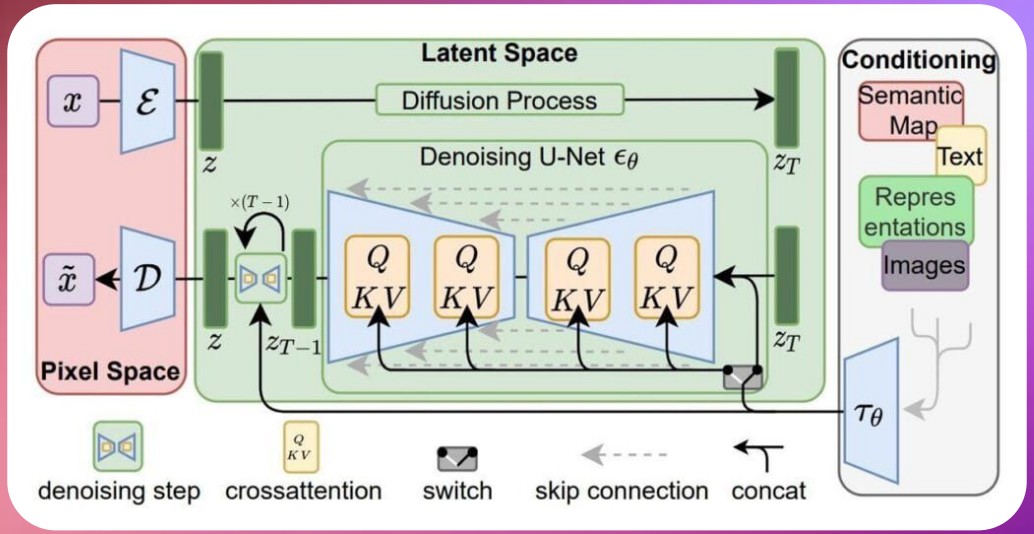
Stable Diffusion Inference Optimization on A100

Deeptanshu and Yashdeep Prasad

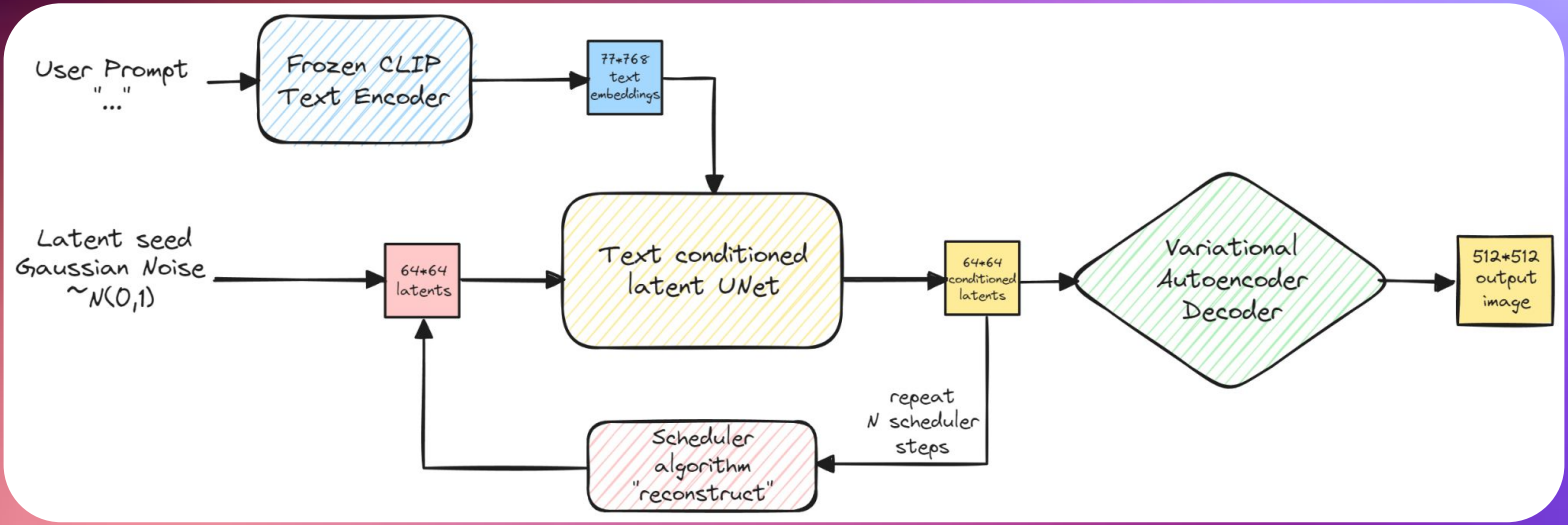
Setup

- **Model:** Stable Diffusion 1.5 (`runwayml/stable-diffusion-v1-5`)
- **Hardware:** NVIDIA A100 GPU (Colab)
- **Goal: Faster** and **lower-memory** image generation without sacrificing visual quality.
- **Constraints:** Optimization used stock PyTorch and Diffusers; did not use TensorRT or xFormers.
- **Experiments:** Conducted at 512×512 resolution, single-image generation.



Stable Diffusion pipeline and Metrics

- The Stable Diffusion pipeline uses CLIP, UNet, VAE, and an Euler Ancestral scheduler.
- The UNet denoiser is the system's primary spot for time and memory consumption.
- The baseline uses the default SD 1.5 pipeline with 30 steps and fp16 on A100.
- Initial performance is 1.30s/image at 2.8 GB peak VRAM.



Pre-midterm: torch.compile and basic GPU tuning

The first phase (pre-midterm) focused on "classical" GPU tuning for performance.

- Optimizations included running torch.compile (inductor backend, `max-autotune`) on UNet and VAE to fuse ops and optimize kernels.
- Used the channels-last memory format and enabling TF32 matrix multiplies on A100.
- **2.3× speedup** but a **70% increase in VRAM** usage

Baseline

1.30s/image at 2.8 GB VRAM

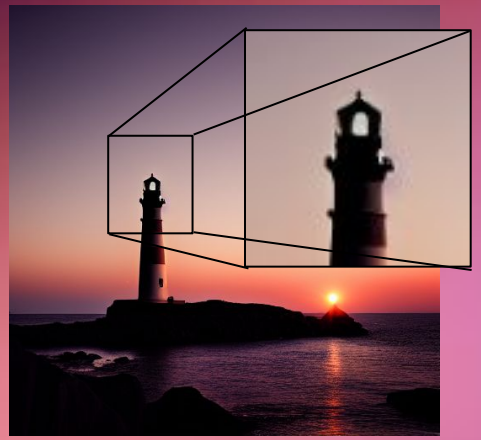
Torch.compile() tuned

0.552 s/img at 4.80 GB VRAM

Tiny VAE for faster, lower-memory decoding

- Swapped the default VAE with AutoencoderTiny(TAESD), a distilled, smaller VAE trained to approximate the original decoder.
- Experiments were run comparing Tiny VAE only, full VAE, and the Compiled model + Tiny VAE.
- There is a quality trade-off, where images are slightly softer in very fine details.

Baseline



1.30s/image at 2.8 GB VRAM

Compiled Model

0.552 s/img at 4.80 GB VRAM

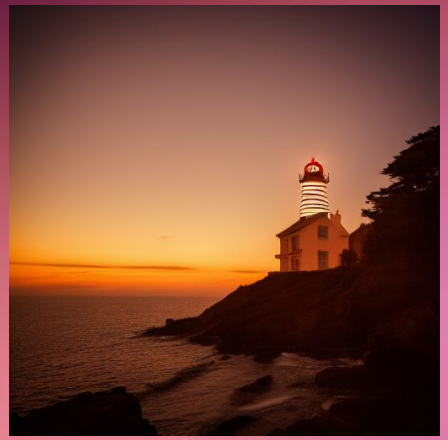
Compiled Model + TinyVAE

0.516 s/img at 2.18 GB VRAM

Changing the denoising schedule

- 30 denoising steps are expensive even on a fast UNet, and many papers show that good schedulers require fewer steps.
- This experiment included varying the Scheduler (E**uler Ancestral vs. DPMSolverMultistep (dpmpp_2m)**) and reducing the number of steps from 30 to 20

Eular 30 steps



0.547 s/image VRAM: 5.13 GB

Eular 20 steps

0.395 s/image VRAM: 5.11 GB

DPMSolver 20 steps

0.402 s/image VRAM: 5.11 GB

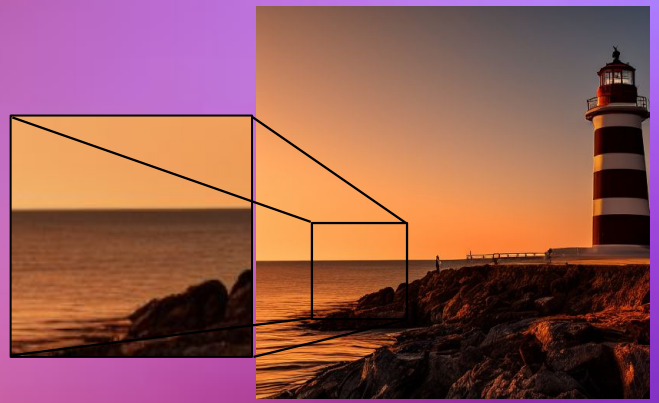
Dynamic CFG (classifier-free guidance) cutoff

- Classifier-Free Guidance (CFG) is expensive because it doubles UNet calls (conditional + unconditional) in every step.
- Later denoising steps contribute less to aligning the image with the prompt, creating an opportunity for optimization.
- The approach uses ‘callback_on_step_end’ in diffusers to implement a dynamic cutoff.
- After a set fraction of timesteps (e.g., 40%), the unconditional branch is dropped, and the guidance scale is set to 0 by modifying ‘Prompt_embeds’ and ‘_guidance_scale’.

Eular 20 steps with CFG

0.343 s/image VRAM: 5.11 GB

DPM Solver 20 steps with CFG



0.347 s/image VRAM: 5.11 GB

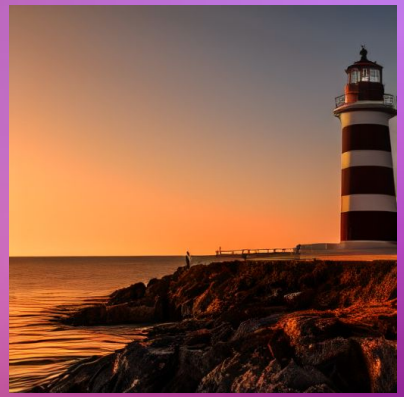
Final Configuration

- Engineering: fp16 + channels-last and `torch.compile` on UNet and VAE.
  Algorithmic: DPMSolverMultistep scheduler with 20 steps and Dynamic CFG cutoff.
  Architectural: Tiny VAE for decoding.
- **Key Takeaways:**
  - The final configuration is a combination of engineering improvements, algorithmic tweaks, and an architectural swap.
  - The final system is significantly faster than the baseline but using slightly more memory.
  - The quality of end result remains similar to the baseline.

Baseline

1.30s/image at 2.8 GB VRAM

DPM Solver 20 steps with CFG and TinyVAE



0.297 s/image VRAM: 4.13 GB

Final Results