# Accelerating Stable Diffusion-based Video Generation

程煜格 22/08/2024

# Accelerating Stable Diffusion-based Video Generation

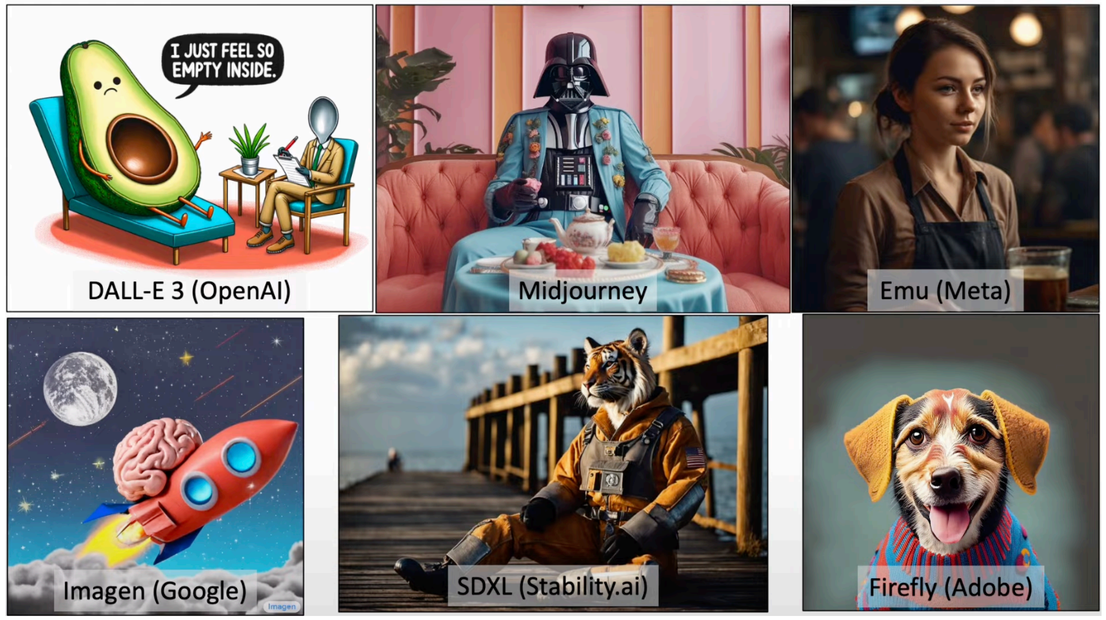
## Outlines

- Foundations - diffusion model & stable diffusion
- **Accelerations**
  - (ISCA’24) Cambricon-D: Full-Network Differential Acceleration for Diffusion Models
  - (CVPR’24) DeepCache: Accelerating Diffusion Models for Free
- Open-sourced video generation models
  - Open-sora: spatial and temporal attention

# Foundations

## Diffusion models

- What can diffusion models do?
  - Image generation




  - Video generation

“A flock of paper airplanes flutters through a dense jungle, weaving around trees as if they were migrating birds. ”

  - And more!

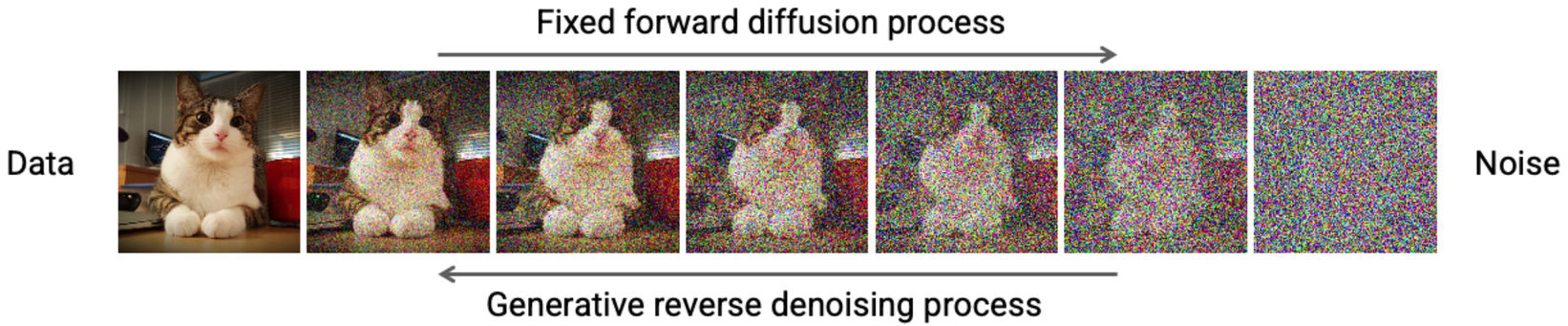
https://openai.com/index/sora/

# Foundations

## Diffusion models

Diffusion models consist of two processes:

- **Forward** diffusion process that gradually adds noise to input
- **Reverse** denoising process that learns to generate data by denoising

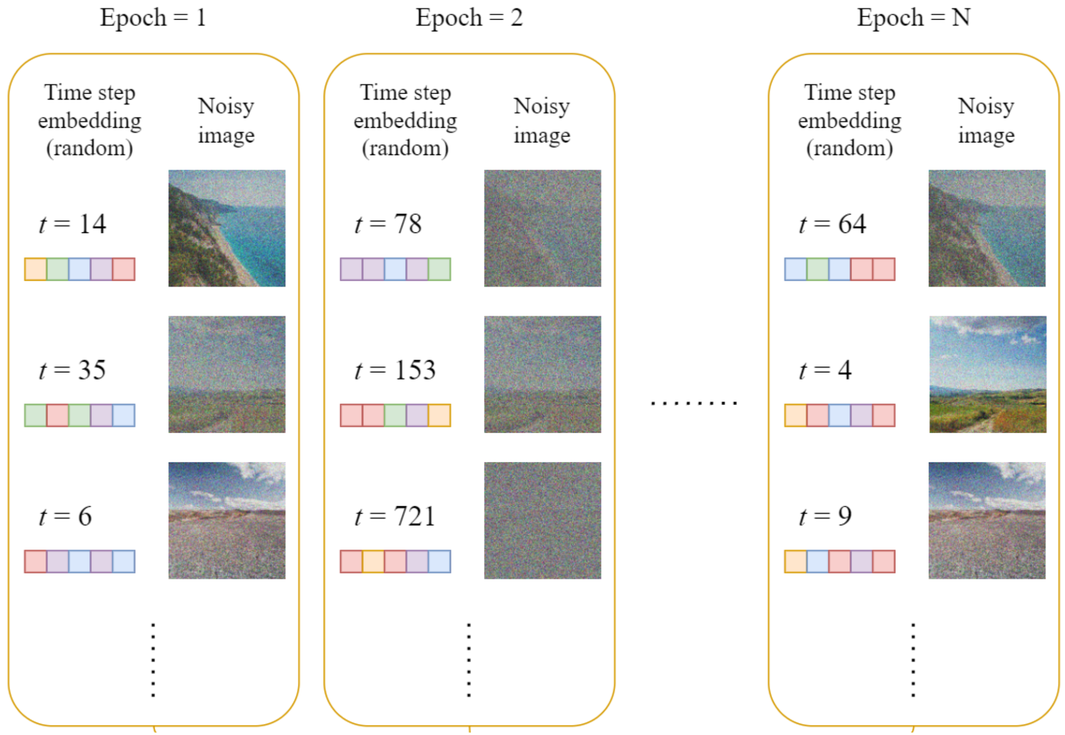


# Foundations

## Diffusion models

**1. Forward process (training)**: **predicts the noise**, given the current timestep **t** and image **Xt**

Each diffusion step is called a *timestep*.



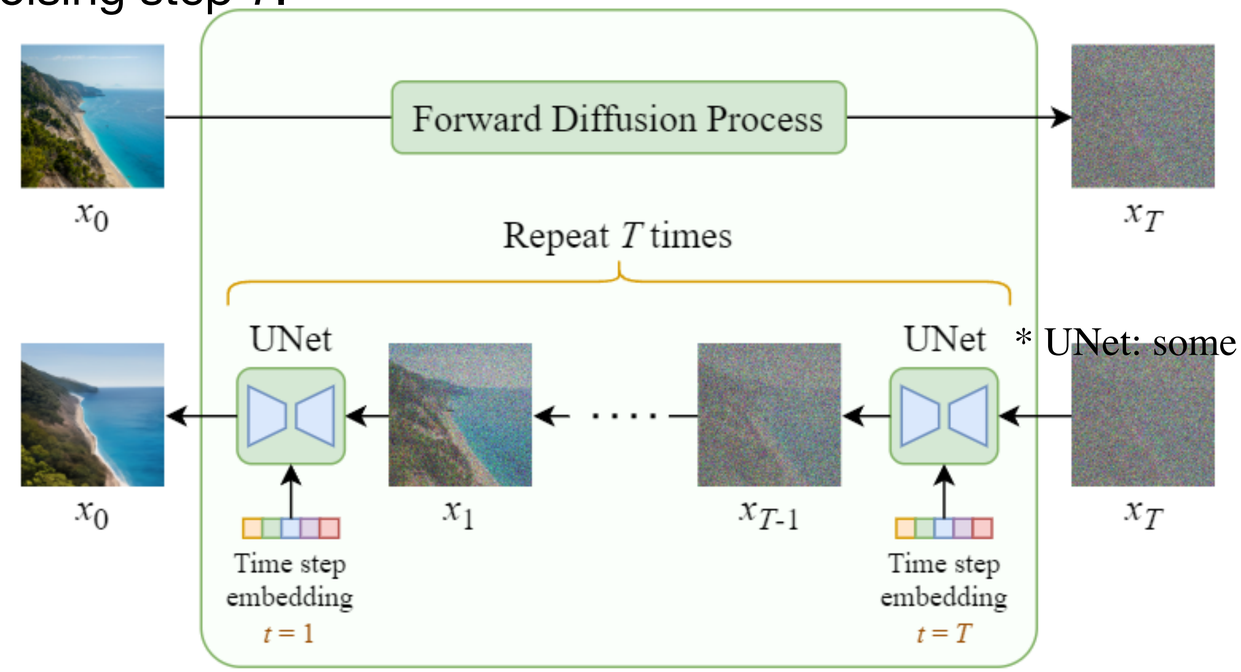

https://medium.com/@steinsfu/stable-diffusion-clearly-explained-ed008044e07e

# Foundations

## Diffusion models

**2. Backward process (inferencing)**: denoises step-by-step to **generate samples**, given the pure noise **$x_T$**

First define the total denoising step $T$.



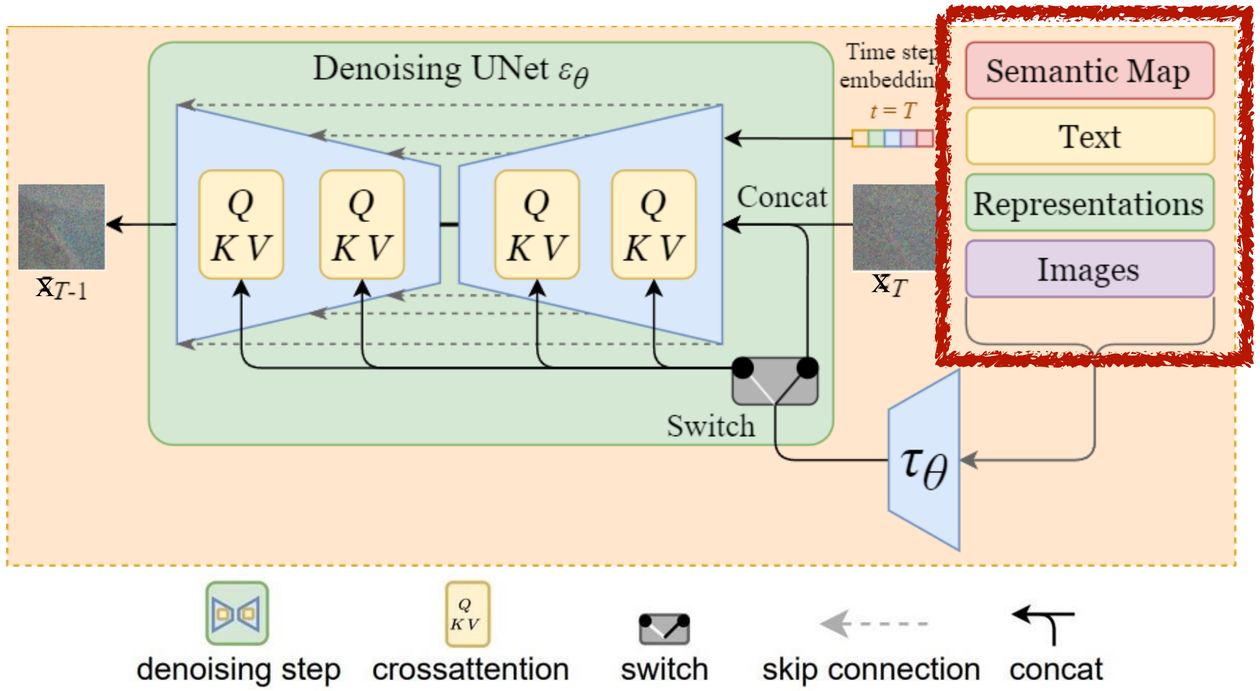

* UNet: some Convolutional Neural Networks (CNN)

https://medium.com/@steinsfu/stable-diffusion-clearly-explained-ed008044e07e

# Foundations

## Diffusion models

Conditioning




e.g. “A smiling cat.”

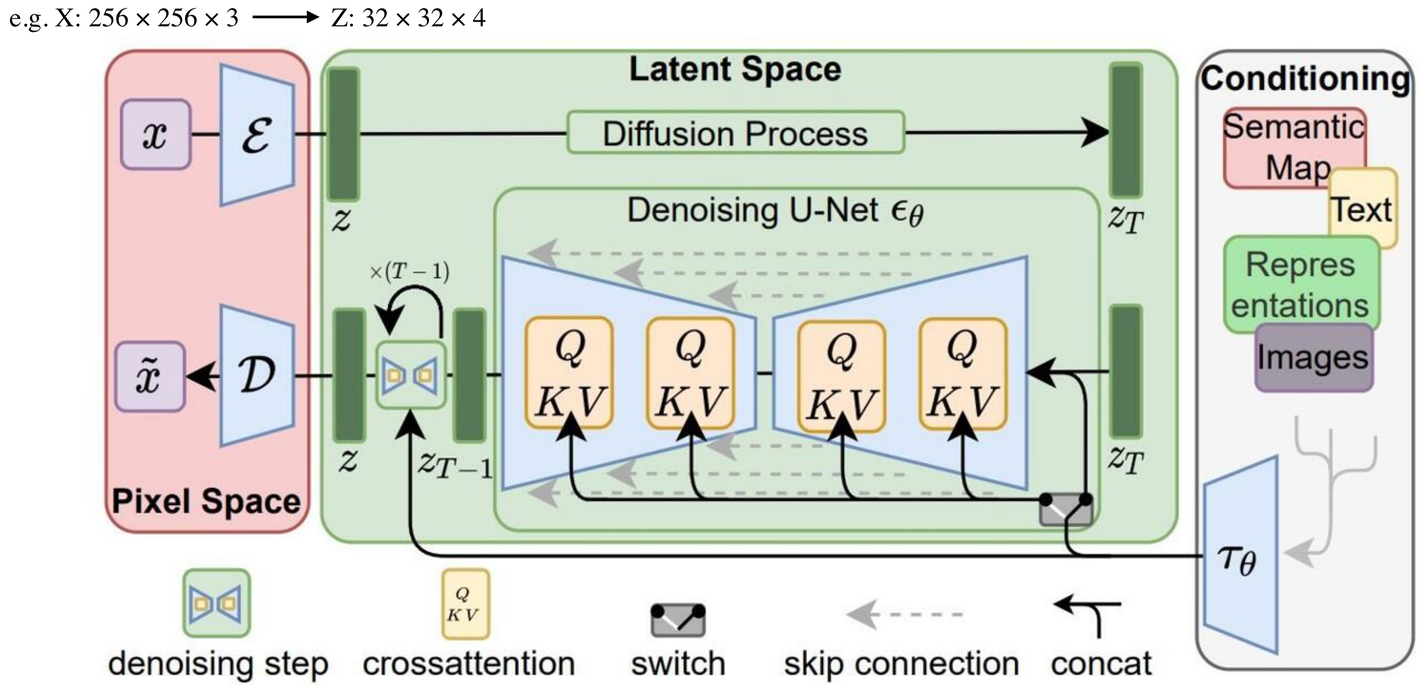


https://cvpr2022-tutorial-diffusion-models.github.io/

# Foundations

## Stable Diffusion / Latent Diffusion Models

The diffusion process happens within the **latent space (a lower-dimensional representation of data).**

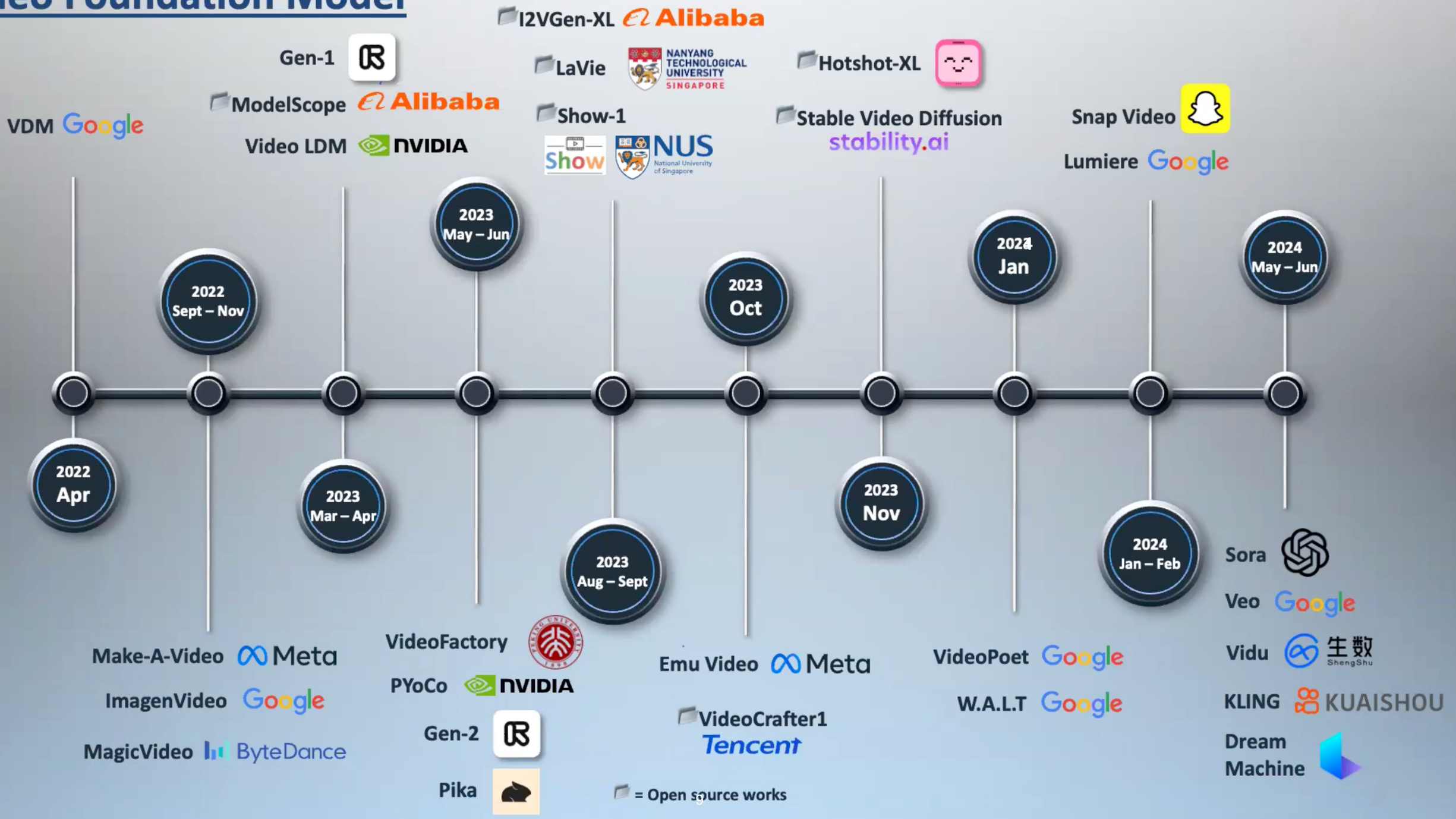



Everything else stays the same :D



https://medium.com/@steinsfu/stable-diffusion-clearly-explained-ed008044e07e

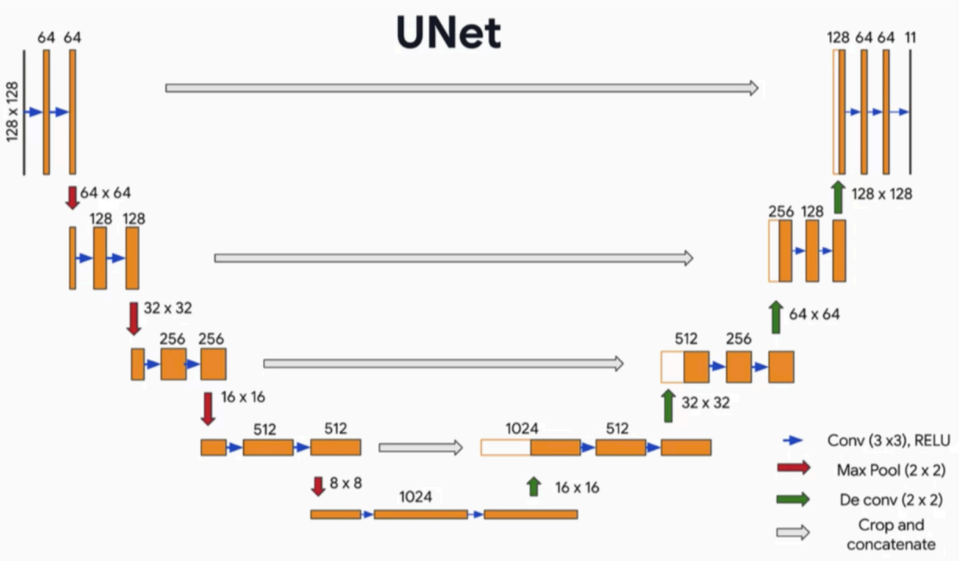
# Video Foundation Model



# Overview

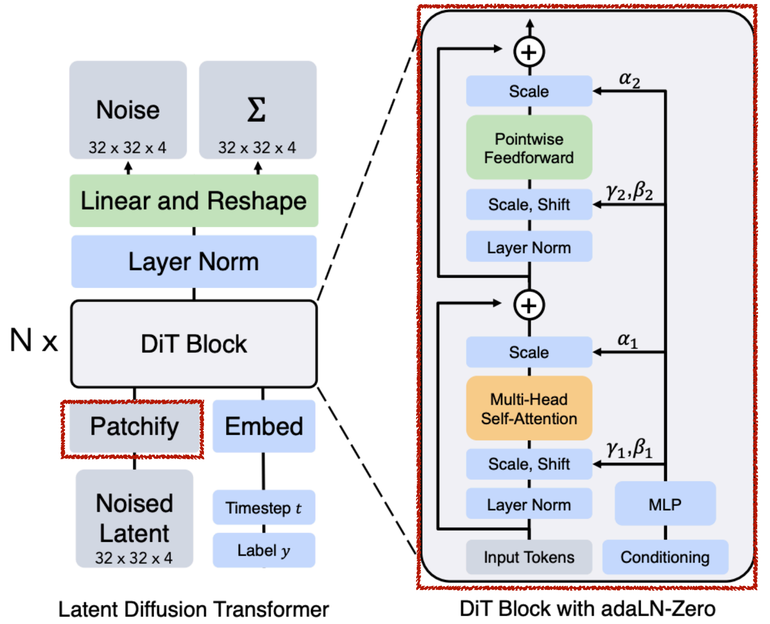
## Stable diffusion models evolution

- Two main backbones used




The diffusion process is performed **on image latent**.

**Transformer**




The diffusion process is performed **on image latent tokens**.

# Acceleration of Diffusion Models Inference

## Cambricon-D

full-network differential acceleration with concise memory access

**Observation**: input data is slightly altered across timesteps.

Therefore, the iterative process of diffusion models implies a remarkable computational redundancy.

(a) (b) (c)

Example of two images in neighboring timesteps and their delta. Pixels with no difference are grey (128,128,128).

Kong, Weihao, et al. "Cambricon-D: Full-Network Differential Acceleration for Diffusion Models." *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2024.

# Acceleration of Diffusion Models Inference

## Cambricon-D

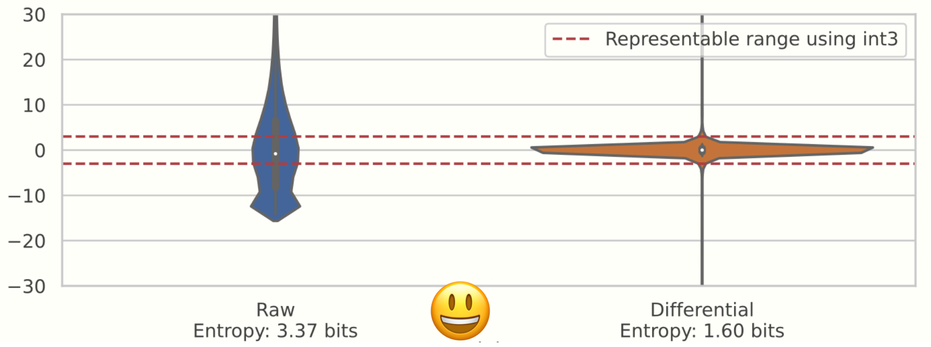
full-network differential acceleration with concise memory access

**Observation**: input data is slightly altered across timesteps.

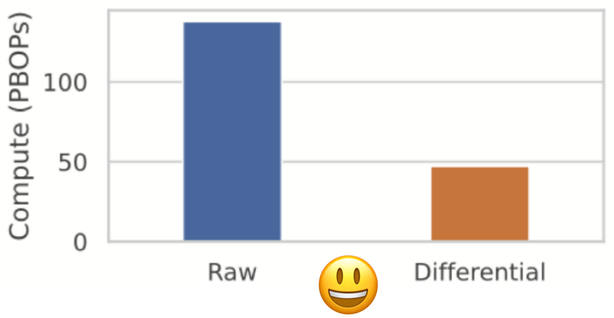
Therefore, the iterative process of diffusion models implies a remarkable computational redundancy.

**Differential computing**: focuses the computation on these input variations (aka. deltas).

- leveraging the narrower data range of deltas (compared to raw input) -> representation with fewer bits




Distributions and entropy of raw and delta input values for a typical layer. Raw activations often fall outside quantizable range.




Execution time reduction from using differential computing compared to raw valued computing.

# Acceleration of Diffusion Models Inference

## Cambricon-D

full-network differential acceleration with concise memory access

**Observation**: input data is slightly altered across timesteps.

Therefore, the iterative process of diffusion models implies a remarkable computational redundancy.

$$delta^t_{out} \neq f_{non-linear}(delta^t_{in})$$

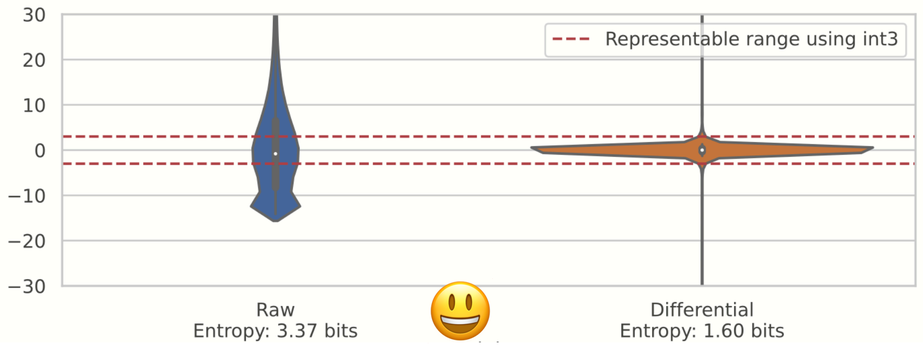
$$raw^t_{out} = f_{non-linear}(raw^{t-1}_{in} + delta^t_{in})$$

$$raw^t_{in}$$

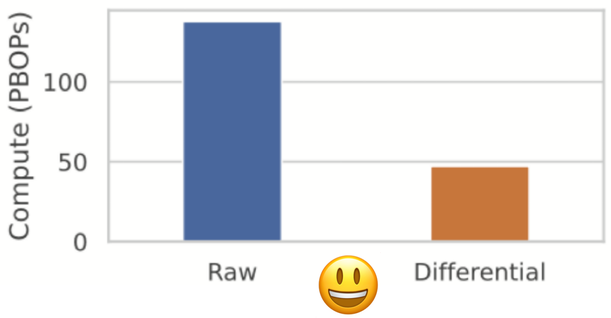
**Differential computing**: focuses the computation on these input variations (aka. deltas).

- leveraging the narrower data range of deltas (compared to raw input) -> representation with fewer bits

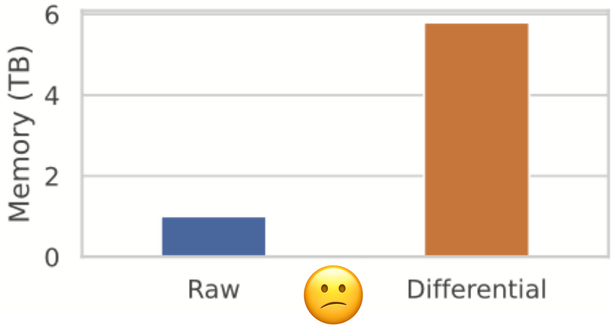
$$delta^t_{out} = raw^t_{out} - raw^t_{in}$$




Distributions and entropy of raw and delta input values for a typical layer. Raw activations often fall outside quantizable range.




Execution time reduction from using differential computing compared to raw valued computing.




Memory traffic overhead from differential computing compared to raw valued computing.

# Acceleration of Diffusion Models Inference

## Cambricon-D

full-network differential acceleration with concise memory access

**Solutions**:

- a sign-mask dataflow, achieving an efficient *full-network* differential computation
- an outlier-aware PE array, dealing with delta outliers effectively

# Acceleration of Diffusion Models Inference

## Cambricon-D

full-network differential acceleration with concise memory access

**1. The sign-mask dataflow**

- Approximates the non-linear ReLU function **using the narrower delta values**

$\text{sgn}(Y_t) \approx \text{sgn}(Y_{t-1})$ because $\Delta Y_t$ is relatively small.

$$\text{ReLU}(Y_t) = \begin{cases} 0, & \text{if } Y_t < 0 \\ Y_t, & \text{if } Y_t \geq 0 \end{cases} = Y_t \cdot \text{sgn}(Y_t) \qquad (1)$$

$$\begin{aligned} \Delta Y'_t &= \text{ReLU}(Y_t) - \text{ReLU}(Y_{t-1}) \\ &= Y_t \cdot \text{sgn}(Y_t) - Y_{t-1} \cdot \text{sgn}(Y_{t-1}) \qquad (2) \\ &\approx \Delta Y_t \cdot \text{sgn}(Y_{t-1}) \qquad (3) \end{aligned}$$

$$\text{where} \quad Y_t := Y_{t-1} + \Delta Y_t.$$

- **Key insight: only read the sign bits of the tensor sgn(Y)** from the off-chip memory, instead of the entire raw tensor Y

# Acceleration of Diffusion Models Inference

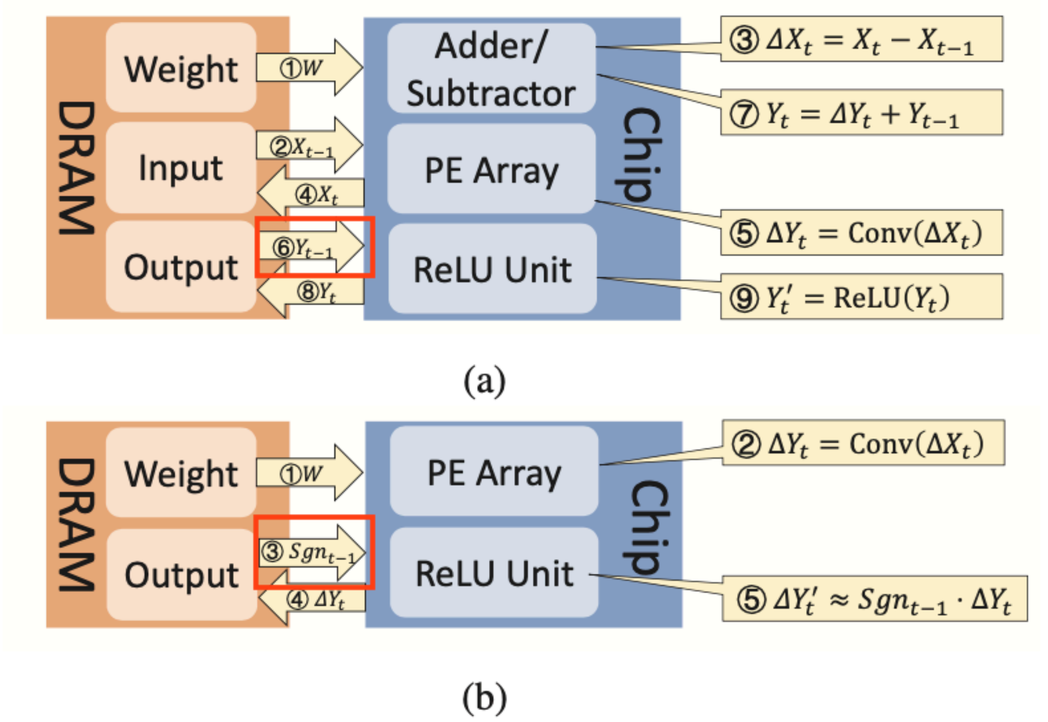
## Cambricon-D

full-network differential acceleration with concise memory access

**The sign-mask dataflow consists of mainly 5 steps:**

1) Load weights W
2) Compute tensor multiplication (using W and delta_X)
3) Load sign bits
4) Store output delta_Y
5) Compute ReLU with sign masking

A **full-network differential** methed!




Comparison of dataflows of Diffy (a) and the sign-mask dataflow of Cambricon-D (b).

Mahmoud, Mostafa, Kevin Siu, and Andreas Moshovos. "Diffy: A Déjà vu-free differential deep neural network accelerator." *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2018.

# Acceleration of Diffusion Models Inference

## Cambricon-D

full-network differential acceleration with concise memory access

**Accessing and Maintaining the Sign Bit**:

- Maintain a tensor of sign bits for each raw input tensor in a separate memory location in DRAM
- Update the sign bits tensor utilizing a **NDP (near-data-processing) engine**
  - read the raw input tensor and update the sign bits tensor in the off-chip memory devices

1. Fetching the entire raw input tensor on-chip
2. Adding deltas for the incremental update
3. Writing updated data tensor off-chip

**only transmit the deltas off-chip**

$$
\begin{aligned}
\Delta Y_t' &= \mathrm{ReLU}(Y_t) - \mathrm{ReLU}(Y_{t-1}) \\
&= Y_t \cdot \mathrm{sgn}(Y_t) - Y_{t-1} \cdot \mathrm{sgn}(Y_{t-1}) \\
&\approx \Delta Y_t \cdot \mathrm{sgn}(Y_{t-1})
\end{aligned}
$$

$$
\text{where} \quad Y_t := Y_{t-1} + \Delta Y_t.
$$

# Acceleration of Diffusion Models Inference

## Cambricon-D

full-network differential acceleration with concise memory access

**Differential computation of other operators:**

- Group normalization (non-linear) $\mathrm{GN}(G) = \frac{G - \mu_G}{\sqrt{\sigma_G^2 + \epsilon}}$
- Attention mechanism (non-linear) $\mathrm{Attn}(Q, K, V) = \mathrm{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$
- The rest: residual connections, dropouts, upsampling, and downsampling. (linear)

# Acceleration of Diffusion Models Inference

## Cambricon-D

full-network differential acceleration with concise memory access

**2. The outlier-aware PE array**

- Outliers: delta values that overflow in the *fp-to-int* quantization
- PE array: processing elements that performs differential computing
- Problems
  - Increasing the bitwidth to incorporate all outliers is unrealistic
  - Throwing them away will also incur a significant precision penalty

# Acceleration of Diffusion Models Inference

## Cambricon-D

full-network differential acceleration with concise memory access

**2. The outlier-aware PE array**

- Solutions

  Each PE is a multiplier group. In each multiplier group,

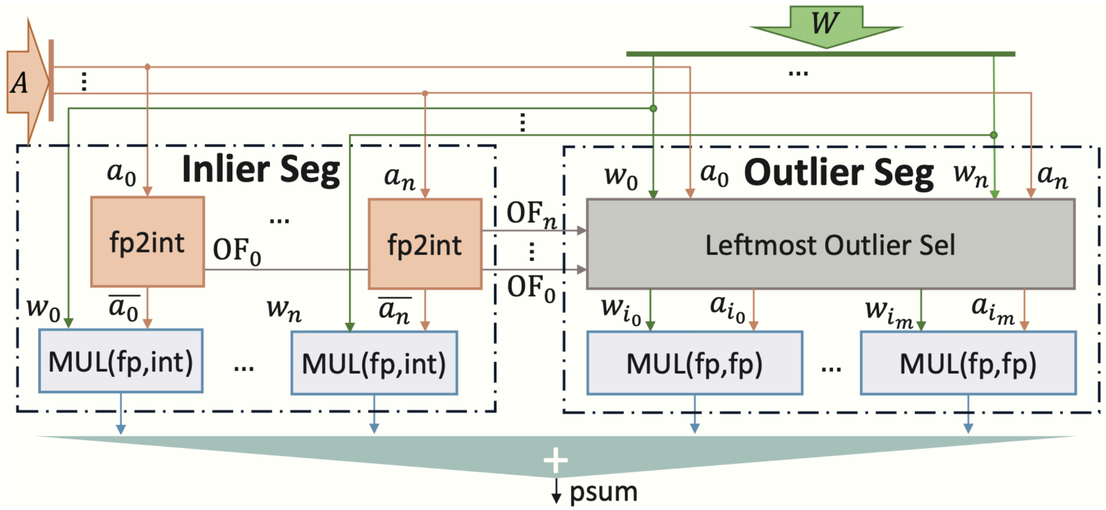
  1. n *int-and-fp* multipliers to multiply the **quantized inliers** with the **floating point weights**
  2. m *fp-and-fp* multipliers to multiply the **floating point outliers** with the **floating point weights**

# Acceleration of Diffusion Models Inference

## Cambricon-D

full-network differential acceleration with concise memory access

**2. The outlier-aware PE array**




Architecture details of a multiplier group in a PE.

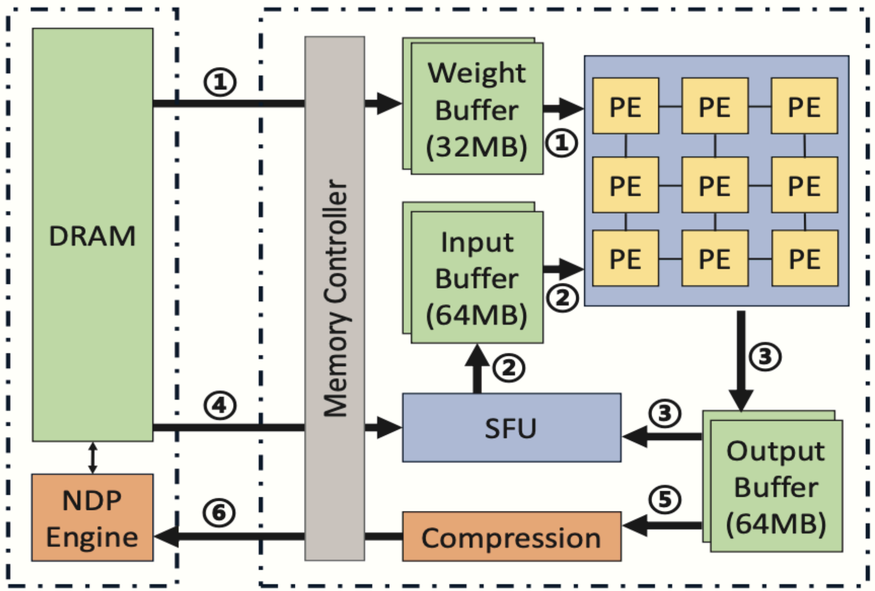
1) Each input delta (fp) is first attempted to be quantified into int.
2) Those who failed to quantize give a overflow flag (OF).
3) A leftmost outlier selection circuit gather the first m outliers according to OFs.

# Acceleration of Diffusion Models Inference

## Cambricon-D

full-network differential acceleration with concise memory access

**The overall architecture of Cambricon-D**




**The PE array**: differential computing.

**The SFU**: other functions besides the tensor multiplication (e.g. ReLU).

**The Compression Unit**: compresses the delta values before writing off-chip.

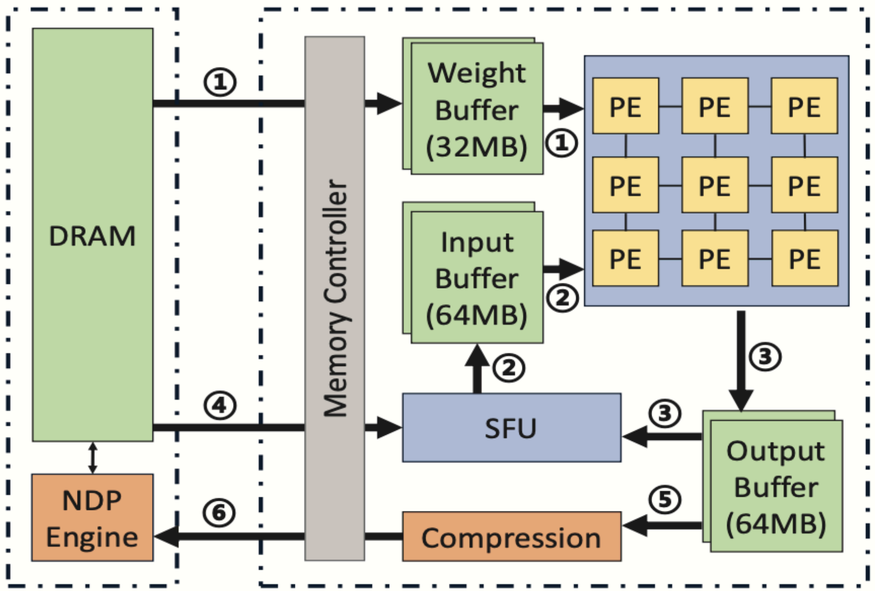
**The NDP engine** (on the DRAM side)

# Acceleration of Diffusion Models Inference

## Cambricon-D

full-network differential acceleration with concise memory access

**The overall architecture of Cambricon-D**




**The dataflow of Cambricon-D of 6 kinds:**

1) Weights: DRAM -> on-chip buffer -> PE array
2) Input deltas: on-chip buffer -> PE array
3) Output deltas: PE array -> on-chip buffer -> SFU

**The PE array**: differential computing.

**The SFU**: other functions besides the tensor multiplication (e.g. ReLU).

**The Compression Unit**: compresses the delta values before writing off-chip.

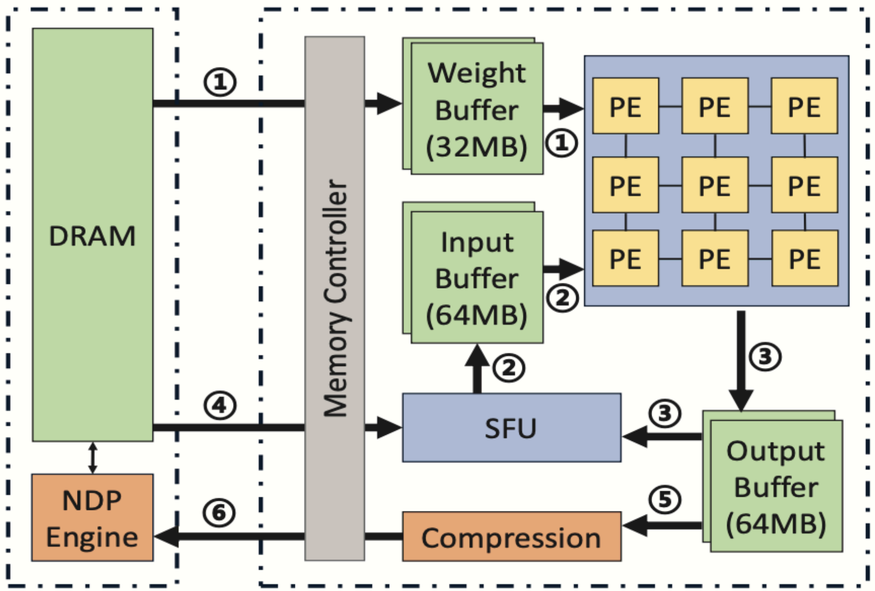
**The NDP engine** (on the DRAM side)

# Acceleration of Diffusion Models Inference

## Cambricon-D

full-network differential acceleration with concise memory access

**The overall architecture of Cambricon-D**




**The dataflow of Cambricon-D of 6 kinds:**

1) Weights: DRAM -> on-chip buffer -> PE array
2) Input deltas: on-chip buffer -> PE array
3) Output deltas: PE array -> on-chip buffer -> SFU
4) Sign bits: DRAM -> SFU
5) Output deltas: PE array -> on-chip buffer -> Compression
6) Compressed output deltas: Compression -> NDP Engine

**The PE array**: differential computing.

**The SFU**: other functions besides the tensor multiplication (e.g. ReLU).

**The Compression Unit**: compresses the delta values before writing off-chip.

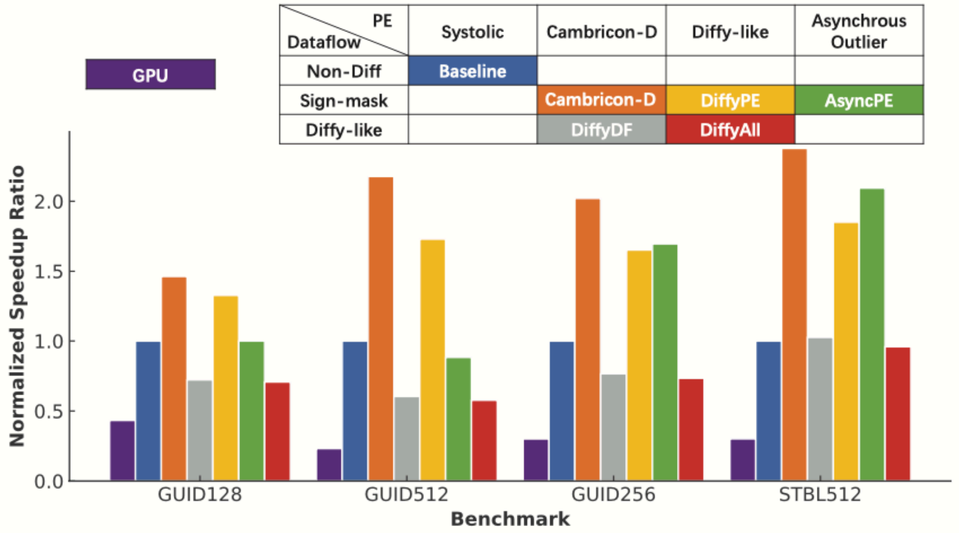
**The NDP engine** (on the DRAM side)

# Acceleration of Diffusion Models Inference

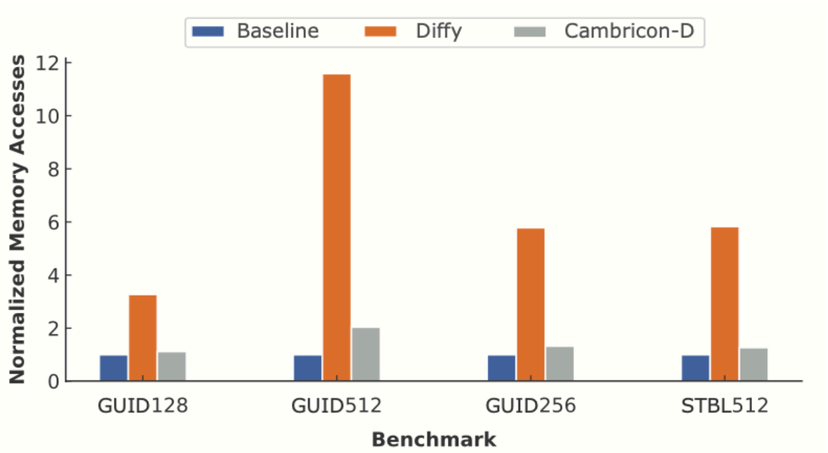
## Cambricon-D

full-network differential acceleration with concise memory access

- **Experiments**




Speedup of Cambricon-D compared with baseline as well as alternative designs.



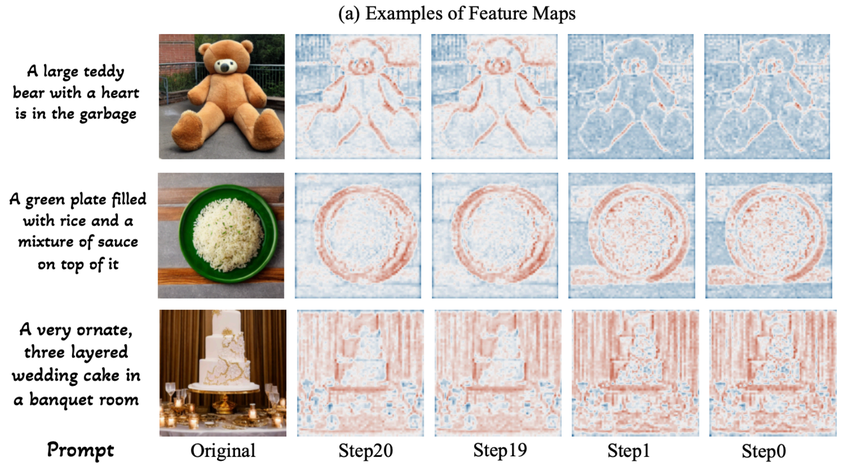

Total memory traffic of Cambricon-D compared with baseline as well as DiffyDF design.

# Acceleration of Diffusion Models Inference

## DeepCache

reduce the computational overhead at each denoising step without additional training

- **Feature redundancy in sequential denoising**
  Adjacent steps in the denoising process exhibit significant temporal similarity in high-level features.




Ma, Xinyin, Gongfan Fang, and Xinchao Wang. "Deepcache: Accelerating diffusion models for free." *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2024.

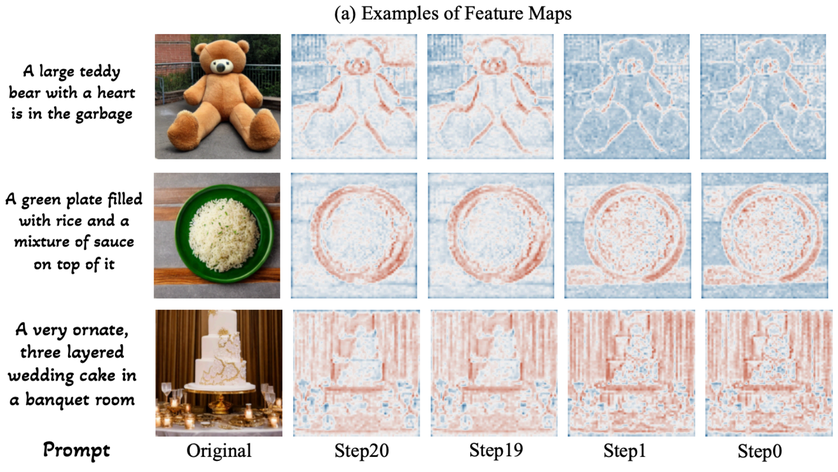
# Acceleration of Diffusion Models Inference

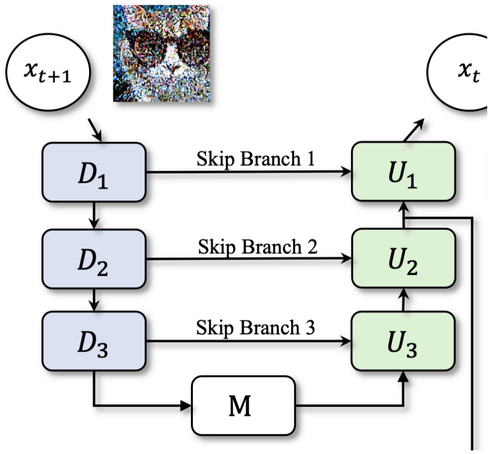
## DeepCache

reduce the computational overhead at each denoising step without additional training

- **Feature redundancy in sequential denoising**
  Adjacent steps in the denoising process exhibit significant temporal similarity in high-level features.

- **Leveraging the structural property of U-Net**
  i.e. a concatenation of :
  1) low-level features from the skip branch
  2) high-level features from the main branch



# Acceleration of Diffusion Models Inference

## DeepCache

reduce the computational overhead at each denoising step without additional training

At each denoising step:

- strategically **cache** slowly evolving high-level features to skip connections with U-Net
- by substituting the compute of the main branch with a retrieving operation from the cache
- while maintaining the low-level features updated

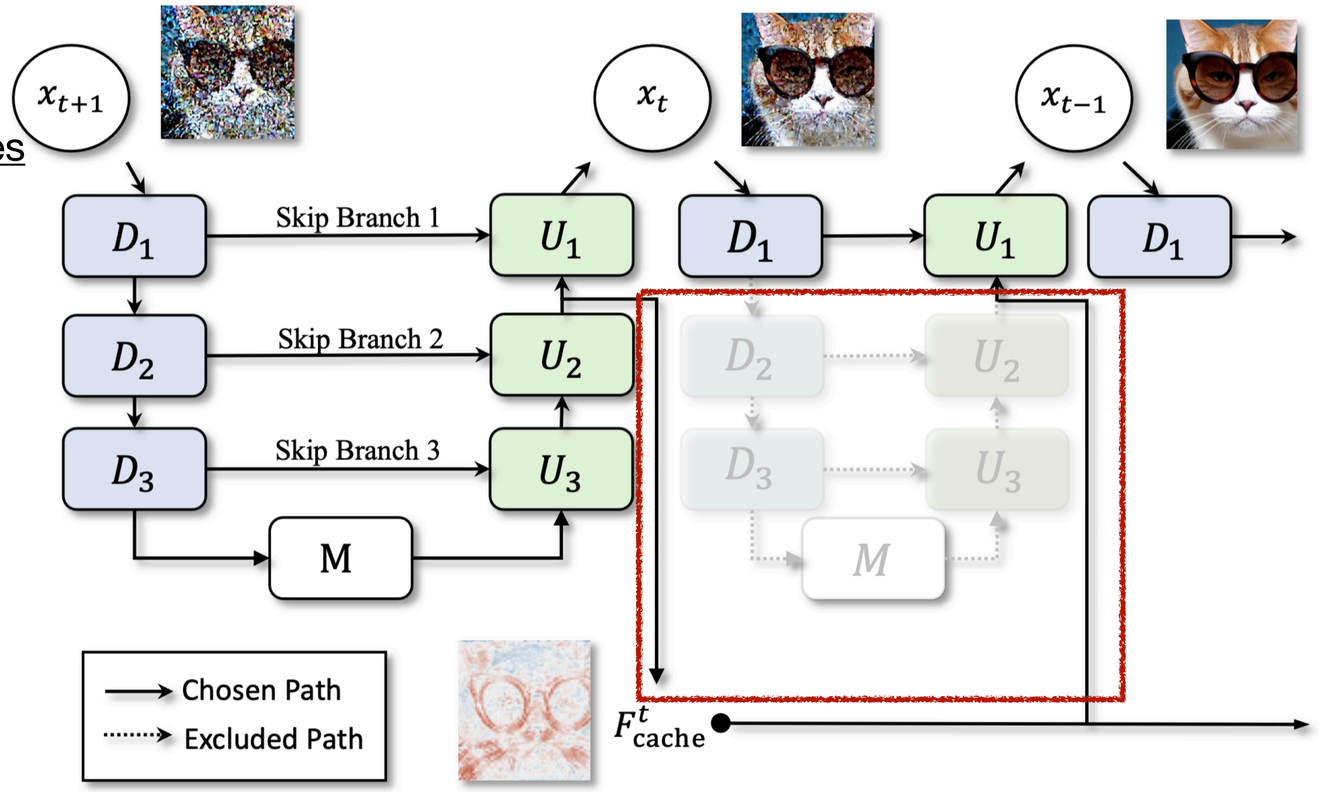



Illustration of DeepCache.

# Acceleration of Diffusion Models Inference

## DeepCache

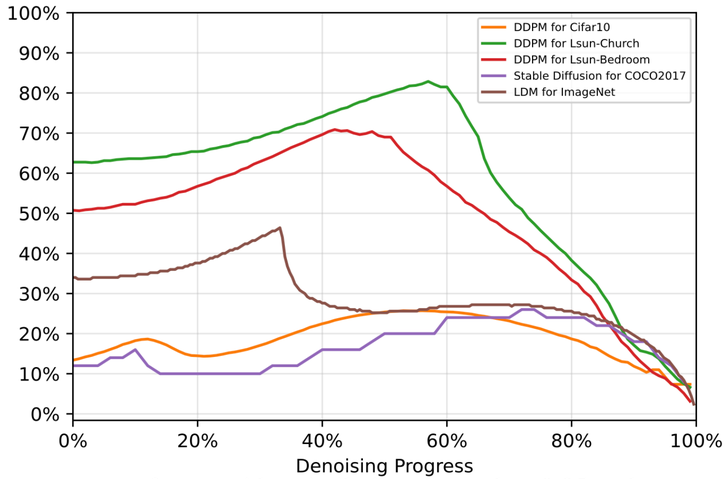
reduce the computational overhead at each denoising step without additional training

Extending to **1:N** Inference:

- The mechanism can be extended to cover more steps
- with the cached features calculated once and reused in the consecutive N-1 steps.

**Non-uniform 1:N** Inference:

- As the similarity of the features does not remain constant across all steps
- sample more on those steps with relatively small similarities.



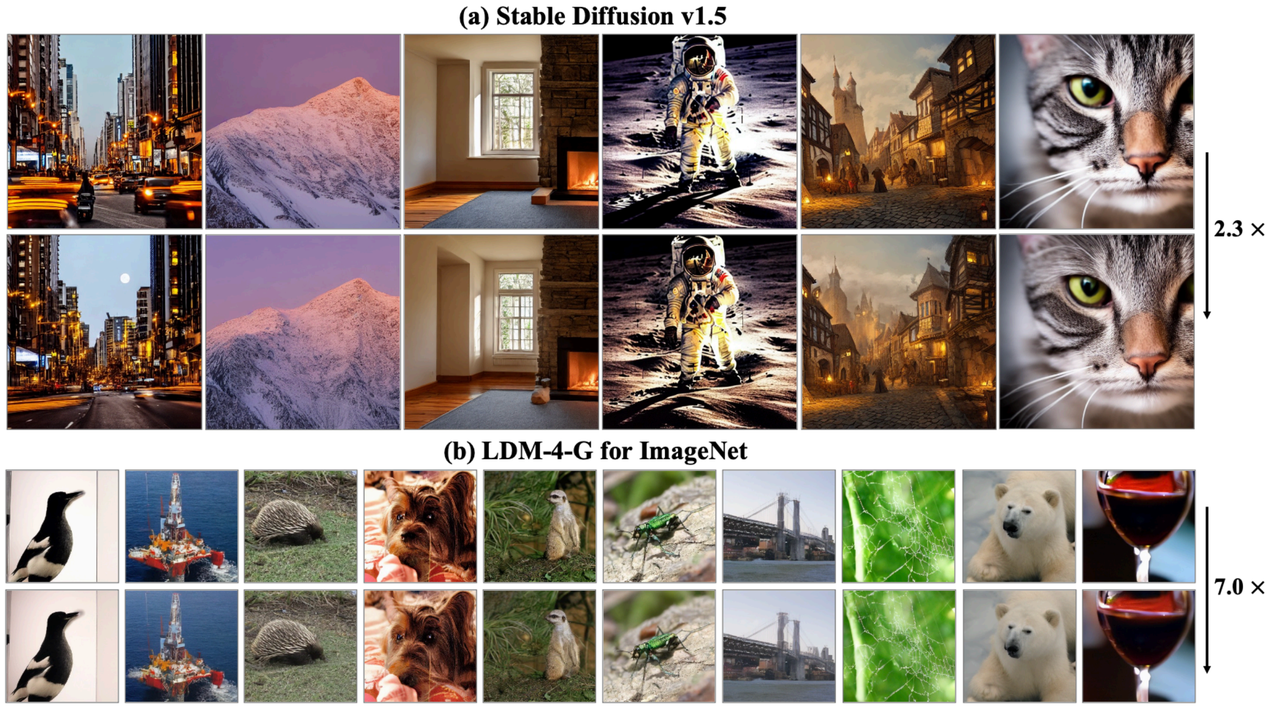

The percentage of steps with a similarity greater than 0.95 to the current step

# Acceleration of Diffusion Models Inference

## DeepCache

reduce the computational overhead at each denoising step without additional training

- Experiments




**Accelerating** Stable Diffusion V1.5 and LDM-4-G by **2.3** and **7.0**, with 50 PLMS steps and 250 DDIM steps respectively.

# Acceleration of Diffusion Models Inference

## DeepCache

reduce the computational overhead at each denoising step without additional training

- Experiments: the effectiveness of the non-uniform 1:N strategy

| Method | MACs ↓ | Throughput ↑ | Speed ↑ | Retrain | FID ↓ | sFID ↓ | IS ↑ | Precision ↑ | Recall ↑ |
|---|---|---|---|---|---|---|---|---|---|
| | ImageNet 256 × 256 | | | | | | | | |
| IDDPM [44] | 1416.3G | - | - | ✗ | 12.26 | 5.42 | - | 70.0 | 62.0 |
| ADM-G [9] | 1186.4G | - | - | ✗ | 4.59 | 5.25 | 186.70 | 82.0 | 52.0 |
| LDM-4 [49] | 99.82G | 0.178 | 1× | ✗ | 3.60 | - | 247.67 | 87.0 | 48.0 |
| LDM-4* | 99.82G | 0.178 | 1× | ✗ | 3.37 | 5.14 | 204.56 | 82.71 | 53.86 |
| Spectral DPM [68] | 9.9G | - | - | ✓ | 10.60 | - | - | - | - |
| Diff-Pruning [10]* | 52.71G | 0.269 | 1.51× | ✓ | 9.27(9.16) | 10.59 | 214.42(201.81) | 87.87 | 30.87 |
| **Uniform - N=2** | 52.12G | 0.334 | 1.88× | ✗ | 3.39 | 5.11 | 204.09 | 82.75 | 54.07 |
| **Uniform - N=3** | 36.48G | 0.471 | 2.65× | ✗ | 3.44 | 5.11 | 202.79 | 82.65 | 53.81 |
| **Uniform - N=5** | 23.50G | 0.733 | 4.12× | ✗ | 3.59 | 5.16 | 200.45 | 82.36 | 53.31 |
| **Uniform - N=10** | 13.97G | 1.239 | 6.96× | ✗ | 4.41 | 5.57 | 191.11 | 81.26 | 51.53 |
| **Uniform - N=20** | 9.39G | 1.876 | 10.54× | ✗ | 8.23 | 8.08 | 161.83 | 75.31 | 50.57 |
| **NonUniform - N=10** | 13.97G | 1.239 | 6.96× | ✗ | 4.27 | 5.42 | 193.11 | 81.75 | 51.84 |
| **NonUniform - N=20** | 9.39G | 1.876 | 10.54× | ✗ | 7.11 | 7.34 | 167.85 | 77.44 | 50.08 |

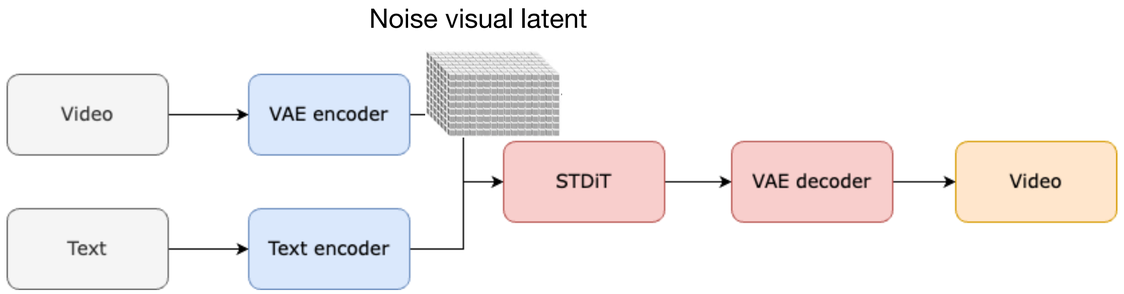
Class-conditional generation quality on ImageNet using LDM-4-G with 250 DDIM steps.

# Open-sora

## Leveraging pretrained T2I models for video generation

In the inference stage

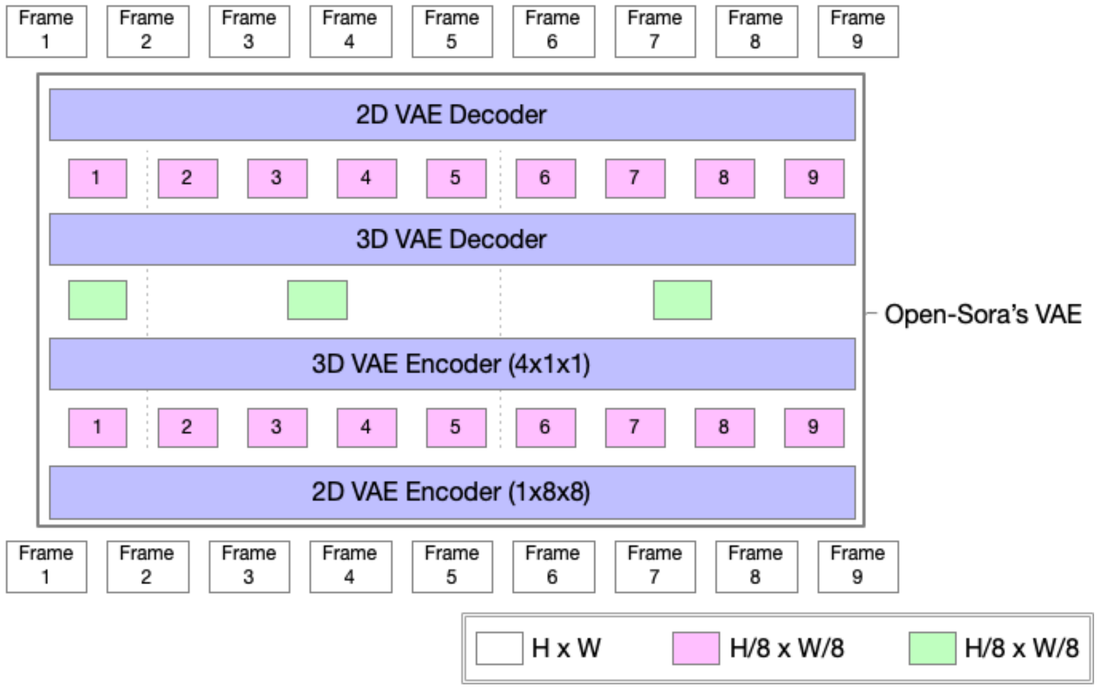
- Randomly sample a Gaussian noise from the latent space of the VAE
- Input noise into the STDiT (Spatial Temporal Diffusion Transformer) model together with the prompt embedding for denoising
- Input denoised features into the VAE decoder to get the video



# Open-sora

## Leveraging pretrained T2I models for video generation

Video Compression Network



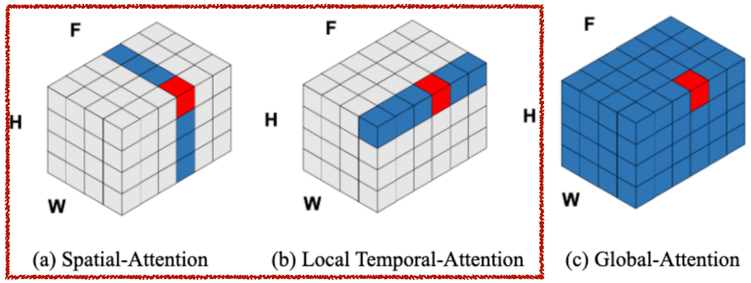

- first compress the video in the spatial dimension by 8x8 times
- then compress the video in the temporal dimension by 4x times

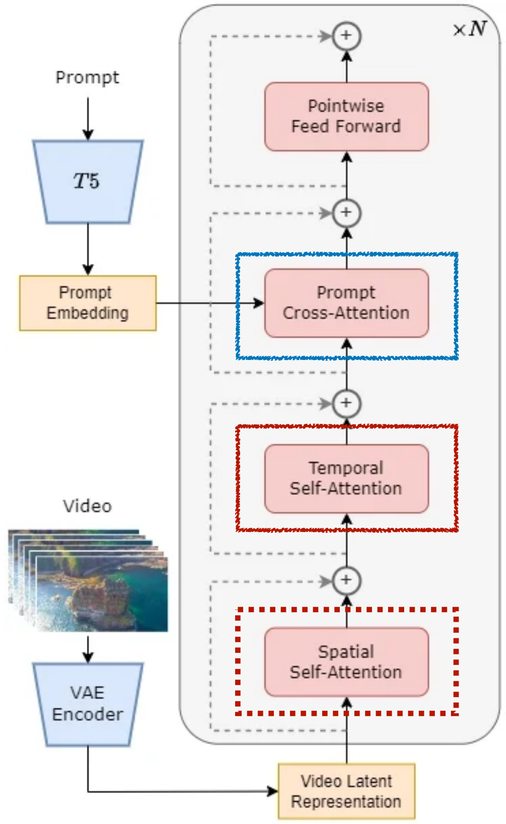
# Open-sora

## Leveraging pretrained T2I models for video generation

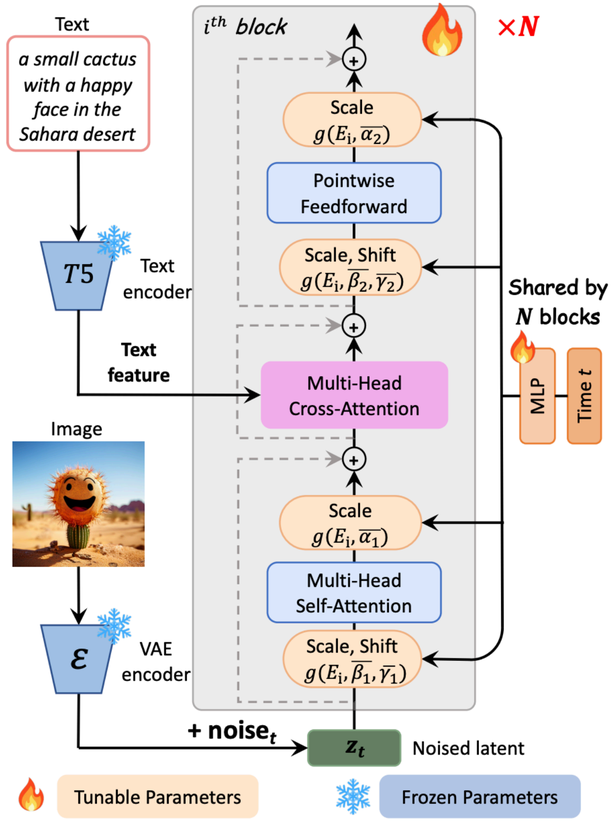
STDiT Model Structure

- The **cross-attention** module is used to align the semantics of the text
- Add a **temporal attention** layer for modeling temporal relationships



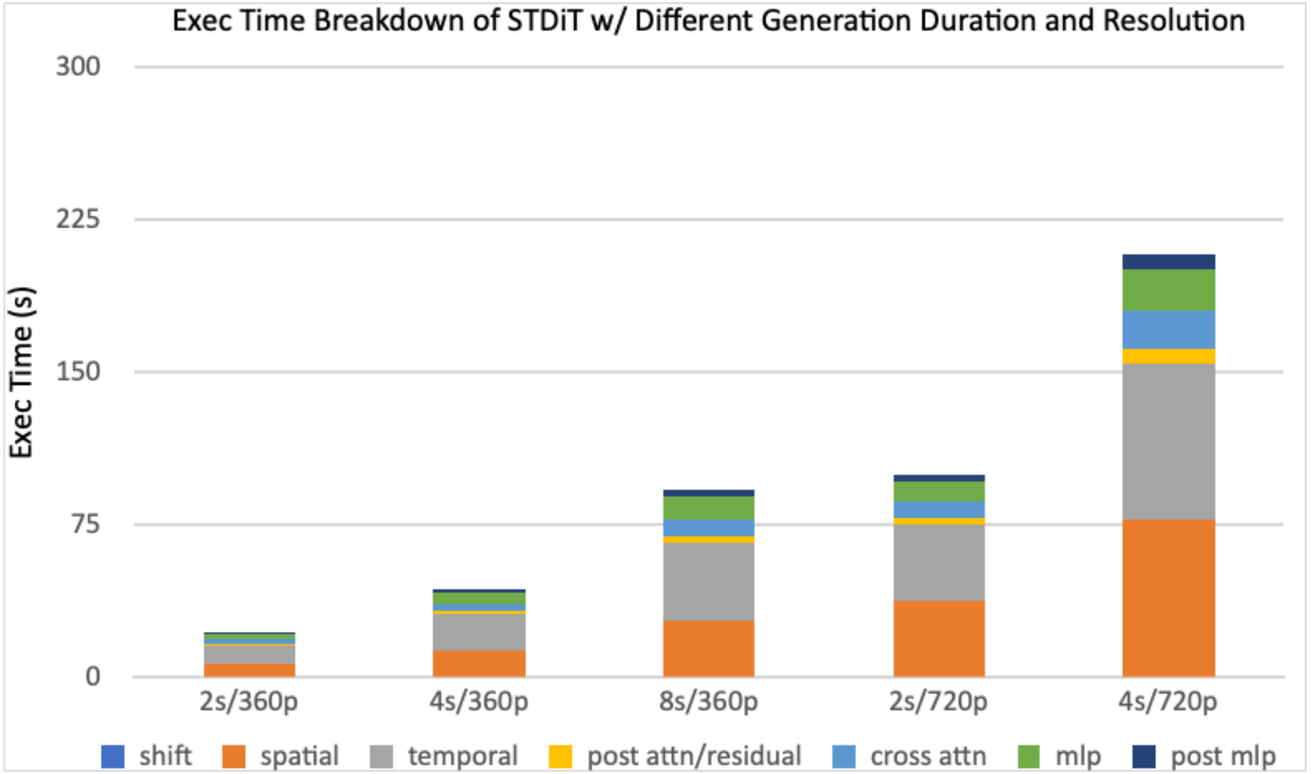

STDiT Model Structure Schematic




Model architecture of PIXART-α

# Our Findings

- Attention computation takes up most of the exec time



# Discussion

**Thanks for listening!**