Efficient Mobile Text-to-Image Diffusion Models

Huan Wang
Northeastern University, Boston, USA
Talk @ASU, Feb. 09, 2024 (Fri)

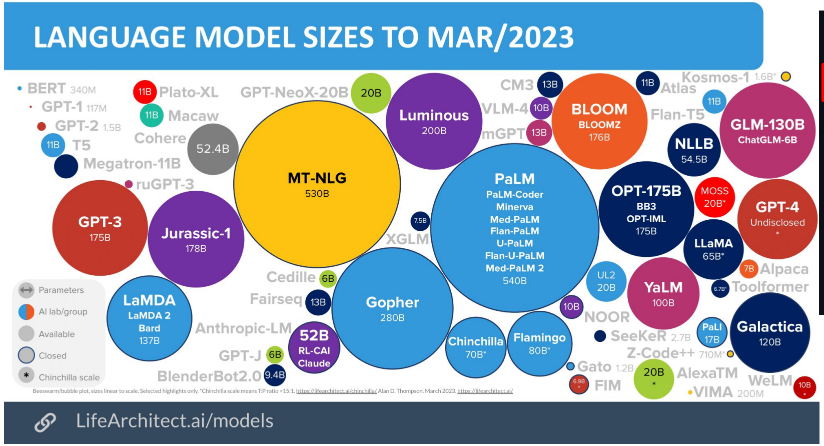
Huan Wang, final-year Ph.D. candidate at SMILE Lab, Northeastern University (Boston, USA), advised by Prof. Yun Raymond Fu.

- BE'16, MS'19 @ZJU, advised by Prof. Haoji Hu.
- Interned Google / Snap / MERL / Alibaba.
- Work on **efficient deep learning** (**pruning, distillation**) in CV & DL: **GenAI**, **3D modelling**.

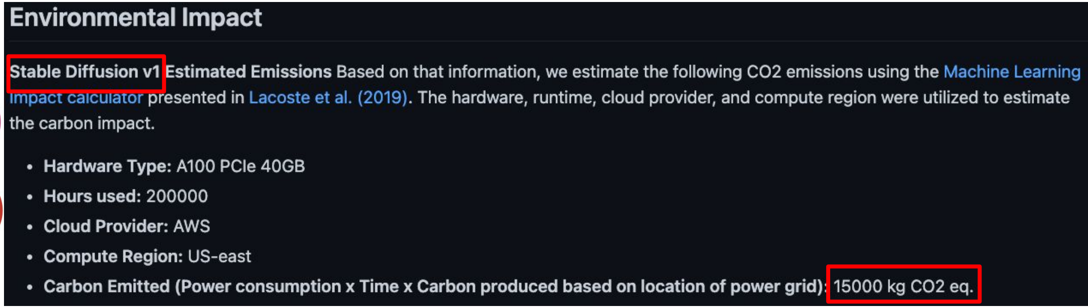
Motivation: Deep Learning Model Size is Inflating (very) Quickly

- Parameters: Millions ⇒ **Billions** (Hundreds of Billions)

- **Past (before 2020)**: Hard to deploy on resource-constrained devices (mobile, IoT, wearable devices) -- **Inference**
- **Now (after 2020)**: The rise of **Generative AI** (e.g., Stable Diffusion, ChatGPT) causes more training cost -- **Inference + Training**
  - GPT-3, 175B params, training once: tens of millions of dollars.
  - Environmental impact.



SOTA LLMs size as of 2023/03. [src]


Environmental Impact

Stable Diffusion v1 **Estimated Emissions** Based on that information, we estimate the following CO2 emissions using the Machine Learning Impact calculator presented in Lacoste et al. (2019). The hardware, runtime, cloud provider, and compute region were utilized to estimate the carbon impact.

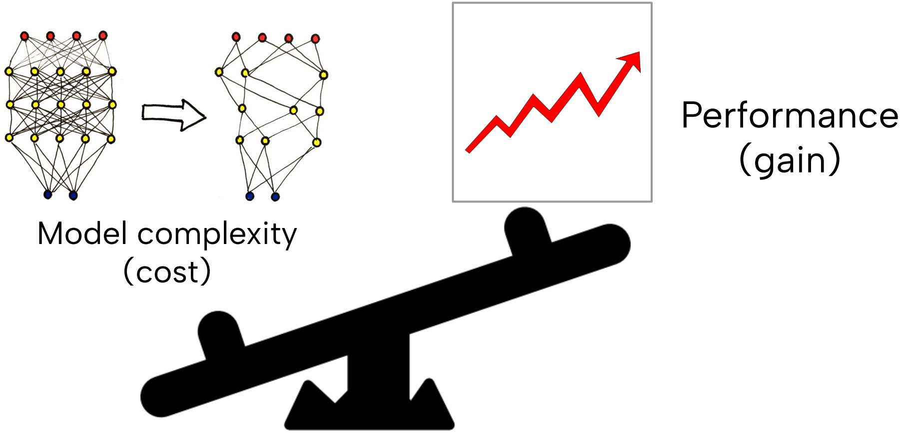
- **Hardware Type:** A100 PCIe 40GB
- **Hours used:** 200000
- **Cloud Provider:** AWS
- **Compute Region:** US-east
- **Carbon Emitted (Power consumption x Time x Carbon produced based on location of power grid):** 15000 kg CO2 eq.

Training Stable Diffusion model emits **15,000 kg of CO_2**.
src: modelcard.md - Stability-AI/stablediffusion · GitHub

⇒ Now, more than ever, the world needs ***efficient*** **deep learning**.

What is Efficient Deep Learning (EDL)?

Take away **model redundancy / complexity** while maintaining the **performance** as much as possible -- tradeoff!



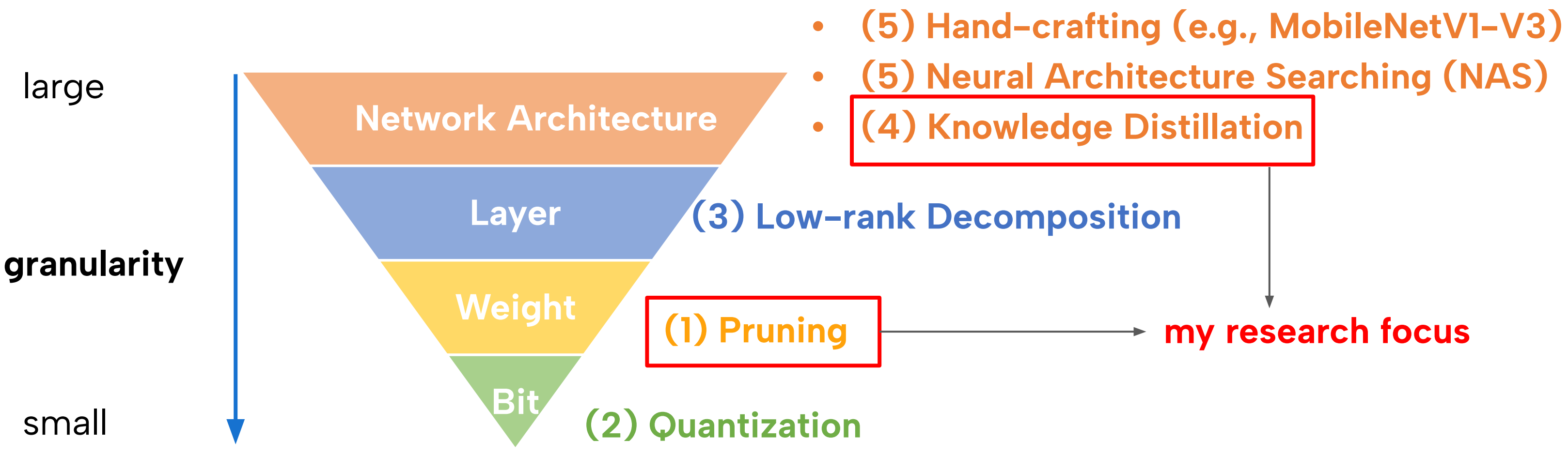
No universal EDL solution

capacity, optimization ⇒ generalization

Essentially, EDL is about neural networks, *not* specific AI tasks.

EDL = better understanding of neural networks.

The 5 Method Categories in EDL



Hierarchical Redundancy in a DNN

Compression		Compensation
Pruning	+	Distillation

Pruning + distillation: **a complete and generic pipeline** for designing efficient models.

Outline of the Talk

- Background of two EDL techniques: pruning & distillation.
- **SnapFusion** from Snap [NeurIPS'23]
- **MobileDiffusion** from Google [Arxiv]
- Summary

Background : Network Pruning

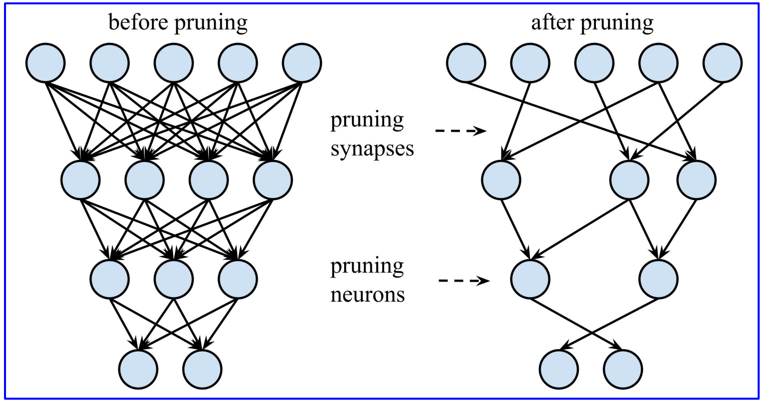


Illustration of pruning [Han et al., 2015, NeurIPS].

Pruning is probably **the earliest** mode compression method among the five.

- 1986: BP was popularized for training neural networks [Rumelhart et al., 1986, Nature].
- 1987: 1st NeurIPS conference.
- 1988: pruning papers appeared in the 2nd NeurIPS!

The Typical 3-Stage Pruning Pipeline
(practiced for 30+ years)

[Liu et al., ICLR, 2019]

(*vs.* **pruning at initialization** – not favored for foundation models.)

More Background: The 4 Key Questions in Network Pruning

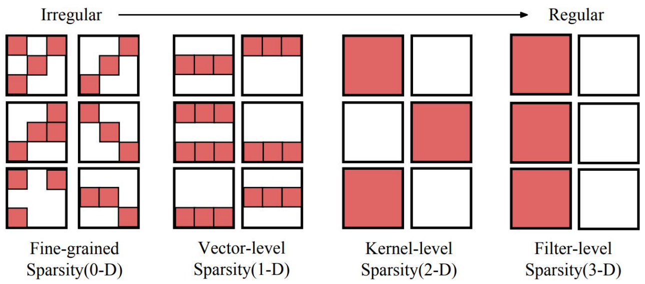
1. What to prune?
(structured *vs.* unstructured)

2. How many to prune?
(layer-wise pruning ratio)

3. Which ones to prune?
(pruning criterion)

most studied!

4. How to schedule the pruning process?
(e.g., one-shot *vs.* progressive)



Different sparsity structures.
[Mao et al., 2017, CVPRw]

Two groups of pruning methods:
- Structured pruning ⇒ acceleration
- Unstructured pruning ⇒ compression

[Wang et al., 2019, JSTSP] Huan Wang, et al. Structured Pruning for Efficient Convolutional Neural Networks via Incremental Regularization. JSTSP, 2019

Background of Knowledge Distillation (KD)

- Or called "*teacher-student learning*"
- Idea was invented in 2006 [1].
- Polished later by Hinton et al. in 2014 [2]

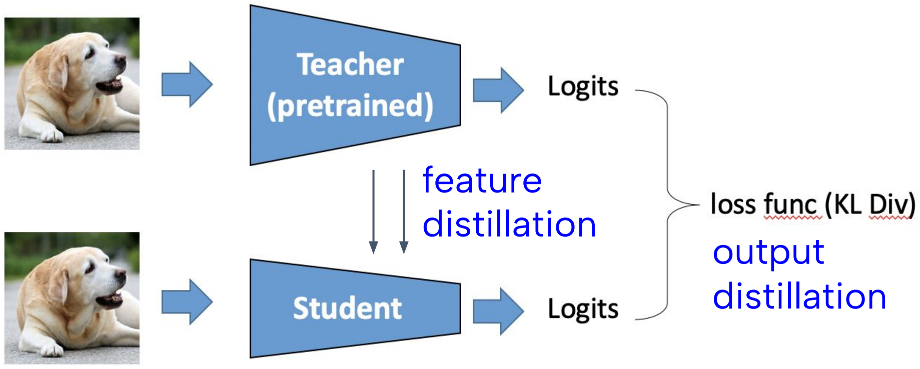


Illustration of KD

Under-explored KD Problems:

1. The KD in **3D vision / neural rendering** is very much under-explored.
2. **KD in GenAI?**
3. How KD interacts with DA has not been well understood so far – **KD+DA, theory** [Wang et al., 2022, NeurIPS] **– not covered today**

The general spirit of KD: **function matching**
Given the same input, we want the student to predict the same output as the teacher.

[1] Buciluˇa, C., Caruana, R., Niculescu-Mizil, A.: Model compression. In SIGKDD'06.
[2] Hinton, G., Vinyals, O., Dean, J.: Distilling the knowledge in a neural network. In NeurIPS Workshop'14.

“Distillation is becoming a dominant tool in deep learning”

Alex Kendall (He/Him) • Following
CEO at Wayve
1mo •

Wrapping up an amazing week at #CVPR2023! It was great to chat with the authors of many impressive research papers. Some interesting trends I observed:

(1) neural rendering methods can now handle dynamic scenes (although w huge compute requirements, but I expect will become real-time within the year). Many examples of this, such as: https://dynibar.github.io/

(2) model distillation is becoming a dominant tool in deep learning, now even enabling continual learning as model architectures change https://lnkd.in/erFA5Fi8 or turning diffusion models into single shot feed forward models https://lnkd.in/e5xaUkM4

(3) many new problems which lack comprehensive datasets are now able to be solved by tricks to train on heterogeneous datasets which only partially share modalities or label classes. A neat example was this paper learning human pose from egocentric videos: https://lnkd.in/e8tEg-6c

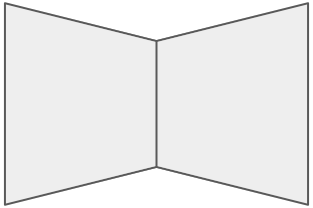
In CVPR’23, 2/12 Award Candidate Papers used distillation:

- MobileNeRF – neural rendering
- W-conditioned distillation – diffusion models / GenAI

Post on LinkedIn, credit: Alex Kendall, CEO of Wayve

“A pikachu fine dining with a view to the Eiffel Tower”

Prompt



Diffusion Model

Image

SnapFusion: Text-to-Image Diffusion Model on Mobile Devices within Two Seconds

Yanyu Li †
Snap Inc., Northeastern University

Huan Wang †
Snap Inc., Northeastern University

Qing Jin †
Snap Inc.

Ju Hu
Snap Inc.

Pavlo Chemerys
Snap Inc.

Yun Fu
Northeastern Unviersity

Yanzhi Wang
Northeastern Unviersity

Sergey Tulyakov
Snap Inc.

Jian Ren †
Snap Inc.

† Equal contribution.

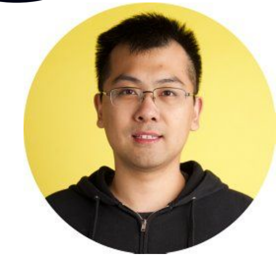
NeurIPS 2023









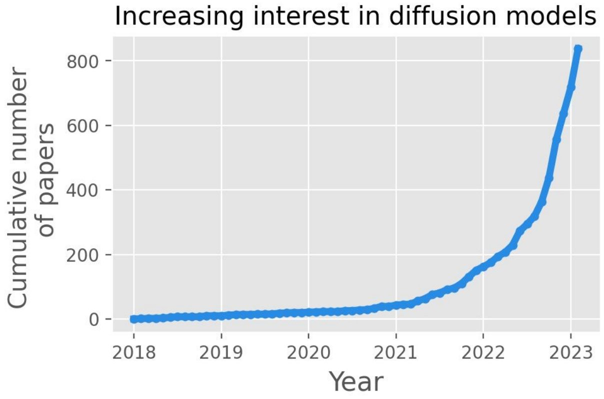


The Rise of Diffusion Models

Early pioneering works

- **2015**-ICML-Deep Unsupervised Learning using Nonequilibrium Thermodynamics (Stanford & UCB) – CIFAR10
- **2020**-NIPS-Denoising diffusion probabilistic models (UCB) – DDPM, 1st demonstration of DM generating high-quality images
- **2021**-ICLR-Denoising Diffusion Implicit Models (Stanford) – DDIM
- **2021.01**-DALL-E 1 (OpenAI)
- **2021.05**-Diffusion Models Beat GANS on Image Synthesis (OpenAI)
- **2022.04**-DALL-E 2 (OpenAI)
- **2022.05**-Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding (Google Imagen)
- **2022.08**-Stable Diffusion first release (CVPR'22, Runway + Stability AI)
- **2022.11**-eDiff-I: Text-to-Image Diffusion Models with Ensemble of Expert Denoisers (NVIDIA)

Papers exploding🔥 now!



[src: Sehwag's blog]

Prerequisites: Diffusion Model in the Generative Family

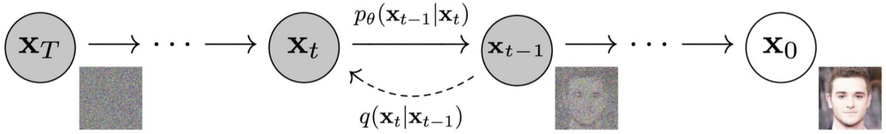
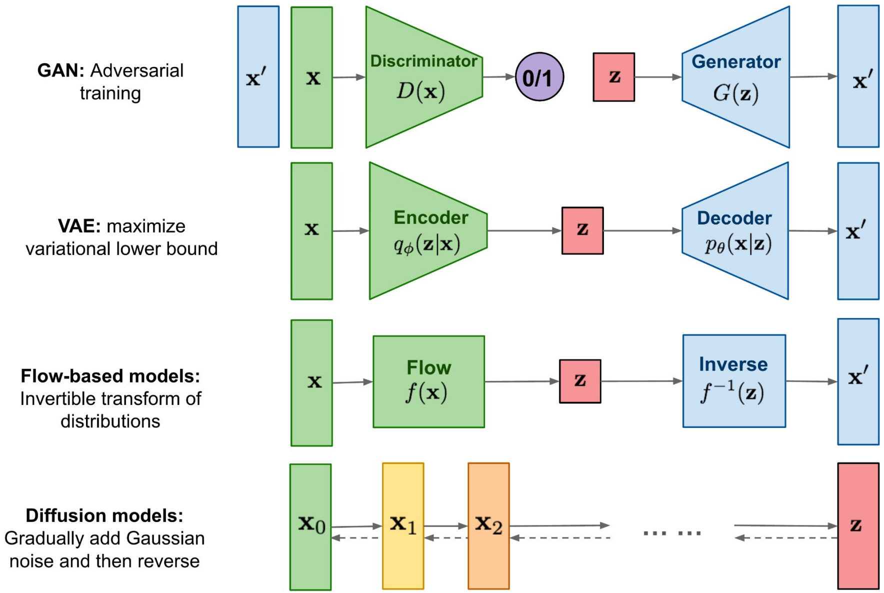


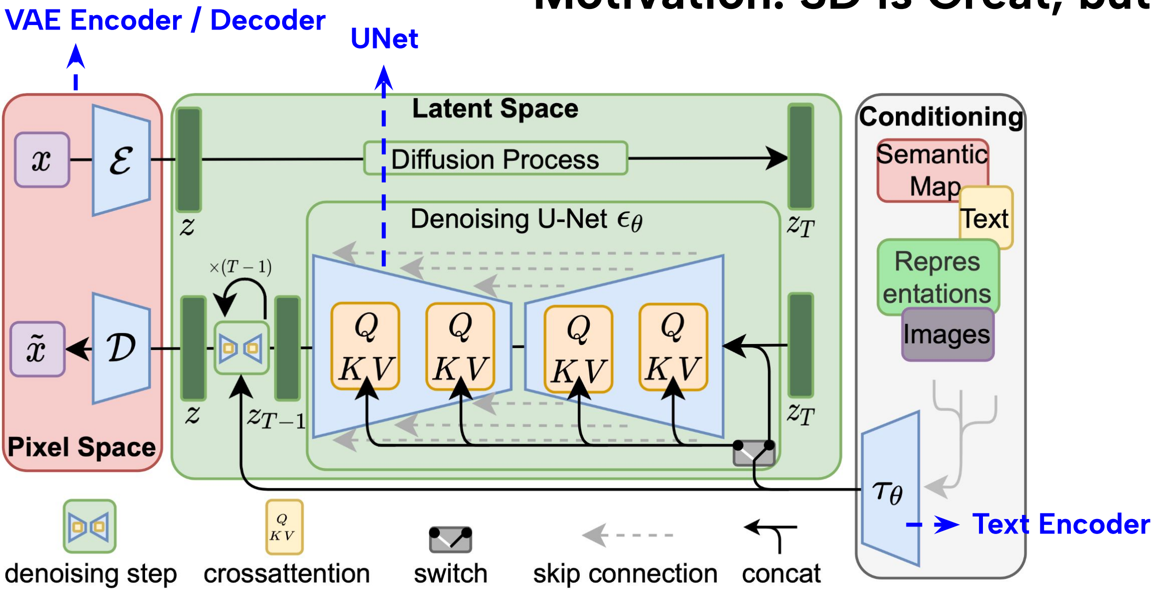
Figure 2: The directed graphical model considered in this work.

Src: DDPM [Ho et al., 2020, NeurIPS]



- Src: *What are Diffusion Models?* by **Lilian Weng** @OpenAI
- DM is featured by the **gradual (iterative) diffusion** and denoising process.
- DM: Feature or latent (z) has the same shape as the input (x).

Motivation: SD is Great, but Huge and Slow



Overview of LDM / SD [Rombach et al., 2022, CVPR]

- **Huge model size** (fp16, in CoreML), 1B params:
  - Text encoder: 246.3 MB
  - **UNet: 1.7 GB**
  - Image Decoder: 99.2 MB

- **Prohibitively slow** (in CoreML):
  - Text encoder: 4.2 ms
  - UNet: unable to profile as one, chunked into 2 parts: ~1.7 ms
  - VAE Decoder: 370.66 ms

Naively run SD on iOS: 1~2mins!

3 parts:

- Text encoder (from CLIP, frozen) -- input prompt
- UNet (key!) -- iterative denoising
- VAE encoder/decoder (frozen) -- generate image
- Inference: z0=noise, c=TextEnc(prompt) ⇒ **z'=UNet(t, z, c)** (iterative) ⇒ img=VAEDec(z).

Early attempts for efficient on-device SD (Qualcomm & Google)

OnQ Blog

SD-v1.4, 15s, via full-stack AI optimization

World's first on-device demonstration of Stable Diffusion on an Android phone

Qualcomm AI Research deploys a popular 1B+ parameter foundation model on an edge device through full-stack AI optimization

FEB 23, 2023 | Snapdragon and Qualcomm branded products are products of Qualcomm Technologies, Inc. and/or its subsidiaries.

Early attempts for efficient on-device SD (Qualcomm & Google)

Google Research   Philosophy   Research Areas   Publications   People   Resources   Outreach

SD-v1.5, 12s, via mobile GPU optimization

BLOG ›

Speed is all you need: On-device acceleration of large diffusion models via GPU-aware optimizations

THURSDAY, JUNE 15, 2023

Posted by Juhyun Lee and Raman Sarokin, Software Engineers, Core Systems & Experiences

The proliferation of large **diffusion models** for **image generation** has led to a significant increase in model size and inference workloads. On-device ML inference in mobile environments requires meticulous performance optimization and consideration of trade-offs due to resource constraints. Running inference of large diffusion models (LDMs) on-device, driven by the need for cost efficiency and user privacy, presents even greater challenges due to the substantial memory requirements and computational demands of these models.

More of engineering optimizations – not change the UNet arch., not optimize loss, no new training pipeline ⇒ **SnapFusion will optimize all these aspects.**

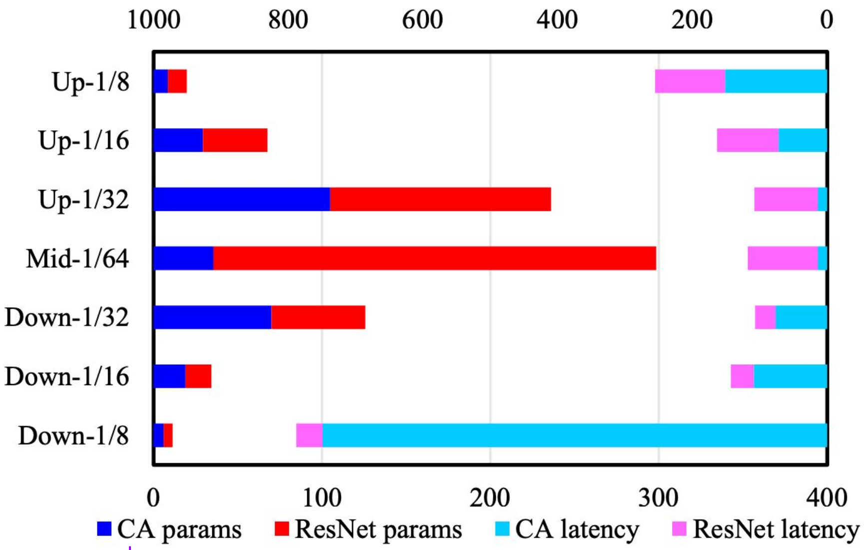
Profiling – Where is the Speed Bottleneck?

Stable Diffusion v1.5	Text Encoder	UNet	VAE Decoder
Input Resolution	77 tokens	64×64	64×64
#Parameters (M)	123	860	50
Latency (ms)	4	~1,700*	369
Inference Steps	2	50	1
Total Latency (ms)	8	85,000	369
Our Model	Text Encoder	**Our UNet**	**Our Image Decoder**
Input Resolution	77 tokens	64×64	64×64
#Parameters (M)	123	**848**	**13**
Latency (ms)	4	**230**	**116**
Inference Steps	2	**8**	1
Total Latency (ms)	8	**1,840**	**116**

Wanna accelerate SD? Two paths!

- Reduce single inference cost – Architecture efficiency
- Reduce #inference steps – Samping efficiency

Profiling – Where is the Speed Bottleneck?
(more fine-grained examination)



A typo: Should be **Attention** (including Self-Attn and Cross-Attn)

Cost inside UNet

Attn: small #params, huge #latency!

complexity of Attn: **O(feature map size^2)**

Methodology (1) – Efficient UNet

Algorithm 1 Optimizing UNet Architecture

Require: UNet: $\hat{\boldsymbol{\epsilon}}_{\boldsymbol{\theta}}$; validation set: $\mathbb{D}_{val}$; latency lookup table $\mathbb{T}$: $\{Cross\text{-}Attention[i, j], ResNet[i, j]\}$.
Ensure: $\hat{\boldsymbol{\epsilon}}_{\boldsymbol{\theta}}$ converges and satisfies latency objective S.
while $\hat{\boldsymbol{\epsilon}}_{\boldsymbol{\theta}}$ not converged **do**
  Perform robust training.
  $\rightarrow$ **Architecture optimization**:
  if perform architecture evolving at this iteration **then**
    $\rightarrow$ **Evaluate blocks**:
    for each $block[i, j]$ **do**
      $\boxed{\Delta CLIP} \leftarrow \text{eval}(\hat{\boldsymbol{\epsilon}}_{\boldsymbol{\theta}}, A^{-}_{block[i,j]}, \mathbb{D}_{val})$,
      $\boxed{\Delta Latency} \leftarrow \text{eval}(\hat{\boldsymbol{\epsilon}}_{\boldsymbol{\theta}}, A^{-}_{block[i,j]}, \mathbb{T})$
    end for
    $\rightarrow$ **Sort actions based on** $\boxed{\frac{\Delta CLIP}{\Delta Latency}}$**, execute action, and evolve architecture to get latency** T:
    if T not satisfied **then**
      $\{\hat{A}^{-}\} \leftarrow \arg\min_{A^{-}} \frac{\Delta CLIP}{\Delta Latency}$,
    else
      $\{\hat{A}^{+}\} \leftarrow \text{copy}(\arg\max_{A^{-}} \frac{\Delta CLIP}{\Delta Latency})$,
    $\hat{\boldsymbol{\epsilon}}_{\boldsymbol{\theta}} \leftarrow \text{evolve}(\hat{\boldsymbol{\epsilon}}_{\boldsymbol{\theta}}, \{\hat{A}\})$
    end if
  end if
end while

We propose an **Automatic Architecture Evolving** Algorithm (General idea: **remove** the **unimportant** modules and **add** the **important** ones.)

smaller is better

$$\text{Unimportance Score} = \frac{\Delta CLIP}{\Delta Latency}$$

larger is better

Efficient UNet

Methodology (1) - Efficient UNet (Final Arch.)

Table 3: Detailed architecture of our efficient UNet model.

Stage	Resolution	Type	Config	UNet Model Origin	UNet Model Ours
Down-1	$\frac{H}{8} \times \frac{W}{8}$	Cross Attention	Dimension	320	
			# Blocks	2	0
		ResNet	Dimension	320	
			# Blocks	2	2
Down-2	$\frac{H}{16} \times \frac{W}{16}$	Cross Attention	Dimension	640	
			# Blocks	2	2
		ResNet	Dimension	640	
			# Blocks	2	2
Down-3	$\frac{H}{32} \times \frac{W}{32}$	Cross Attention	Dimension	1280	
			# Blocks	2	2
		ResNet	Dimension	1280	
			# Blocks	2	1
Mid	$\frac{H}{64} \times \frac{W}{64}$	Cross Attention	Dimension	1280	
			# Blocks	1	1
		ResNet	Dimension	1280	
			# Blocks	7	4
Up-1	$\frac{H}{32} \times \frac{W}{32}$	Cross Attention	Dimension	1280	
			# Blocks	3	3
		ResNet	Dimension	1280	
			# Blocks	3	2
Up-2	$\frac{H}{16} \times \frac{W}{16}$	Cross Attention	Dimension	640	
			# Blocks	3	6
		ResNet	Dimension	640	
			# Blocks	3	3
Up-3	$\frac{H}{8} \times \frac{W}{8}$	Cross Attention	Dimension	320	
			# Blocks	3	0
		ResNet	Dimension	320	
			# Blocks	3	3

- Remove the Cross-Attention module at high resolution (the 1st downsample and last upsample).
- Add more modules for the upsample stage (Up-2).

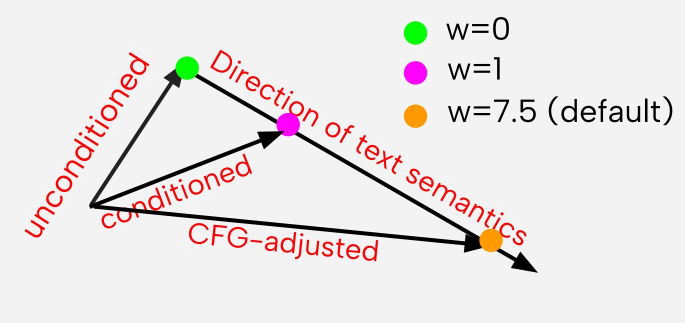
7.4x speedup! *vs.* SD-v1.5

Methodology (2) – CFG-Aware Step Distillation (a new loss)

What is CFG? ("classifier-free guidance")

- A trick used to improve image quality (**for enhancing text semantics**).

How CFG works? A simple illustration.

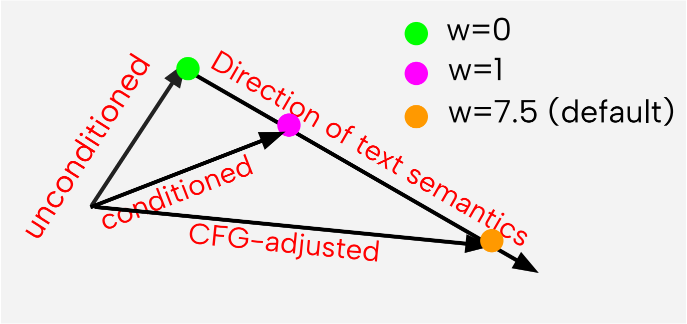


Methodology (2) - CFG-Aware Step Distillation (a new loss)

CFG ("classifier-free guidance")

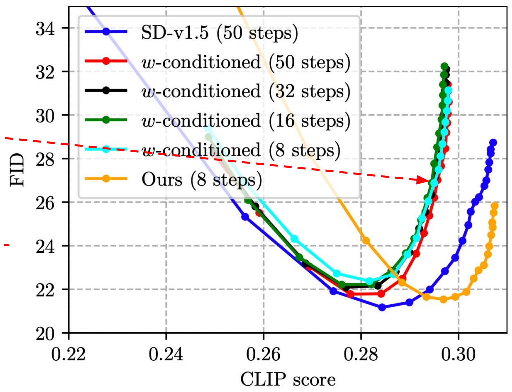
- A trick used to improve image quality (**for enhancing text semantics**).

How CFG works? A simple illustration.



- **Problem / Motivation**: CFG is used in inference, *not in distilled training* ⇒ Student is CFG-unaware.

- **Solution**: We propose to apply CFG to the student during step distillation ⇒ Student is CFG-aware.



Other Optimizations?

The major contributions are two:

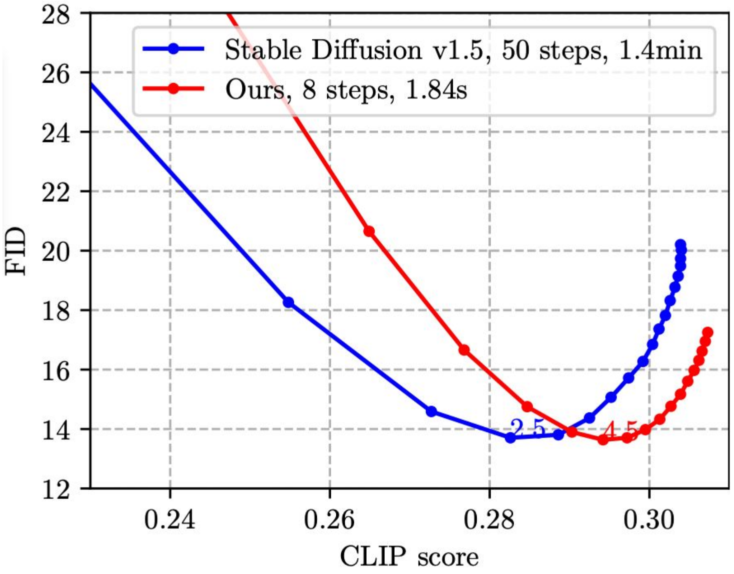
- Efficient UNet
- CFG-aware Distillation, as presented above.

Please refer to the paper for more:

- Efficient VAE **decoder** via **structured pruning** (L1-n
  pruning).
- **Training pipeline**. E.g., which teacher is used for distilling the 8-step student?

Stable Diffusion v1.5	Text Encoder	UNet	VAE Decoder
Input Resolution	77 tokens	64 × 64	64 × 64
#Parameters (M)	123	860	50
Latency (ms)	4	~1,700*	369
Inference Steps	2	50	1
Total Latency (ms)	8	85,000	369
Our Model	Text Encoder	**Our UNet**	**Our Image Decoder**
Input Resolution	77 tokens	64 × 64	64 × 64
#Parameters (M)	123	**848**	**13**
Latency (ms)	4	**230**	**116**
Inference Steps	2	**8**	1
Total Latency (ms)	8	**1,840**	**116**

Experimental Results

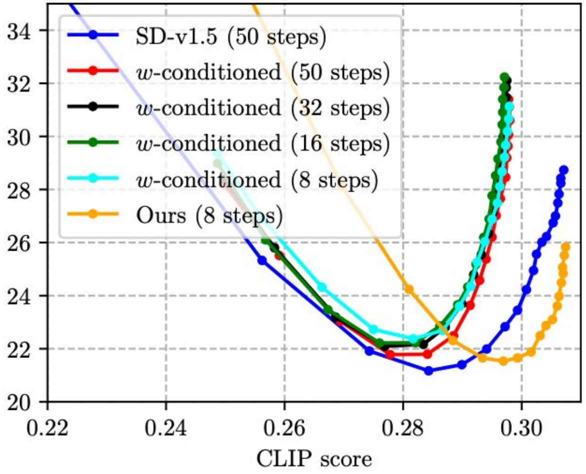


Ours vs. original SD-v1.5: **Better quality, and 46x faster!**

SnapFusion is the 1st mobile SD model that can run text-to-image generation <2s!

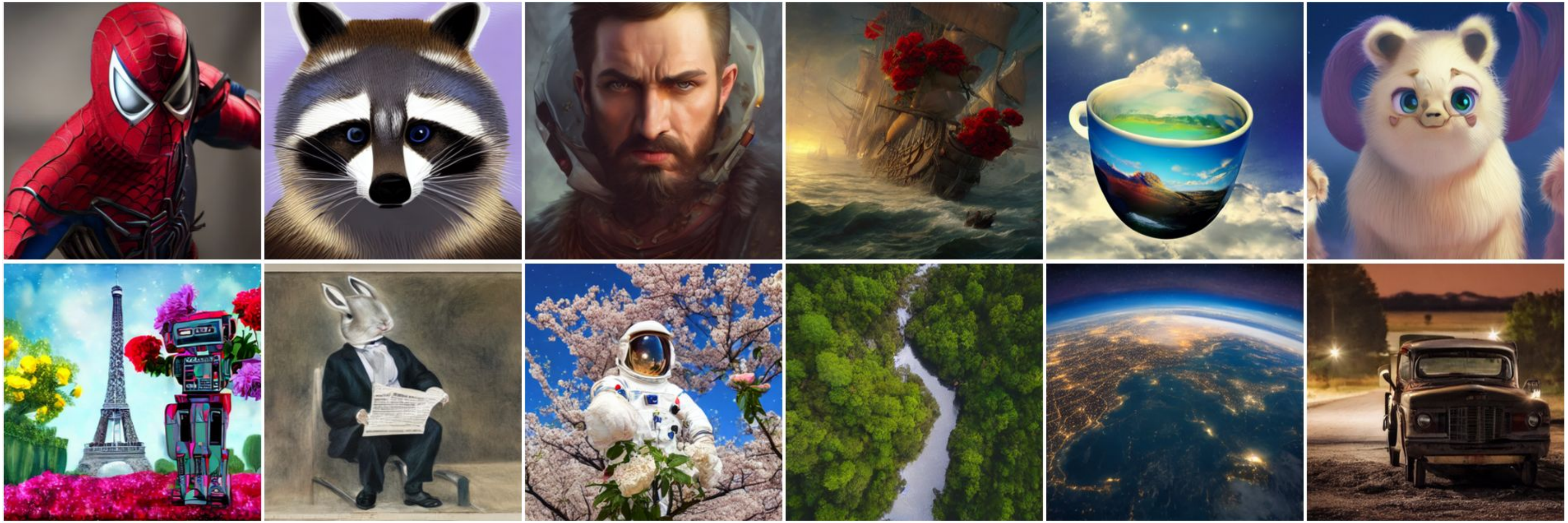
Method	Steps	FID	CLIP
DPM (Lu et al., 2022a)	8	31.7	0.32
DPM++ (Lu et al., 2022b)	8	25.6	0.32
Meng *et al.* (Meng et al., 2023)	8	26.9	0.30
Ours	8	**24.2**	0.30

Zero-shot evaluation on MS-COCO 2017 5K subset.



Comparison to w-conditioning [Meng et al., CVPR, 2023] 1/12 Award Candidates

Examples of Generated Images



(See more results in the Appendix of the paper on arxiv)

SnapFusion is becoming a part in Snapchat, used by hundreds of millions of users.

video demo, iPhone14 Pro.

More Recent Works – MobileDiffusion from Google

MobileDiffusion: Subsecond Text-to-Image Generation on Mobile Devices

Yang Zhao, Yanwu Xu, Zhisheng Xiao, Tingbo Hou
Google
{yzhaoeric, yanwuxu, zsxiao, tingbo}@google.com

(a) MobileDiffusion, distilled 8 steps

(b) MobileDiffusion, finetuned 1 step

512x512, 0.2s on iPhone15 Pro! Amazing!

[arXiv:2311.16567]

Like SnapFusion, they optimize in two axes: Architecture & Sampling

Attention Modules

1. More transformers in the middle of U-Net & less channels.
   ⇒ 26% efficiency improvement, no quality drop!
2. Decouple SA from CA ⇒ 15% efficient improvement
3. Share key-value projections ⇒ 5% params. reduction.
4. Replace gelu with swish - gelu is unstable for low-bits.
5. Finetune softmax into relu in Attention. ⇒ More efficient.
6. Trim feed-forward layers ⇒ 10% params reduction.

Conv Modules

1. Separable convolution ⇒ ~10% params. reduction
2. Prune redundant residual blocks ⇒ 19% efficiency improvement, 15% params reduction.

⇒

- "Bag of tricks"
- More fine-grained optimization than SnapFusion.

Sampling: Build upon prior works: cfg-aware distillation (8-step) and UFOGen [1] (1-step)

[1] Xu, Yanwu, et al. "Ufogen: You forward once large scale text-to-image generation via diffusion gans." arXiv preprint arXiv:2311.09257 (2023).

Efficiency Comparison of MD

Models	#Channels	#ConvBlocks	#(SA+CA)	#Params(M)	#GFLOPs	Latency(ms)	Model Size (GB)
SD-XL [36]	(320, 640, 1280)	17	31+31	2,300	710	29.5	5.66
SD-1.4/1.5	(320, 640, 1280, 1280)	22	16+16	862	392	21.7	2.07
SnapFusion [23]	(320, 640, 1280, 1280)	18	14+14	848	285	15.0	1.97
MobileDiffusion	(320, 640, 1024)	11	15+18	386	182	9.9	1.04
MobileDiffusion-Lite	(320, 640, 896)	11	12+15	278	153	8.8	0.82

Table 1. Comparison with other recognized latent diffusion models. Latency and GFLOPs, computed with jit per forward step, are measured for an input latent size of 64×64 on TPU v3. Model size (fp16) includes all, *i.e.*, UNet, text encoder and VAE decoder.

Models	**Text Encoder**	**Decoder**	**UNet**	**Steps**	**Overall**
SnapFusion [23][3]	4	116	230	8	1960
UFOGen	4	285	1580	1	1869
MD	4	92	142	8 1	1232 238
MD-Lite	4	92	123	1	219

Table 5. On-device latency (ms) measurements.

Compared to SD-v1.5:

- **~2x faster**
- **~2x smaller**

~1.6x faster than SnapFusion (8 steps)

Quantitatively, 8-step MD ≈ SD-v1.5, 1-step MD < SD-v1.5

Models	Sample	#Steps	FID-30K↓	#Params(B)	#Data(B)
GigaGAN [18]	GAN	1	9.09	0.9	0.98
LAFITE [62]	GAN	1	26.94	0.23	0.003
Parti [59]	AR	-	7.23	20.0	5.00
DALL·E-2 [38]	DDPM	292	10.39	5.20	0.25
GLIDE [34]	DDPM	250	12.24	5.00	0.25
Imagen [42]	DDPM	256	7.27	3.60	0.45
SD [39]	DDIM	50	8.59	0.86	0.60
SnapFusion [23]	Distilled	8	13.5	0.85	-
PIXART-α [4]	DPM	20	10.65	0.6	0.025
BK-SDM [21]	DDIM	50	16.54	0.50	-
SD-replicated[1]	DDIM	50	8.43	0.86	0.15
MD	DDIM	50	8.65	0.40	0.15
	Distilled	8	9.01		
	UFOGen	1	11.67		
MD-Lite	DDIM	50	9.45	0.26	0.15
	Distilled	8	9.87		
	UFOGen	1	12.89		

Table 4. Quantitative evaluations on zero-shot MS-COCO.

Some samples of MD

SD-1.5(865M)	**MD-Lite (278M)**	**MD (386M)**	**MD (386M)**	**MD (386M)**
DDIM 50 steps	DDIM 50 steps	DDIM 50 steps	Distilled 8 steps	UFOGen 1 step

A sunflower wearing sunglasses

A robot painted as graffiti on a brick wall. a sidewalk is in front of the wall, and grass is growing out of cracks in the concrete.

Summary: Towards Efficient Mobile DMs

- Two major efficiency paths: **Architecture** & **Sampling**.

- **Architecture**: **Hand-design** or **search** or **pruning** – hardware/system oriented
  a. SnapFusion: Coarse-grained
  b. MobileDiffusion: Fine-grained

- **Sampling**: Few-step **distillation** or one-step fine-tuning. – algorithm oriented
  a. SnapFusion: cfg-aware distillation (8-step)
  b. MobileDiffusion: cfg-aware distillation (8-step), UFOGen (1-step)

Take-aways:
1. Do profiling!
2. Hardware-algorithm co-design
3. No panacea – "bag of tricks"

Thanks!
Questions?