Stable Diffusion 1.5 Inference Optimization on an NVIDIA A100: Speed–Memory–Quality Tradeoffs

Deeptanshu Yashdeep Prasad
New York University
dl5473@nyu.edu yp2693@nyu.edu

***Abstract*—Diffusion-based text-to-image generation is widely used but remains expensive at inference time due to sequential denoising and repeated UNet evaluations. We study inference-only optimizations for Stable Diffusion 1.5 using stock PyTorch and the Hugging Face `diffusers` pipeline on an NVIDIA A100. Our final configuration combines (i) engineering optimizations (FP16, channels-last, `torch.compile`), (ii) faster sampling (DPMSolverMultistep with 20 steps), (iii) a dynamic classifier-free guidance (CFG) cutoff that drops the unconditional branch after 0.4 of timesteps, and (iv) a Tiny VAE decoder. Compared to the baseline (Euler-Ancestral, 30 steps), we achieve up to $4.38\times$ lower latency (1.30 s/img $\rightarrow$ 0.297 s/img) with a modest increase in peak VRAM (2.8 GB $\rightarrow$ 4.13 GB), while maintaining comparable perceptual quality.**

***Index Terms*—Stable Diffusion, diffusion models, inference optimization, torch.compile, schedulers, CFG, GPU performance**

I. Introduction

A. Background and Motivation

Latent diffusion models (LDMs) generate images by iteratively denoising a latent variable through a UNet conditioned on a text embedding [1]. While LDMs reduce compute relative to pixel-space diffusion, inference is still dominated by sequential denoising steps and the UNet forward pass. For interactive applications, lowering latency and memory footprint is essential.

B. Problem Statement

Given a fixed, pretrained Stable Diffusion 1.5 pipeline, can we significantly reduce inference time on a modern GPU without unacceptable quality loss, using only stock PyTorch and `diffusers` (no TensorRT, xFormers, or custom CUDA extensions)?

C. Objectives and Scope

Our primary objective is to minimize end-to-end image generation time at 512×512 resolution. A secondary objective is to reduce peak VRAM where possible. We optimize *inference only* (no training or fine-tuning). Quality is evaluated qualitatively through side-by-side comparisons under fixed prompts and seeds.

II. Literature Review

A. Review of Relevant Literature

Stable Diffusion is based on LDMs [1]. Sampling improvements are a major route to accelerating diffusion inference; DPM-Solver provides high-order solvers that can generate high-quality samples in 10–20 function evaluations [2]. Classifier-free guidance (CFG) improves prompt adherence by mixing conditional and unconditional predictions, but roughly doubles UNet compute [3]. Systems-level approaches include compiler-based kernel fusion and graph lowering (e.g., `torch.compile`) [4], [5].

III. Methodology

A. Data Collection and Preprocessing

We do not use a dataset. Instead, we generate images for a fixed set of prompts (e.g., "a lighthouse at sunset, cinematic lighting, 35mm, detailed") and fixed random seeds to ensure fair comparisons across configurations.

B. Model Selection

We evaluate the Stable Diffusion 1.5 pipeline (CLIP text encoder, UNet denoiser, VAE decoder, scheduler) using `diffusers` [6]. Unless specified, we use FP16 weights and activations.

C. Optimization Procedure(s): Training vs. Inference

All optimizations are **inference-only**. We combine system-level, architectural, and algorithmic changes:

Engineering (kernel-level): We enable channels-last memory layout and TF32 matmuls on A100, and apply `torch.compile` (Inductor backend, `mode=max-autotune`) to UNet and VAE [4]. Empirically, `torch.compile` improves latency with no observable quality change.

Architectural (decoder swap): We replace the full VAE decoder with `AutoencoderTiny` (TAESD) to reduce decoding cost and VRAM [7], [8]. This introduces a small softening of fine textures.

Algorithmic (sampling): We compare Euler-Ancestral sampling to DPMSolverMultistep and reduce steps from 30 to 20 [9], [10], [2]. We observe Euler-Ancestral at 20 steps is softer than 30 steps, whereas DPMSolverMultistep at 20 steps better preserves quality.

Algorithmic (dynamic CFG cutoff): CFG requires two UNet evaluations per step (conditional + unconditional) [3]. We implement a dynamic cutoff via `callback_on_step_end`: after 0.4 of timesteps, we drop the unconditional branch by removing the unconditional half of `prompt_embeds` and setting guidance scale to 0. If the cutoff is reduced too aggressively, we observe artifacts such as curved lines where straight edges are expected and prompt-incoherent objects.

D. Profiling Tools and Methods

We report (i) average latency (seconds per image), measured after warm-up using `torch.cuda.synchronize()` and wall-clock timing, and (ii) peak VRAM via `torch.cuda.max_memory_allocated()` (reset between runs). Each configuration is run multiple times and averaged.

E. Evaluation Metrics

Latency: seconds per generated 512×512 image.
Memory: peak allocated VRAM (GB).
Quality: qualitative visual inspection for prompt adherence, sharpness, and artifacts.

IV. Experimental Results

A. Experimental Setup

All experiments were run on an NVIDIA A100 (Colab) at 512×512 resolution with single-image generation. For timing, we use wall-clock measurements with `torch.cuda.synchronize()` before/after inference and report steady-state latency (after warm-up). Peak VRAM is measured using `torch.cuda.max_memory_allocated()` with a reset between runs. Qualitative comparisons are generated using the same prompt and seed across configurations.

B. Experiment 1: Baseline Pipeline

Our baseline uses Stable Diffusion 1.5 with Euler-Ancestral sampling at 30 denoising steps and FP16. This establishes the reference point for both latency and perceptual quality: 1.30 s/image at 2.8 GB peak VRAM. We treat the baseline as the quality reference for all later changes (sharpness, prompt adherence, texture fidelity).

C. Experiment 2: `torch.compile` and Basic GPU Tuning

This stage targets kernel efficiency and operator fusion. We apply `torch.compile` (Inductor, `mode=max-autotune`) to the UNet and VAE, and use inference-friendly settings (e.g., channels-last). Since inference is dominated by repeated UNet evaluations inside the denoising loop, compilation provides a large gain.

Result: latency improves from 1.30 s/img to 0.552 s/img (∼2.3×), while peak VRAM increases from 2.8 GB to 4.80 GB. Qualitatively, `torch.compile` does not introduce visible degradation; prompt structure and geometry remain consistent with baseline.

D. Experiment 3: TinyVAE for Lower VRAM

We replace the default VAE decoder with TinyVAE (TAESD / `AutoencoderTiny`), primarily targeting memory reduction. This is especially useful because compilation increases peak VRAM.

Result: compiled+TinyVAE achieves 0.516 s/img while reducing peak VRAM to 2.18 GB. The main qualitative change is slight softening of fine textures and micro-contrast; global composition and prompt adherence remain similar.

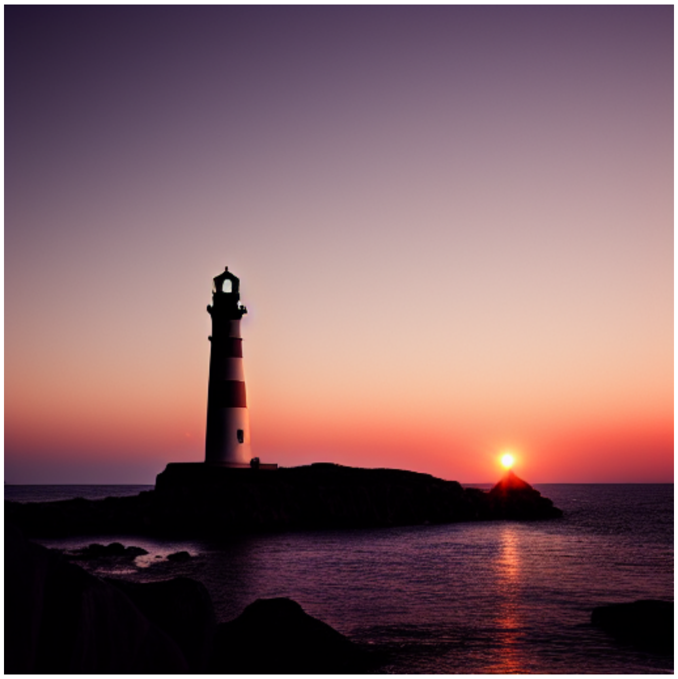


Fig. 1. Baseline output (Euler-Ancestral, 30 steps): 1.30 s/img, 2.8 GB peak VRAM.

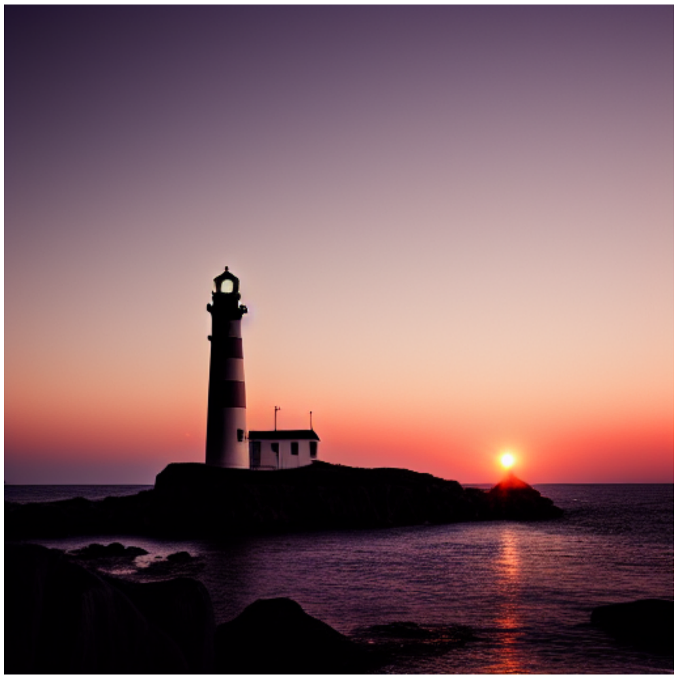


Fig. 2. `torch.compile` tuned output: speed improves significantly with no noticeable quality change, but peak VRAM increases (0.552 s/img, 4.80 GB).

E. Experiment 4: Scheduler and Step Reduction (20 Steps)

Because each denoising step requires expensive UNet computation, reducing steps is a direct route to lower latency.

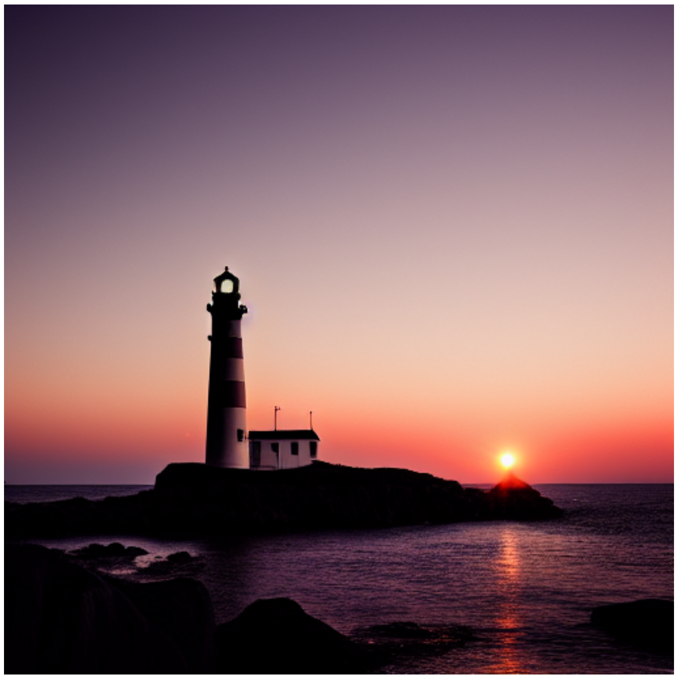


Fig. 3. Compiled + TinyVAE output: large VRAM reduction with slightly softer details (0.516 s/img, 2.18 GB).

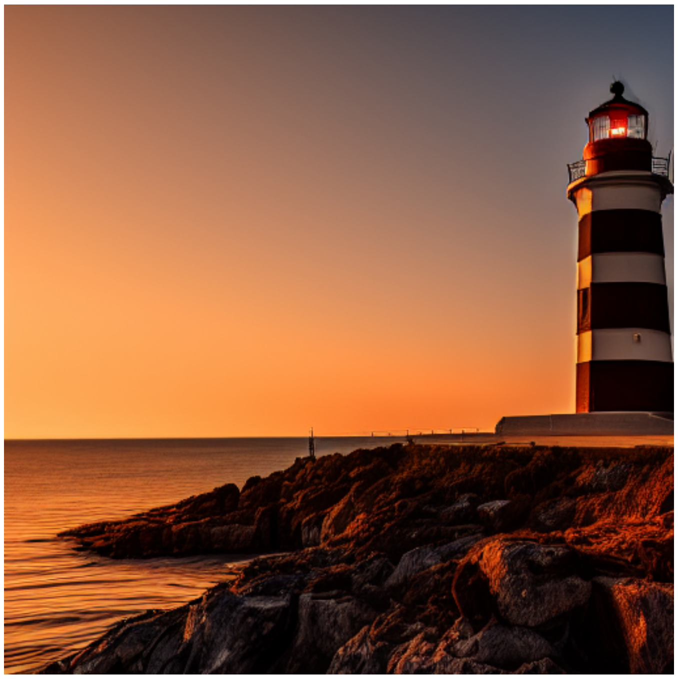


Fig. 4. DPMSolverMultistep at 20 steps: better quality retention than Euler at the same step budget while remaining fast.

We evaluate both (i) Euler-Ancestral at 20 steps and (ii) DPMSolverMultistep at 20 steps.

Result: Euler-Ancestral at 20 steps is faster but tends to produce softer images than 30-step sampling. In contrast, DPMSolverMultistep at 20 steps retains perceptual quality better (sharper edges and better texture retention) while staying in the same latency regime. Based on this, we adopt DPMSolverMultistep at 20 steps for later experiments.

F. Experiment 5: Dynamic CFG Cutoff (40%)

Classifier-Free Guidance (CFG) typically doubles UNet compute (conditional and unconditional passes). We implement a dynamic cutoff using `callback_on_step_end`: after 40% of timesteps, we drop the unconditional branch and set guidance to 0. The intuition is that early steps establish prompt alignment, while later steps focus on refinement.

Result: applying a 40% cutoff reduces latency to ≈0.347 s/img (DPMSolver, 20 steps) with minimal perceptual change under moderate cutoffs. However, if the cutoff is reduced too aggressively, quality degrades: we observe warped geometry (e.g., curved lines where straight edges are expected), artifacts, and unrelated objects appearing. Therefore, the cutoff must be tuned to balance speed and prompt fidelity.

G. Experiment 6: Final Configuration

Our final system combines: FP16 + channels-last + `torch.compile` (UNet+VAE), DPMSolverMultistep with 20 steps, dynamic CFG cutoff at 40%, and TinyVAE decoding. This configuration targets end-to-end latency while keeping quality close to baseline and partially offsetting compilation-induced VRAM overhead via TinyVAE.

Result: 0.297 s/img at 4.13 GB peak VRAM, versus 1.30 s/img at 2.8 GB in baseline. Qualitatively, the final output remains similar in composition and prompt adherence; the most noticeable difference is slightly reduced micro-texture sharpness, largely attributable to TinyVAE.

H. Aggregate Performance Summary

Table I consolidates latency, VRAM, and the qualitative takeaway from each stage. Across experiments we observe a consistent tradeoff between inference time, peak VRAM, and visual fidelity: compilation improves speed but increases VRAM; TinyVAE reduces VRAM with slight softening; fewer steps reduce runtime but can soften outputs; dynamic CFG cutoff accelerates inference but can introduce artifacts if applied too aggressively.

I. Final Result Plots

We include the final summary plots from the presentation that visualize speed–memory tradeoffs and performance relative to baseline.

V. Batched Inference Configuration

All final performance measurements were obtained using a fixed model configuration designed to maximize throughput on a single NVIDIA A100 GPU. The Stable Diffusion 1.5 pipeline was executed with a batch size of 64 images, which was the maximum batch size that fit in GPU memory for this configuration.

The model used a DPM-Solver++ multistep scheduler with 20 denoising steps, a TinyVAE decoder, and classifier-free guidance with a dynamic cutoff applied after 40% of the

TABLE I
PERFORMANCE SUMMARY AT 512×512 (SINGLE IMAGE).

Configuration	Scheduler / Steps	Time (s/img)	VRAM (GB)	Qualitative takeaway
Baseline	Euler-A / 30	1.30	2.80	Reference quality.
`torch.compile` tuned	Euler-A / 30	0.552	4.80	No quality change; VRAM increases.
Compiled + TinyVAE	Euler-A / 30	0.516	2.18	Slightly softer fine detail; large VR drop.
Euler-A step reduction	Euler-A / 20	0.395	5.11	Faster but softer than 30 steps.
Scheduler swap	DPMSolver / 20	0.402	5.11	Better quality than Euler at same step
Dynamic CFG cutoff	DPMSolver / 20 + cutoff 40%	0.347	5.11	Speedup; quality preserved for mod cutoff.
Final	DPMSolver / 20 + cutoff 40% + TinyVAE	0.297	4.13	Best speed; minor softness from TinyV

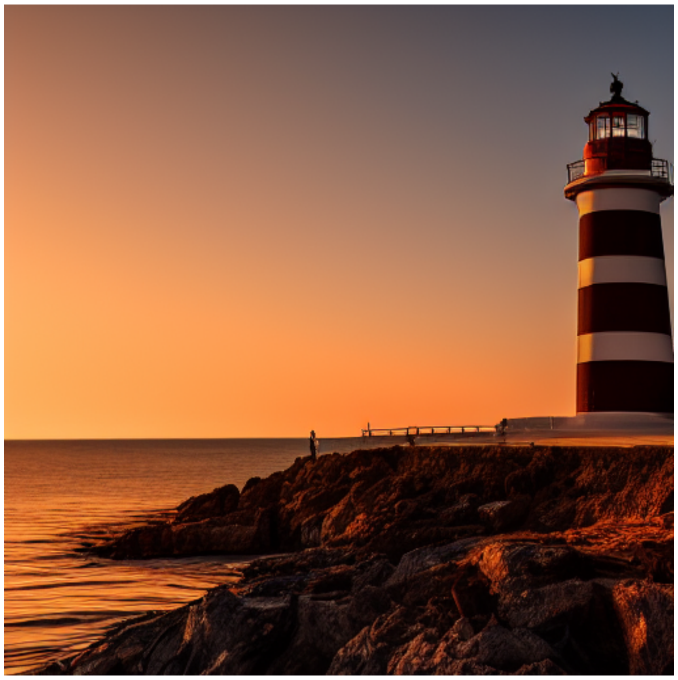


Fig. 5. DPMSolver 20 steps with CFG cutoff at 40%: faster due to fewer unconditional UNet evaluations, while maintaining good prompt fidelity under a moderate cutoff.

diffusion steps. The UNet and VAE were compiled using `torch.compile` with `max-autotune` enabled.

Using this configuration, 192 images were generated in 55.2 seconds, corresponding to an average latency of 0.288 seconds per image (3.48 images per second). Peak GPU memory usage during execution was approximately 11.9 GB.

VI. DISCUSSION

A. Interpretation of Results

The best latency improvements come from reducing the number of expensive UNet evaluations (steps and CFG compute). Memory reductions are mainly driven by decoder choice (TinyVAE), while compilation can increase peak VRAM due to runtime/graph overheads.

B. Comparison with Previous Studies

The strong performance at 20 steps with DPM-Solver aligns with prior findings that high-order solvers can maintain sample quality with fewer evaluations [2]. CFG behavior follows the original classifier-free guidance formulation [3].

VII. CONCLUSION

A. Summary of Findings

By combining compiler optimizations, faster schedulers, a dynamic CFG cutoff, and a TinyVAE decoder, we reduce Stable Diffusion 1.5 latency from 1.30 s/img to 0.297 s/img (4.38×) with peak VRAM increasing from 2.8 GB to 4.13 GB.

B. Contributions

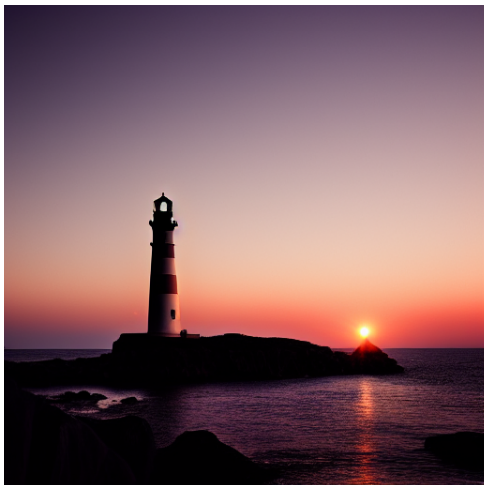
We provide an inference-only ablation of practical, stock PyTorch/`diffusers` optimizations and document the resulting speed–memory–quality tradeoffs, supported by qualitative comparisons.

C. Recommendations for Future Work

Integrate attention optimizations (e.g., xFormers/FlashAttention), deploy with TensorRT, explore learned guidance schedules, and evaluate across GPUs/resolutions/prompts.

REFERENCES

[1] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022, arXiv:2112.10752.
[2] C. Lu, Y. Zhou, F. Bao, J. Chen, C. Li, and J. Zhu, "Dpm-solver: A fast ode solver for diffusion probabilistic model sampling in around 10 steps," arXiv:2206.00927, 2022.
[3] J. Ho and T. Salimans, "Classifier-free diffusion guidance," arXiv:2207.12598, 2022.
[4] PyTorch, "torch.compile (pytorch documentation)," https://docs.pytorch.org/docs/stable/generated/torch.compile.html, 2025, accessed: 2025-12-13.
[5] ——, "Pytorch 2.x: torch.compile," https://pytorch.org/get-started/pytorch-2-x/, 2023, accessed: 2025-12-13.
[6] Hugging Face, "Diffusers: State-of-the-art diffusion models for image and audio generation," https://huggingface.co/docs/diffusers/index, 2025, accessed: 2025-12-13.
[7] ——, "Tiny autoencoder (autoencodertiny) for stable diffusion (diffusers documentation)," https://huggingface.co/docs/diffusers/api/models/autoencoder_tiny, 2025, accessed: 2025-12-13.
[8] O. B. Bohan, "Taesd: Tiny autoencoder for stable diffusion," https://github.com/madebyollin/taesd, 2023, accessed: 2025-12-13.


(a) Baseline

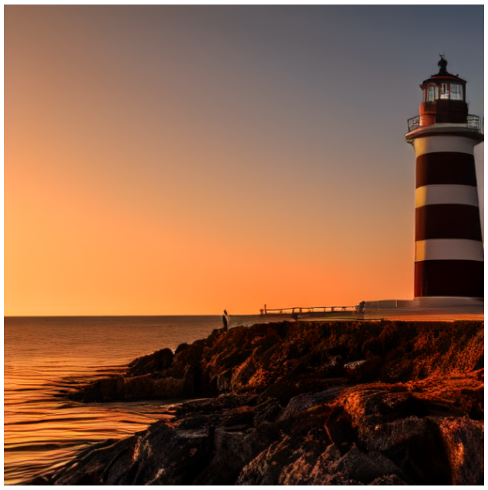
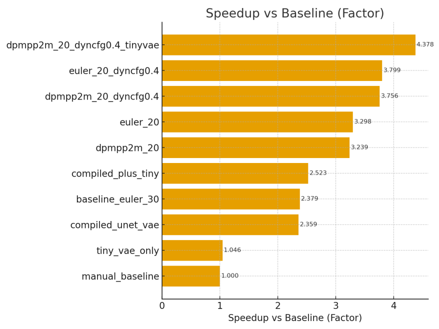
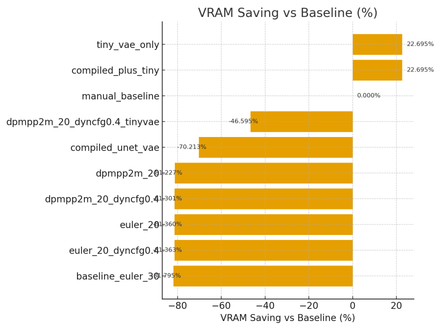

(b) Final

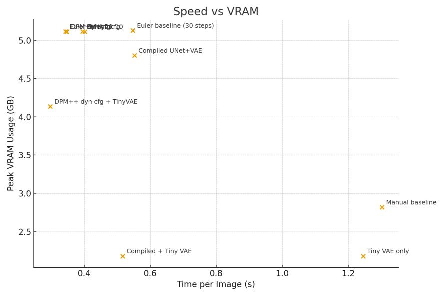
Fig. 6. Baseline vs Final comparison. Final achieves the best latency while maintaining comparable perceptual quality (1.30→0.297 s/img).



(a) Speed vs. VRAM



(b) Speedup vs. baseline



(c) VRAM saving vs. baseline

Fig. 7. Final summary plots: (1) latency vs peak VRAM, (2) speedup factors, and (3) VRAM savings relative to baseline.

[9] Hugging Face, "Eulerancestraldiscretescheduler (diffusers documentation)," https://huggingface.co/docs/diffusers/en/api/schedulers/euler_ancestral, 2025, accessed: 2025-12-13.

[10] ——, "Dpmsolvermultistepscheduler (diffusers documentation)," https://huggingface.co/docs/diffusers/en/api/schedulers/multistep_dpm_solver, 2025, accessed: 2025-12-13.