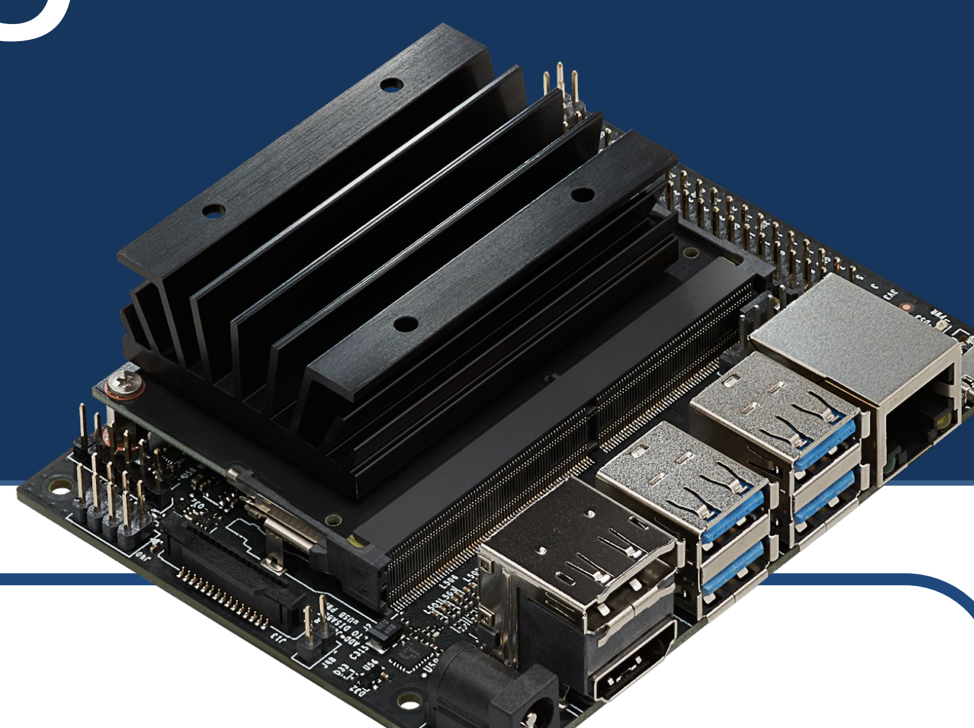
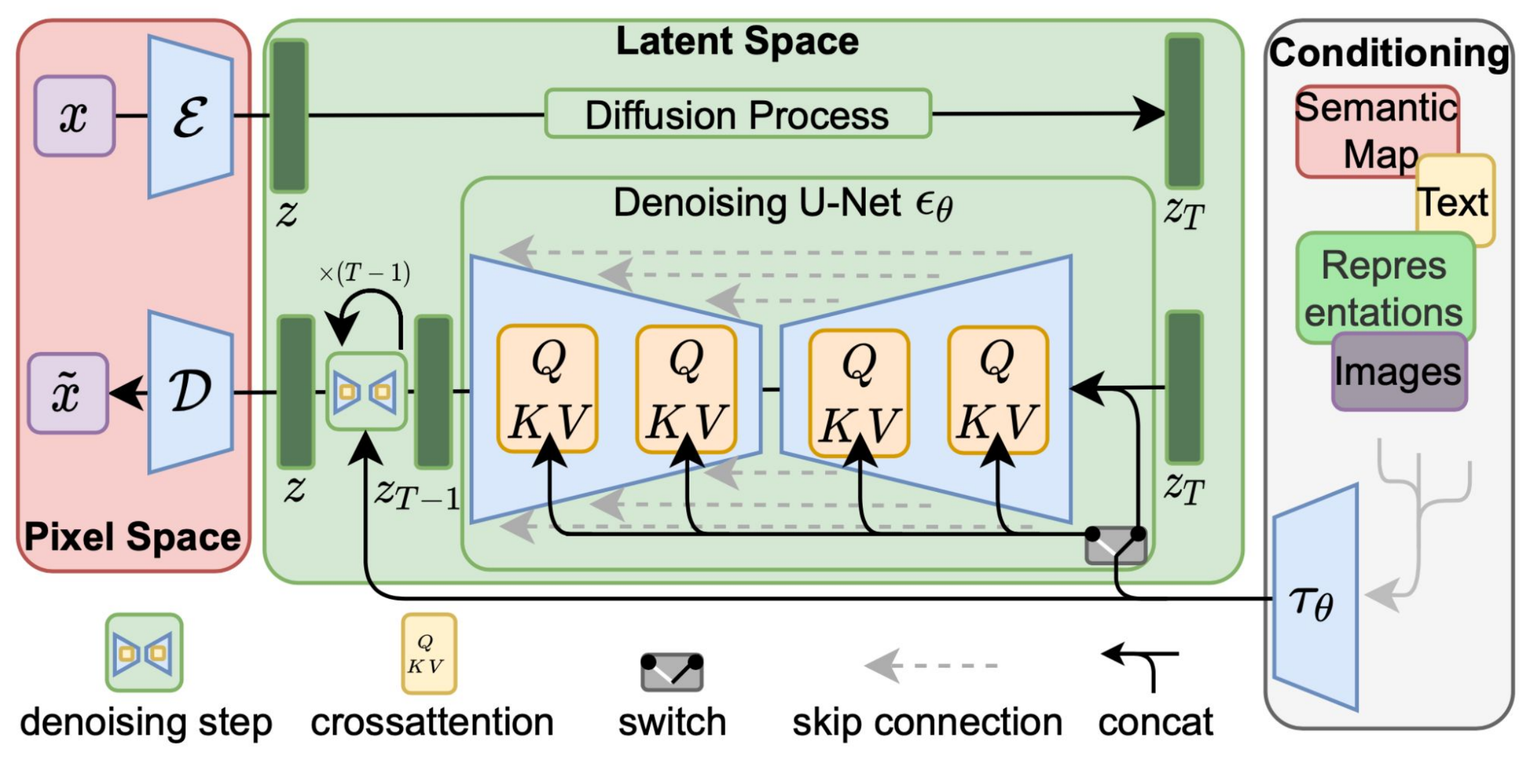
Diffusion Models on Edge

Akash Haridas, Jazib Ahmad, Sumant Bagri
University of Toronto

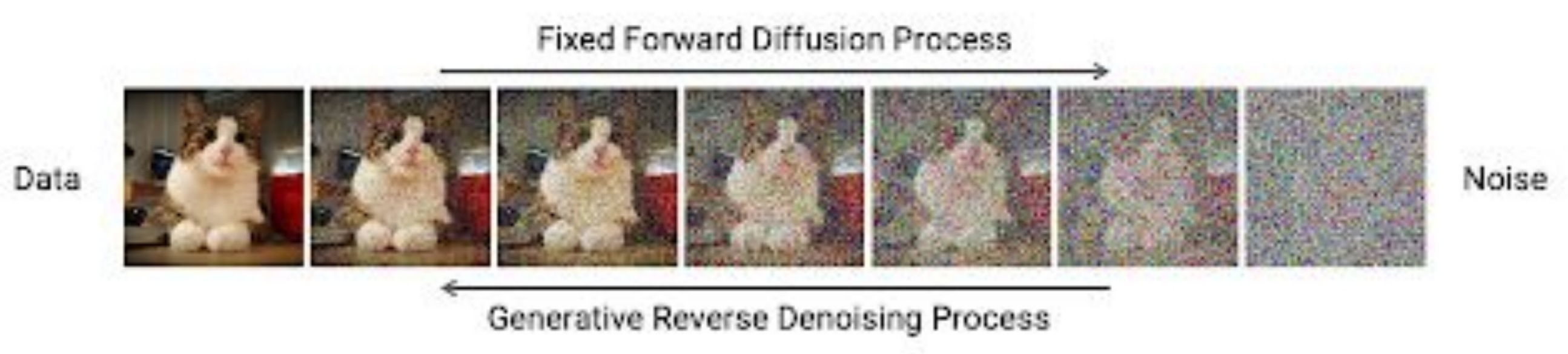


Motivation

- Diffusion models: SOTA model for image synthesis, also have applications in mobile apps and AR/VR



- Start with an image of pure noise and iteratively denoise it using a U-Net


Fixed Forward Diffusion Process
Data
Noise
Generative Reverse Denoising Process

- Computationally expensive: Each generation takes up to 1000 denoising steps
- Challenge to run on cheap and low-power edge devices.
- Motivation: Utilize inexpensive edge devices for diffusion models, allowing these models to become more accessible and consume less energy.
- Tradeoff: Computation time vs Memory vs Power
- Goal: Measure this tradeoff and benchmark it for future work

Related Work

Performance Benchmarking.

- Deep Neural Networks and Deep Convolutional Neural Networks [1] have been benchmarked on edge devices such as Nvidia Jetson Nano and smartphones.
- However, no studies on implementing and benchmarking Diffusion Models on edge devices.

Performance Optimization.

- Pretrained models can be optimized through techniques such as model serialization, graph optimization, operator fusion and post-training quantization [2].

Model Improvements.

- Latent Diffusion Models and advanced sampling techniques are attempts to make the model itself more efficient.

References

[1] Baller, et al. Benchmarking deep neural networks on edge devices. IEEE International Conference on Cloud Engineering (IC2E), 2021
[2] Jacob, et al. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In Proceedings of the IEEE conference on computer vision and pattern recognition, 2018.
[3] Karras, Tero, et al. "Elucidating the design space of diffusion-based generative models." arXiv preprint arXiv:2206.00364 (2022).

Methods

Implementing and benchmarking a **330M**-parameter Latent Diffusion Model on an NVIDIA Jetson Nano and an Android smartphone:

1. Create a custom LDM pipeline and load pretrained weights
2. Perform 16-bit quantization and JIT compilation
3. Graph level optimization using ONNX Runtime
4. Layer and Tensor fusion using TensorRT

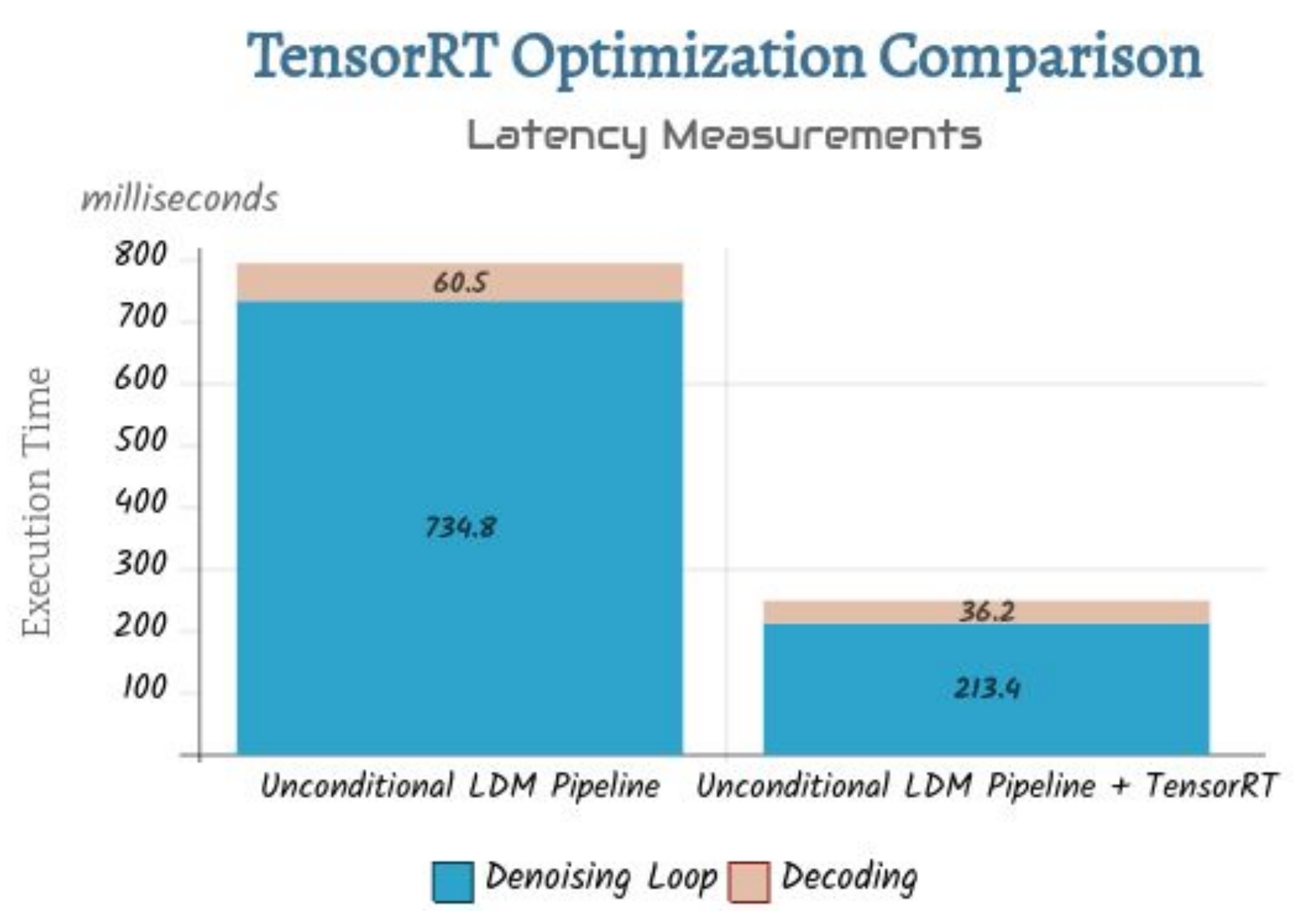
NVIDIA Jetson Nano

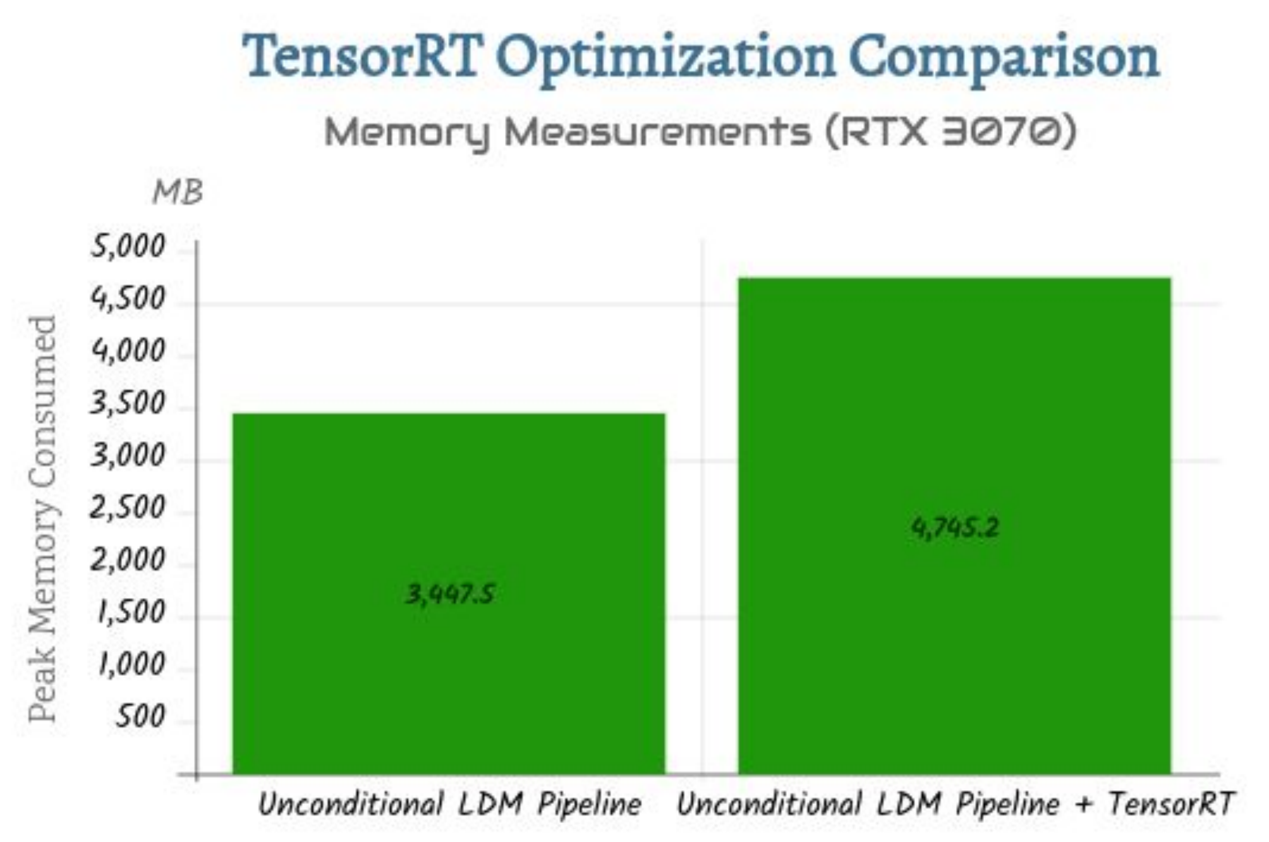
1. Build custom docker images containing TensorRT and ONNX Runtime using NGC's pytorch container as base image
2. Create an evaluation pipeline accommodating the various optimizations
3. Run evaluation pipeline with CUDA-CPU synchronization
4. Extract latency, memory and power metrics using *tegrastats* CLI.
5. *Future*: Perform optimizations by creating TensorRT engines for Jetson Orin Nano

Samsung Galaxy S22 8GB

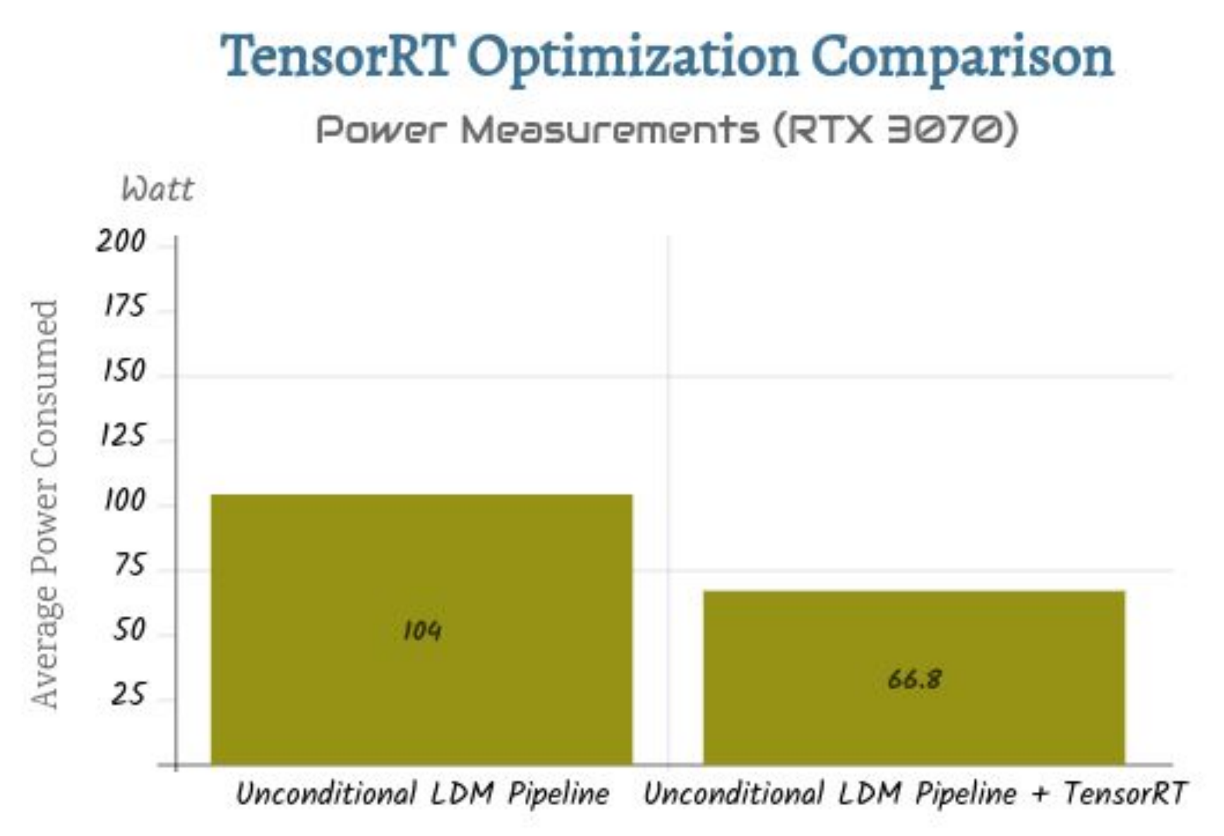
1. Obtain PyTorch binaries for *Arm-v9 (aarch64)* architecture and install it in a Linux environment inside the Termux terminal emulator.
2. Increase available memory by allocating additional swap memory.
3. Run model inference on the CPU, obtain power consumption by measuring battery discharge rate (*mA*).
4. *Future*: Quantize model down to INT8 for significant gains.

Experimental Results





High-quality output produced on edge device



Performance Evaluation Summary

Metric	RTX 3070	Jetson Nano	Android
Latency (s)	1.17	93.22	160.51
Peak Memory (MB)	2032	2473	2379
Power (W)	101.43	3.73	3.52
Total Energy (J)	118.67	347.71	564.96
Relative Latency	1X	79.68X	137.18X
Relative Energy	1X	2.93X	4.76X