)$
Down 4	$h_0/8 \times w_0/8$	$2 \cdot \text{FLOPs}_{\text{Res}}(3, 1280, 1280, h_0/8, w_0/8)$
Mid	$h_0/8 \times w_0/8$	$2 \cdot \text{FLOPs}_{\text{Res}}(3, 1280, 1280, h_0/8, w_0/8)$ $+ \text{FLOPs}_{\text{Tr}}(1280, 0, 1280, 1, h_0 w_0/64)$ $+ \text{FLOPs}_{\text{CrossTr}}(1280, 1024, 7680, 1280, 1, h_0 w_0/64, 77)$ $+ \frac{h_0 w_0}{16} \cdot 1280^2$ (proj in/out)
Up 1	$h_0/8 \times w_0/8 \to h_0/4 \times w_0/4$	$3 \cdot \text{FLOPs}_{\text{Res}}(3, 2560, 1280, h_0/8, w_0/8)$ $+ \text{FLOPs}_{\text{conv}}(3, 1280, 1280, h_0/4, w_0/4)$
Up 2	$h_0/4 \times w_0/4 \to h_0/2 \times w_0/2$	$2 \cdot \text{FLOPs}_{\text{Res}}(3, 2560, 1280, h_0/4, w_0/4) + \text{FLOPs}_{\text{Res}}(3, 1920, 1280, h_0/4, w_0/4)$ $+ 3 \cdot \text{FLOPs}_{\text{Tr}}(1280, 0, 1280, 1, h_0 w_0/16)$ $+ 3 \cdot \text{FLOPs}_{\text{CrossTr}}(1280, 1024, 7680, 1280, 1, h_0 w_0/16, 77)$ $+ \frac{3 h_0 w_0}{4} \cdot 1280^2$ (proj in/out) $+ \text{FLOPs}_{\text{conv}}(3, 1280, 1280, h_0/2, w_0/2)$
Up 3	$h_0/2 \times w_0/2 \to h_0 \times w_0$	$\text{FLOPs}_{\text{Res}}(3, 1920, 640, h_0/2, w_0/2) + \text{FLOPs}_{\text{Res}}(3, 1280, 640, h_0/2, w_0/2)$ $+ \text{FLOPs}_{\text{Res}}(3, 960, 640, h_0/2, w_0/2)$ $+ 3 \cdot \text{FLOPs}_{\text{Tr}}(640, 0, 640, 1, h_0 w_0/4)$ $+ 3 \cdot \text{FLOPs}_{\text{CrossTr}}(640, 1024, 3840, 640, 1, h_0 w_0/4, 77)$ $+ 3 h_0 w_0 \cdot 640^2$ (proj in/out) $+ \text{FLOPs}_{\text{conv}}(3, 640, 640, h_0, w_0)$
Up 4	$h_0 \times w_0$	$2 \cdot \text{FLOPs}_{\text{Res}}(3, 640, 320, h_0, w_0) + \text{FLOPs}_{\text{Res}}(3, 960, 320, h_0, w_0)$ $+ 3 \cdot \text{FLOPs}_{\text{Tr}}(320, 0, 320, 1, h_0 w_0)$ $+ 3 \cdot \text{FLOPs}_{\text{CrossTr}}(320, 1024, 1920, 320, 1, h_0 w_0, 77)$ $+ 12 h_0 w_0 \cdot 320^2$ (proj in/out)
ConvOut	$h_0 \times w_0$	$\text{FLOPs}_{\text{conv}}(3, 320, 4, h_0, w_0)$

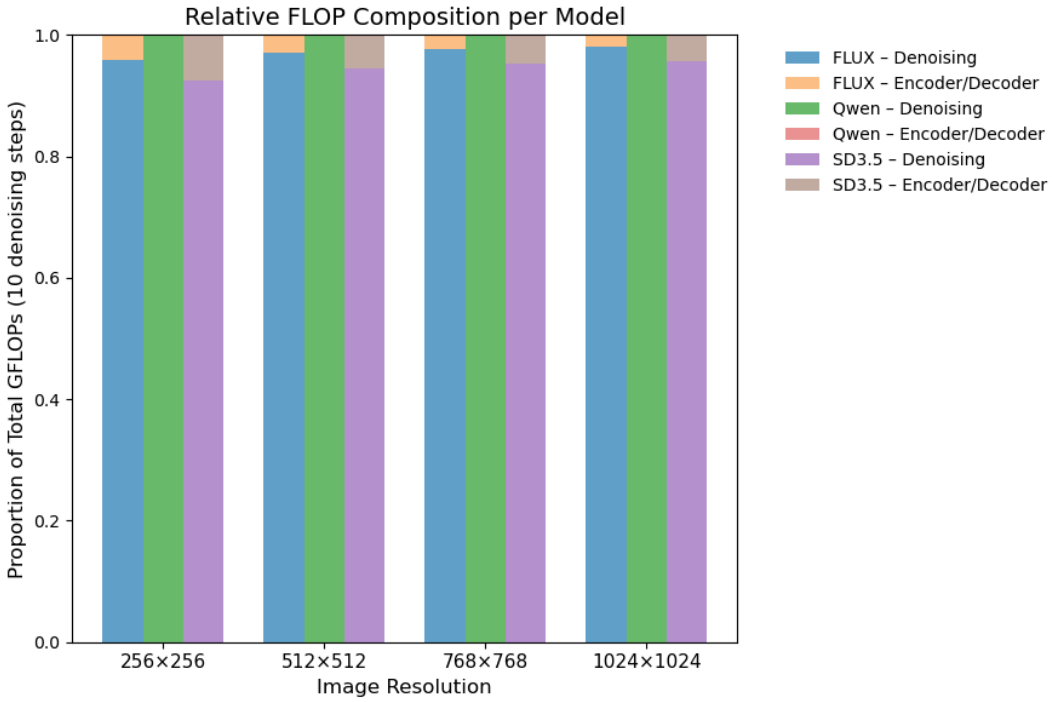


Figure 8: **Relative compute distribution across models and resolutions.** For each model (FLUX, QWEN, and SD 3.5), we plot the proportion of total GFLOPs attributed to the iterative denoising process (scaled to 10 diffusion steps) versus the encoder/decoder overhead.

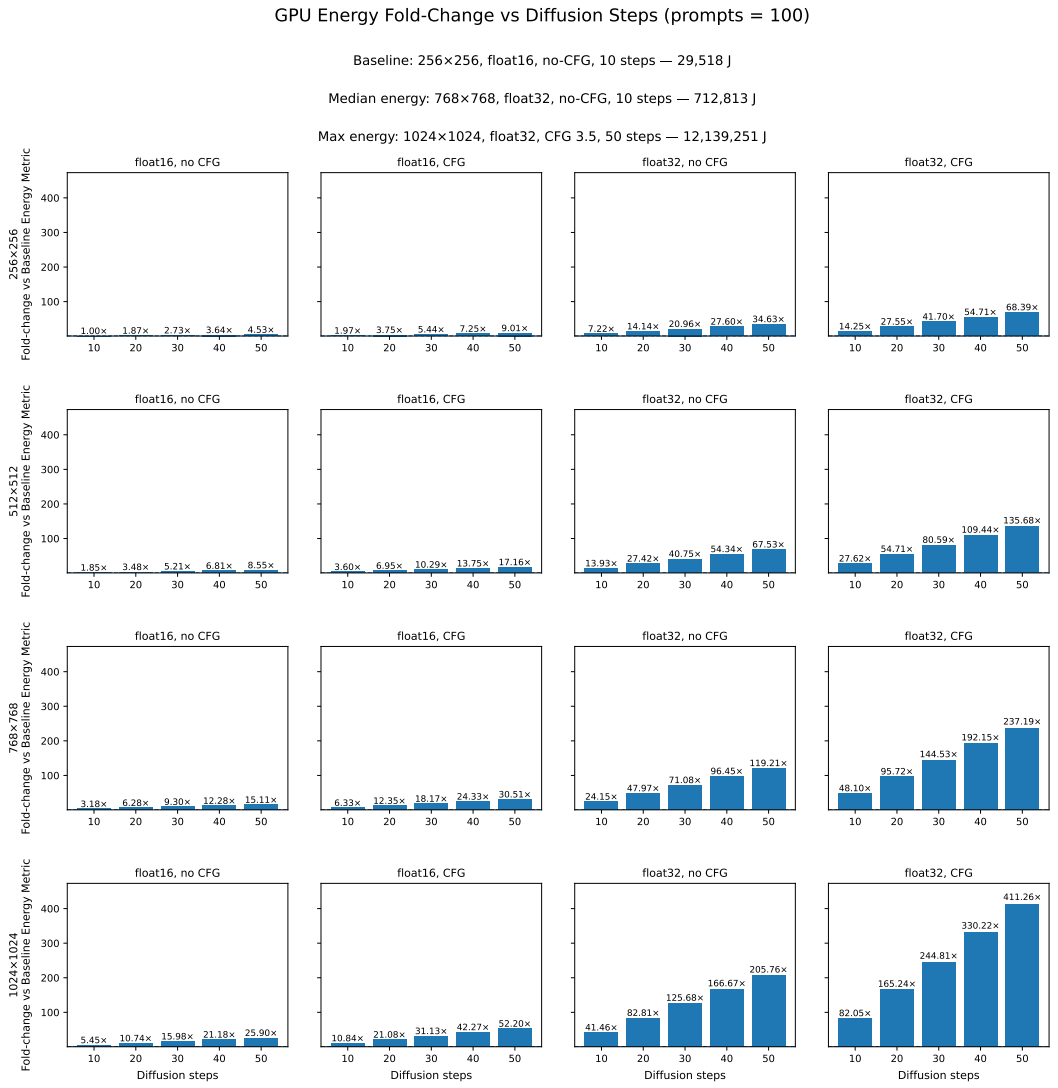


Figure 9: Energy scaling of Flux, relative to smallest energy setting (baseline). Flux has 38 layers in its MMDiT and 17B parameters total (12B MMDit, 5B text encoder, 80M VAE parameters).

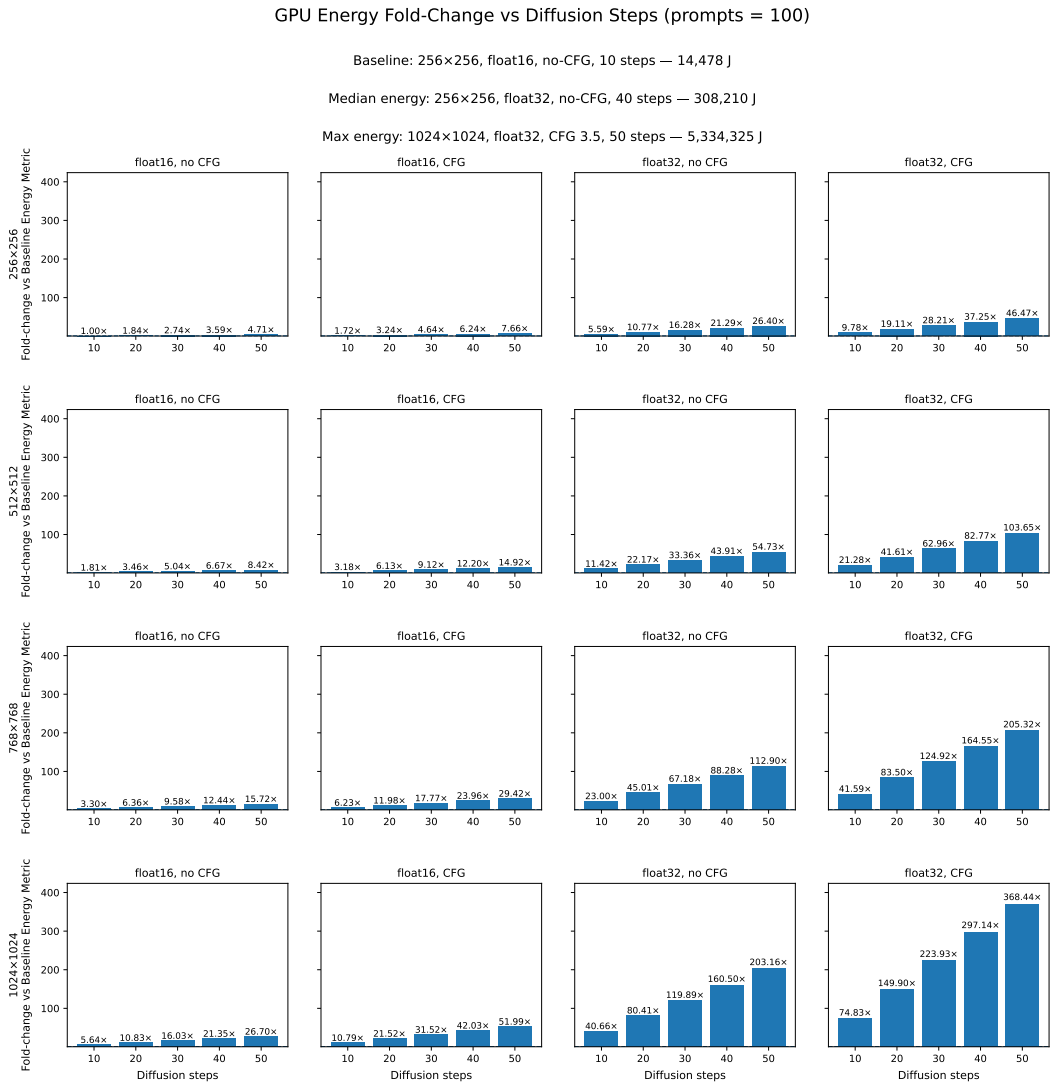


Figure 10: Energy scaling of SD3-5, relative to smallest energy setting (baseline). SD3-5 has 38 layers in its MMDiT and 14B parameters total (8B MMDit, 5B text encoder, 80M VAE parameters).

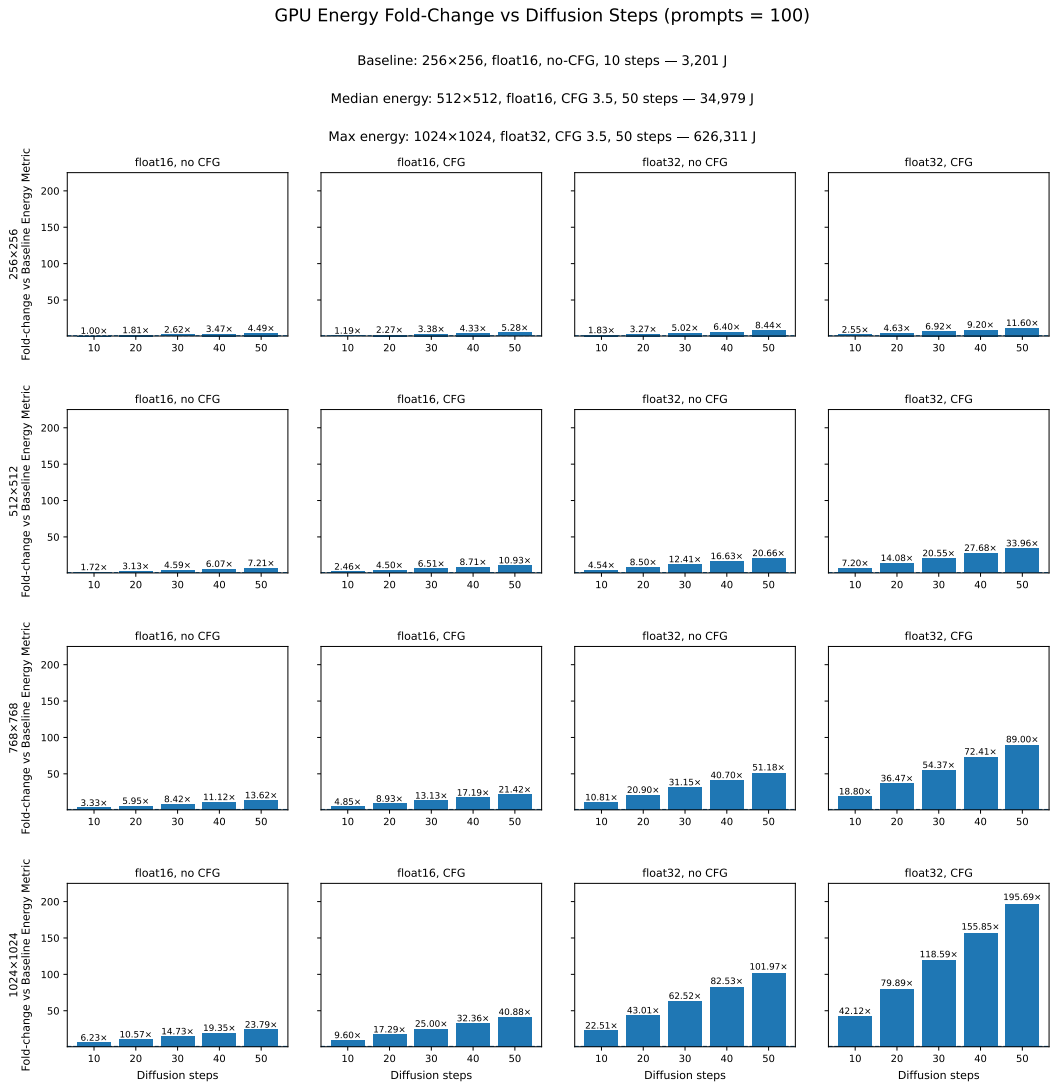


Figure 11: Energy scaling of SD2, relative to smallest energy setting (baseline). SD2 has 9 main blocks and 1.2B parameters total (866M UNet, 340M text encoder, 83M VAE parameters).

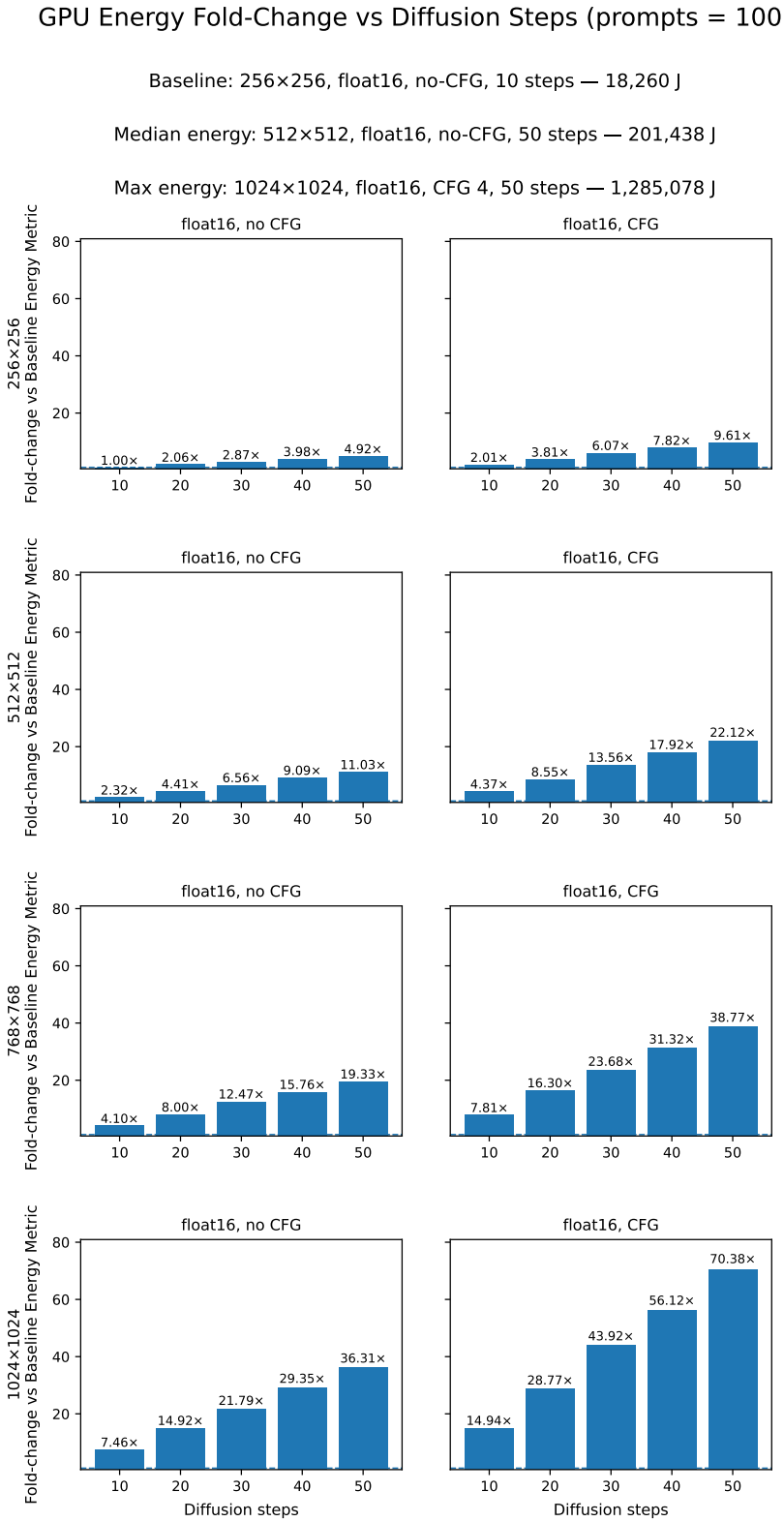


Figure 12: Energy scaling of Qwen, relative to smallest energy setting (baseline). Qwen has 60 layers in its MMDiT and 28B parameters total (20B MMDit, 8B text encoder, 126M VAE parameters).

Table 3: A100 GPU energy (Joules) for hyperparameter settings for FLUX (100 prompts).

Resolution	Steps	f16 no-CFG	f16 CFG	f32 no-CFG	f32 CFG
256×256	10	2.95×10^4	5.81×10^4	2.13×10^5	4.21×10^5
	20	5.53×10^4	1.11×10^5	4.17×10^5	8.13×10^5
	30	8.06×10^4	1.61×10^5	6.19×10^5	1.23×10^6
	40	1.08×10^5	2.14×10^5	8.15×10^5	1.61×10^6
	50	1.34×10^5	2.66×10^5	1.02×10^6	2.02×10^6
512×512	10	5.45×10^4	1.06×10^5	4.11×10^5	8.15×10^5
	20	1.03×10^5	2.05×10^5	8.09×10^5	1.62×10^6
	30	1.54×10^5	3.04×10^5	1.20×10^6	2.38×10^6
	40	2.01×10^5	4.06×10^5	1.60×10^6	3.23×10^6
	50	2.52×10^5	5.07×10^5	1.99×10^6	4.00×10^6
768×768	10	9.38×10^4	1.87×10^5	7.13×10^5	1.42×10^6
	20	1.85×10^5	3.65×10^5	1.42×10^6	2.83×10^6
	30	2.75×10^5	5.36×10^5	2.10×10^6	4.27×10^6
	40	3.62×10^5	7.18×10^5	2.85×10^6	5.67×10^6
	50	4.46×10^5	9.01×10^5	3.52×10^6	7.00×10^6
1024×1024	10	1.61×10^5	3.20×10^5	1.22×10^6	2.42×10^6
	20	3.17×10^5	6.22×10^5	2.44×10^6	4.88×10^6
	30	4.72×10^5	9.19×10^5	3.71×10^6	7.23×10^6
	40	6.25×10^5	1.25×10^6	4.92×10^6	9.75×10^6
	50	7.65×10^5	1.54×10^6	6.07×10^6	1.21×10^7

Table 4: A100 GPU energy (Joules) for different hyperparameter settings for SD3-5 (100 prompts).

Resolution	Steps	f16 no-CFG	f16 CFG	f32 no-CFG	f32 CFG
256×256	10	1.45×10^4	2.49×10^4	8.09×10^4	1.42×10^5
	20	2.66×10^4	4.69×10^4	1.56×10^5	2.77×10^5
	30	3.97×10^4	6.72×10^4	2.36×10^5	4.08×10^5
	40	5.20×10^4	9.04×10^4	3.08×10^5	5.39×10^5
	50	6.82×10^4	1.11×10^5	3.82×10^5	6.73×10^5
512×512	10	2.62×10^4	4.60×10^4	1.65×10^5	3.08×10^5
	20	5.00×10^4	8.88×10^4	3.21×10^5	6.02×10^5
	30	7.29×10^4	1.32×10^5	4.83×10^5	9.11×10^5
	40	9.66×10^4	1.77×10^5	6.36×10^5	1.20×10^6
	50	1.22×10^5	2.16×10^5	7.92×10^5	1.50×10^6
768×768	10	4.78×10^4	9.02×10^4	3.33×10^5	6.02×10^5
	20	9.21×10^4	1.73×10^5	6.52×10^5	1.21×10^6
	30	1.39×10^5	2.57×10^5	9.73×10^5	1.81×10^6
	40	1.80×10^5	3.47×10^5	1.28×10^6	2.38×10^6
	50	2.28×10^5	4.26×10^5	1.63×10^6	2.97×10^6
1024×1024	10	8.16×10^4	1.56×10^5	5.89×10^5	1.08×10^6
	20	1.57×10^5	3.12×10^5	1.16×10^6	2.17×10^6
	30	2.32×10^5	4.56×10^5	1.74×10^6	3.24×10^6
	40	3.09×10^5	6.08×10^5	2.32×10^6	4.30×10^6
	50	3.87×10^5	7.53×10^5	2.94×10^6	5.33×10^6

Table 5: A100 GPU energy (Joules) for different hyperparameter settings for SD2 (100 prompts).

Resolution	Steps	f16 no-CFG	f16 CFG	f32 no-CFG	f32 CFG
256×256	10	3.20×10^3	3.81×10^3	5.84×10^3	8.18×10^3
	20	5.80×10^3	7.26×10^3	1.05×10^4	1.48×10^4
	30	8.40×10^3	1.08×10^4	1.61×10^4	2.22×10^4
	40	1.11×10^4	1.39×10^4	2.05×10^4	2.94×10^4
	50	1.44×10^4	1.69×10^4	2.70×10^4	3.71×10^4
512×512	10	5.50×10^3	7.88×10^3	1.45×10^4	2.31×10^4
	20	1.00×10^4	1.44×10^4	2.72×10^4	4.51×10^4
	30	1.47×10^4	2.08×10^4	3.97×10^4	6.58×10^4
	40	1.94×10^4	2.79×10^4	5.32×10^4	8.86×10^4
	50	2.31×10^4	3.50×10^4	6.61×10^4	1.09×10^5
768×768	10	1.07×10^4	1.55×10^4	3.46×10^4	6.02×10^4
	20	1.90×10^4	2.86×10^4	6.69×10^4	1.17×10^5
	30	2.69×10^4	4.20×10^4	9.97×10^4	1.74×10^5
	40	3.56×10^4	5.50×10^4	1.30×10^5	2.32×10^5
	50	4.36×10^4	6.85×10^4	1.64×10^5	2.85×10^5
1024×1024	10	1.99×10^4	3.07×10^4	7.20×10^4	1.35×10^5
	20	3.38×10^4	5.53×10^4	1.38×10^5	2.56×10^5
	30	4.71×10^4	8.00×10^4	2.00×10^5	3.80×10^5
	40	6.19×10^4	1.04×10^5	2.64×10^5	4.99×10^5
	50	7.61×10^4	1.31×10^5	3.26×10^5	6.26×10^5

Table 6: A100 GPU energy (Joules) for different hyperparameter settings for Qwen (100 prompts).

Resolution	Steps	f16 no-CFG	f16 CFG
256×256	10	1.83×10^4	3.67×10^4
	20	3.76×10^4	6.96×10^4
	30	5.25×10^4	1.11×10^5
	40	7.26×10^4	1.43×10^5
	50	8.99×10^4	1.75×10^5
512×512	10	4.24×10^4	7.97×10^4
	20	8.05×10^4	1.56×10^5
	30	1.20×10^5	2.48×10^5
	40	1.66×10^5	3.27×10^5
	50	2.01×10^5	4.04×10^5
768×768	10	7.48×10^4	1.43×10^5
	20	1.46×10^5	2.98×10^5
	30	2.28×10^5	4.32×10^5
	40	2.88×10^5	5.72×10^5
	50	3.53×10^5	7.08×10^5
1024×1024	10	1.36×10^5	2.73×10^5
	20	2.72×10^5	5.25×10^5
	30	3.98×10^5	8.02×10^5
	40	5.36×10^5	1.02×10^6
	50	6.63×10^5	1.29×10^6