

**Lecture 8:**

# **Controlling Visual Generative AI**

---

**Visual Computing Systems**  
**Stanford CS348K, Spring 2026**

# Many examples of visual/spatial generation today

- Task → text answer
  - Example: (image, text question) → text answer ... **“How many cats are in this photo?”**
- Task → image
  - Example: (text, image) → image ... **“Take the provided photo of me and make my hairstyle more punk.”**
- Task → video
  - Example: (text) → video ... **“Make me a video of Will Smith eating Spaghetti”**
- Task → 3D object
  - Example: (text, image) → 3D model ... **“Give me a 3D model of a vintage 1960’s pickup truck like the one in this photo, but add cartoony eyes”**
- Task → animation
  - Example: (text) → 3D animation ... **“an animation of a triple jump”**
- Task → action
  - Example: (image, text) → action ... **“given that you see this state of the kitchen, what action should the agent take if the goal is to put the milk on the counter in the fridge”**
- Task → code/program
  - Example: (text → code) ... **“give me a program that when executed creates geometric primitives that resemble a picket fence”**

# **Micro-intro to diffusion-based generation**

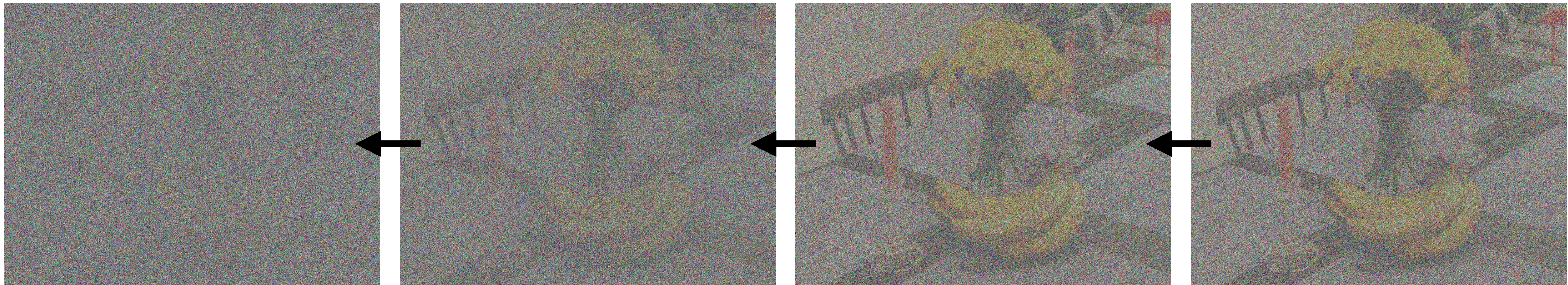
# Suppose you are given a dataset of images $x_i$

- You'll like to draw a sample according to the underlying data distribution  $p(x)$

# Diffusion-based image synthesis

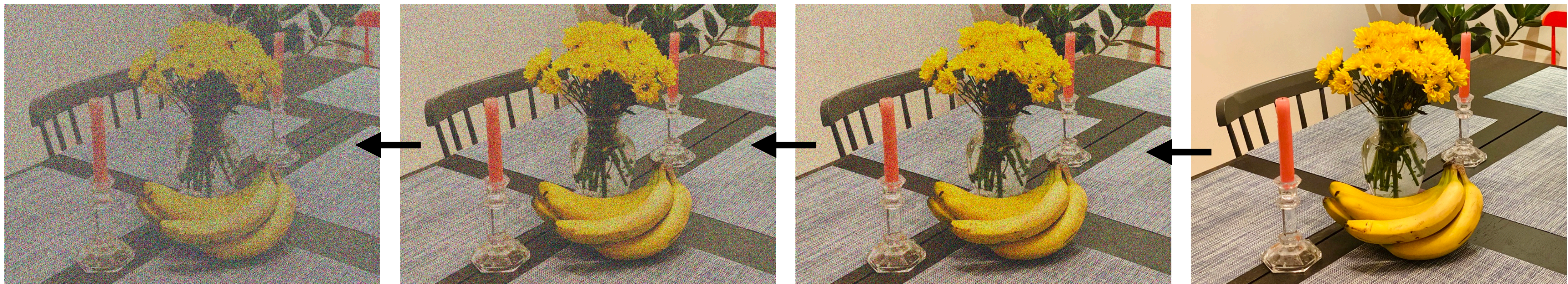
Iterative Markov-chain Monte-carlo (MCMC) process to generate a sample  $\mathbf{x}$  (an image) from distribution  $p(\mathbf{x})$  of observed images

**Forward diffusion: iteratively add noise**  $q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1 - \beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I})$



$\mathbf{x}_T$

$\mathbf{x}_{T-1}$



$\mathbf{x}_1$

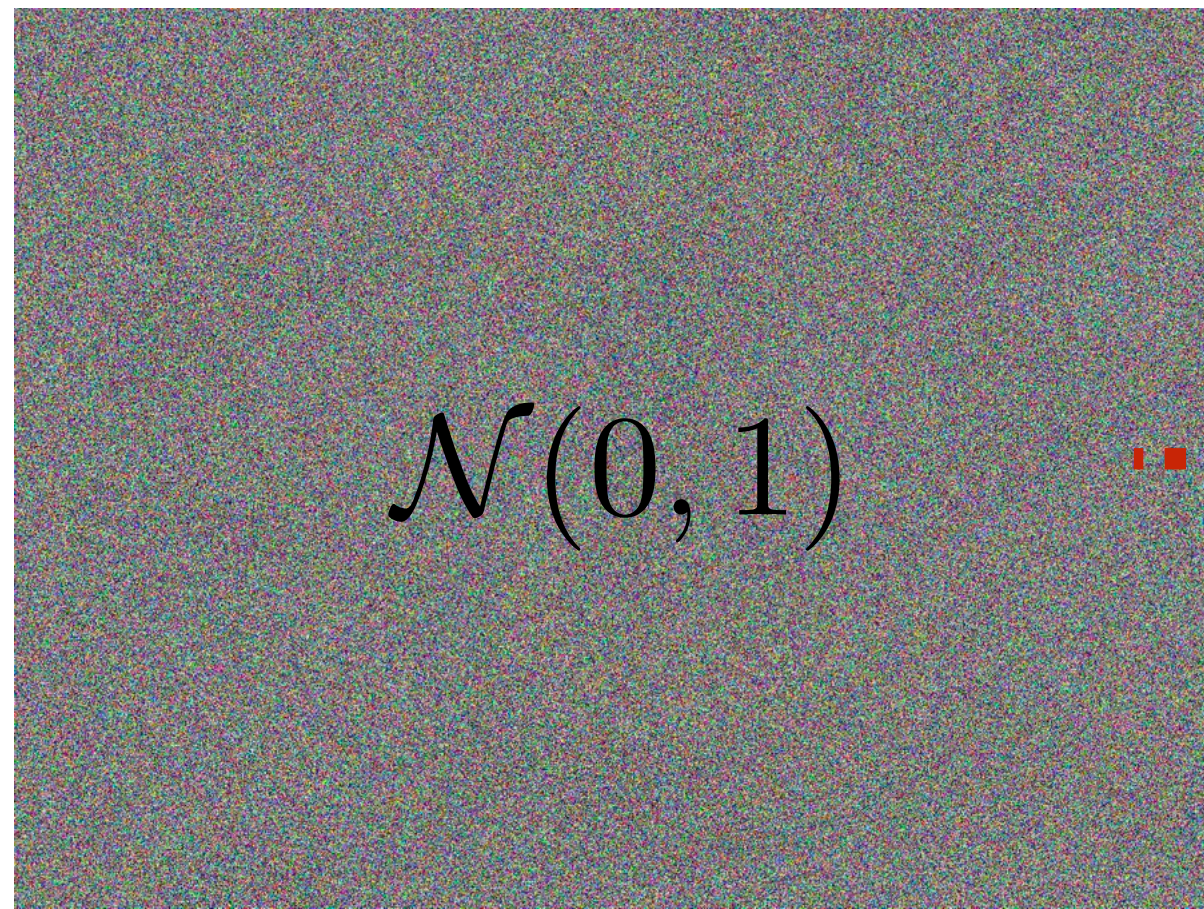
$\mathbf{x}_0$

# Diffusion-based image synthesis

Reverse: iteratively remove noise from random sample to obtain image from  $p(\mathbf{x})$

$$\mathbf{x}_{i+1} \leftarrow \mathbf{x}_i + \epsilon \nabla_{\mathbf{x}} \log p(\mathbf{x}) + \sqrt{2\epsilon} \mathbf{z}_i, \quad i = 0, 1, \dots, T$$

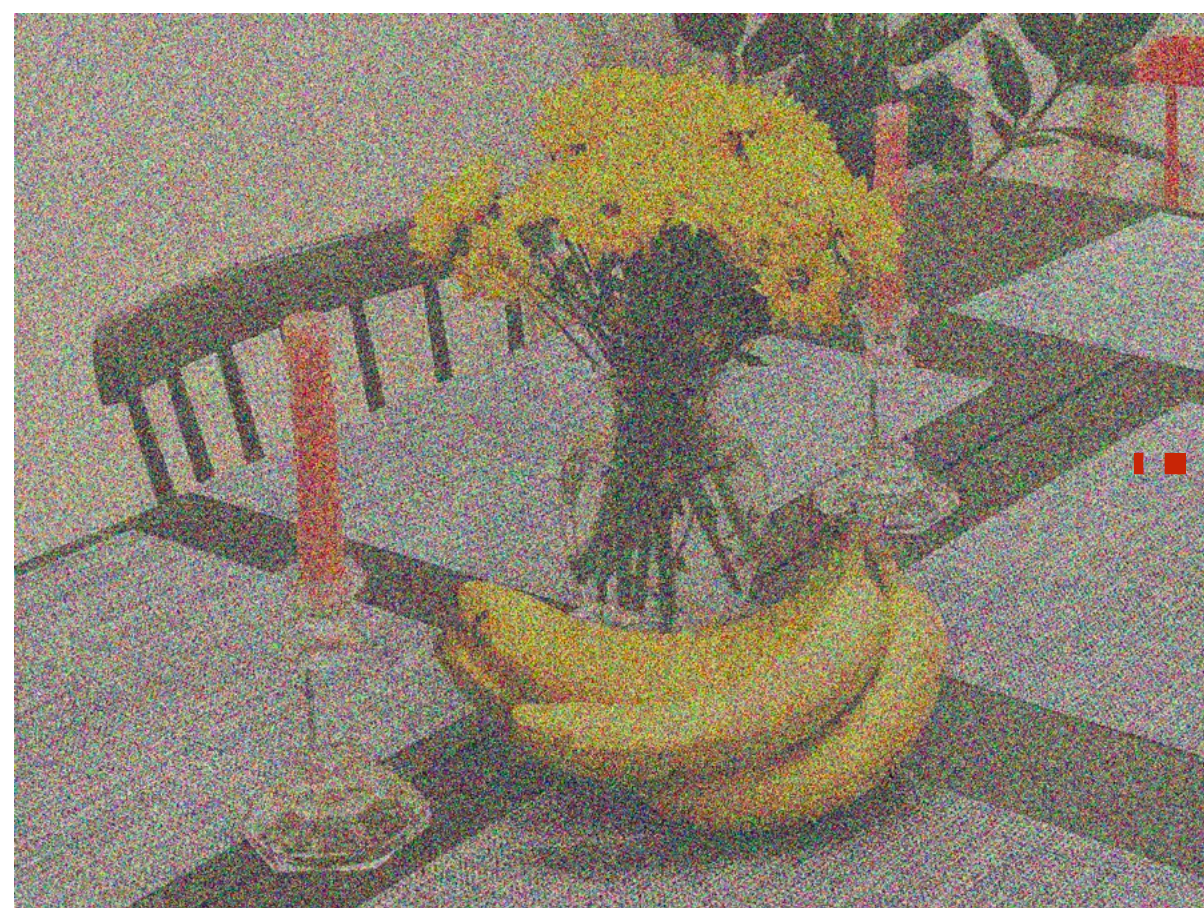
← (“score function”)



$\mathbf{x}_0$



$\mathbf{x}_1$



$\mathbf{x}_{T-1}$



$\mathbf{x}_T$

# Conditioned generation

*"cat"*

Condition

$X_t$



Model

$X_{t+1}$



# Guided diffusion

- Assume we know  $p(\mathbf{y} \mid \mathbf{x})$  for random variables  $\mathbf{x}$  and  $\mathbf{y}$  (e.g., a classifier)
  - Example:  $\mathbf{x}$  is an image,  $\mathbf{y}$  is a string describing the image
  - Given an image ( $\mathbf{x}$ ), infer a caption ( $\mathbf{y}$ )

$$p(\mathbf{x} \mid \mathbf{y}) = p(\mathbf{x})p(\mathbf{y} \mid \mathbf{x}) / \int p(\mathbf{x})p(\mathbf{y} \mid \mathbf{x})d\mathbf{x} \quad \text{(Bayes Rule)}$$

## Bayes for score function

$$\nabla_{\mathbf{x}} \log p(\mathbf{x} \mid \mathbf{y}) = \nabla_{\mathbf{x}} \log p(\mathbf{x}) + \nabla_{\mathbf{x}} \log p(\mathbf{y} \mid \mathbf{x})$$

↑  
(Unguided score function)

Modify image  $\mathbf{x}$  so that image is more likely  
[to come from the training set]

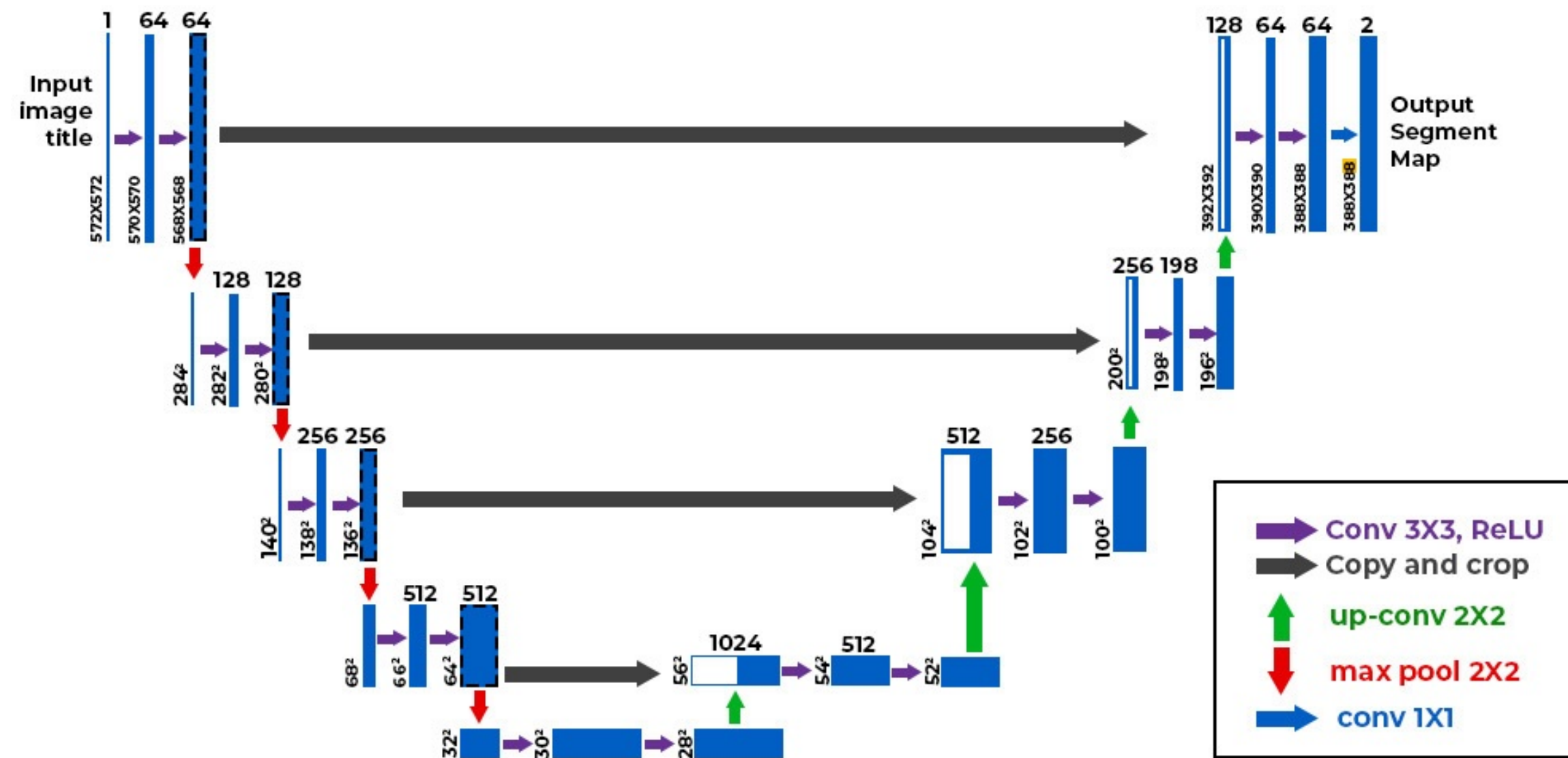
←  
(Prompt guidance)

Modify image  $\mathbf{x}$  to make the prompt a  
more likely description of the image

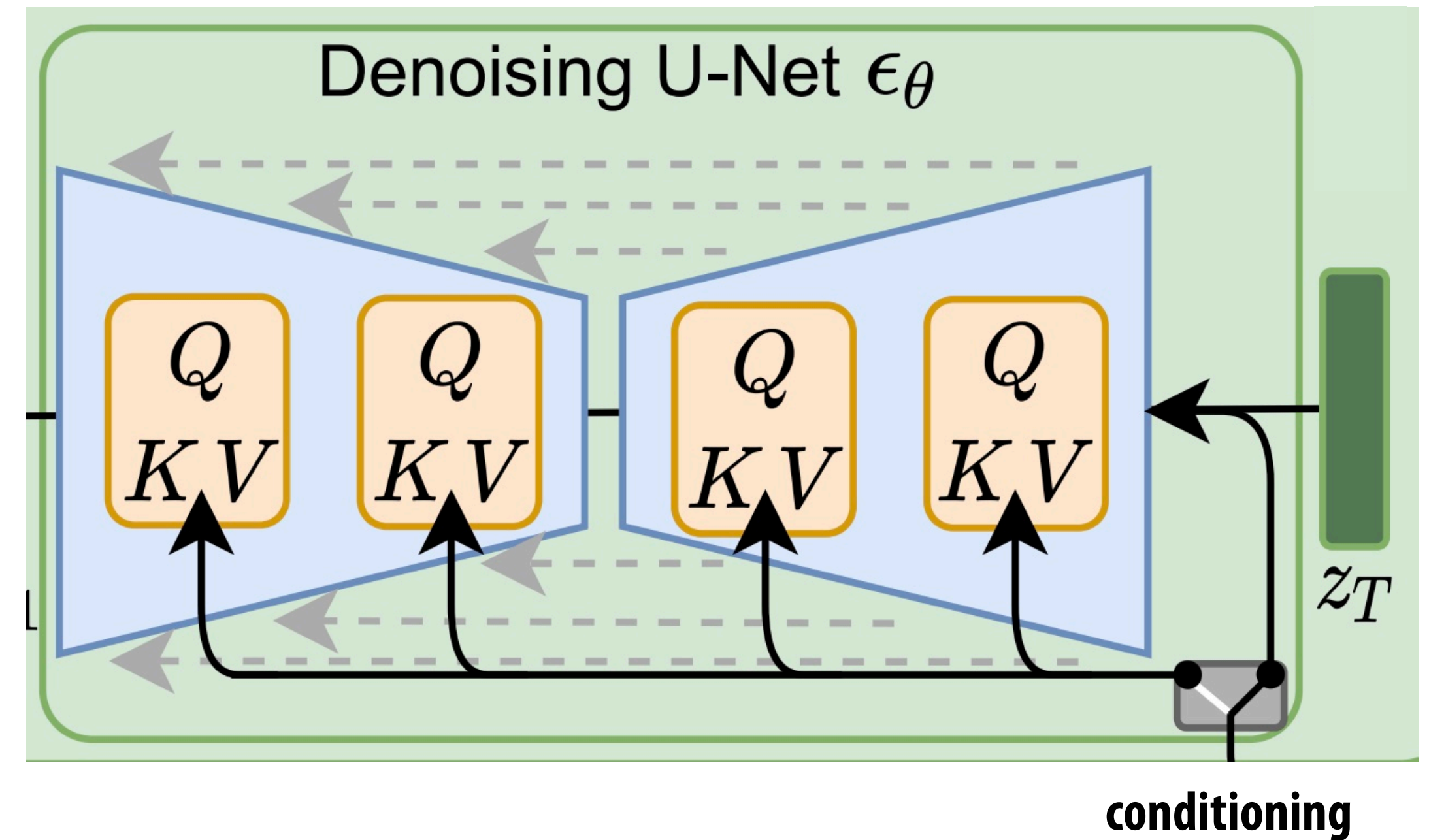
# Performance/efficiency optimizations

# A basic U-NET architecture

Classic U-Net for image segmentation  
(Basic blocks are convolutional layers)

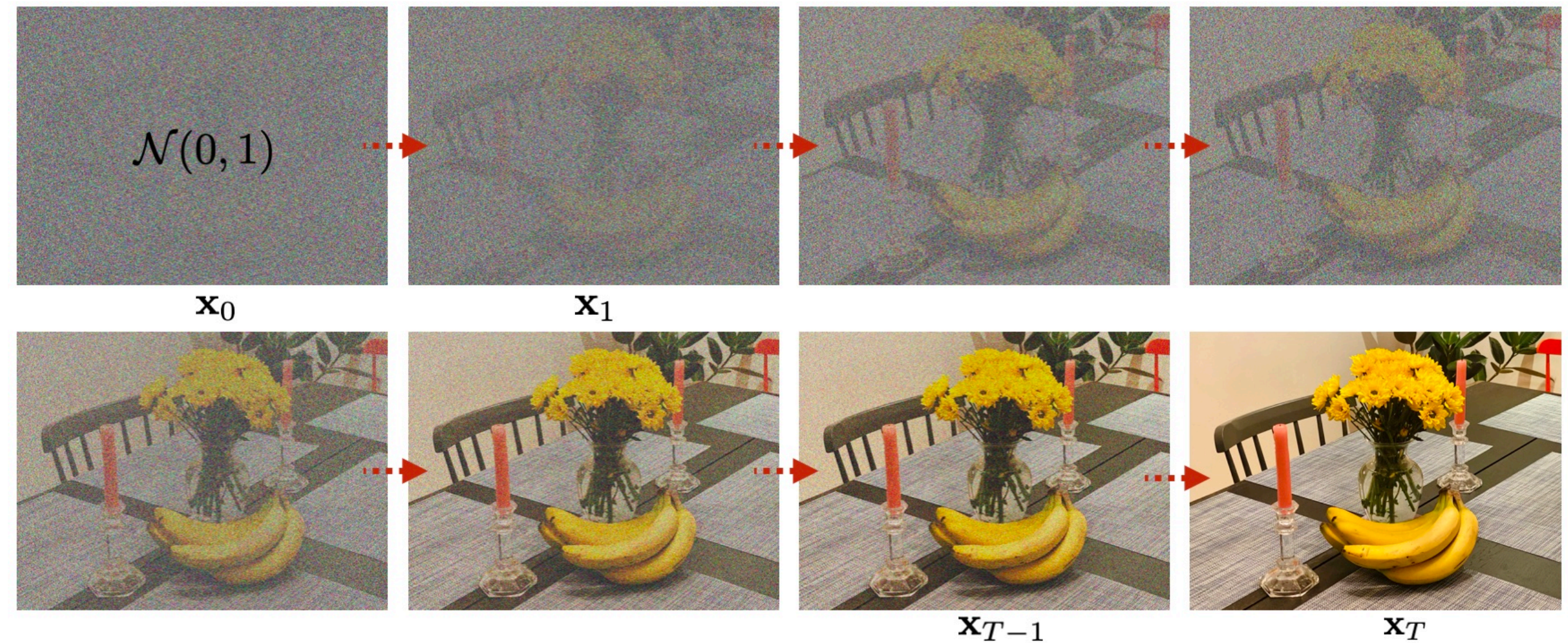


U-Net used in modern diffusion models  
(Basic blocks are transformer modules with cross-attention layers)



# Performance challenges of inference

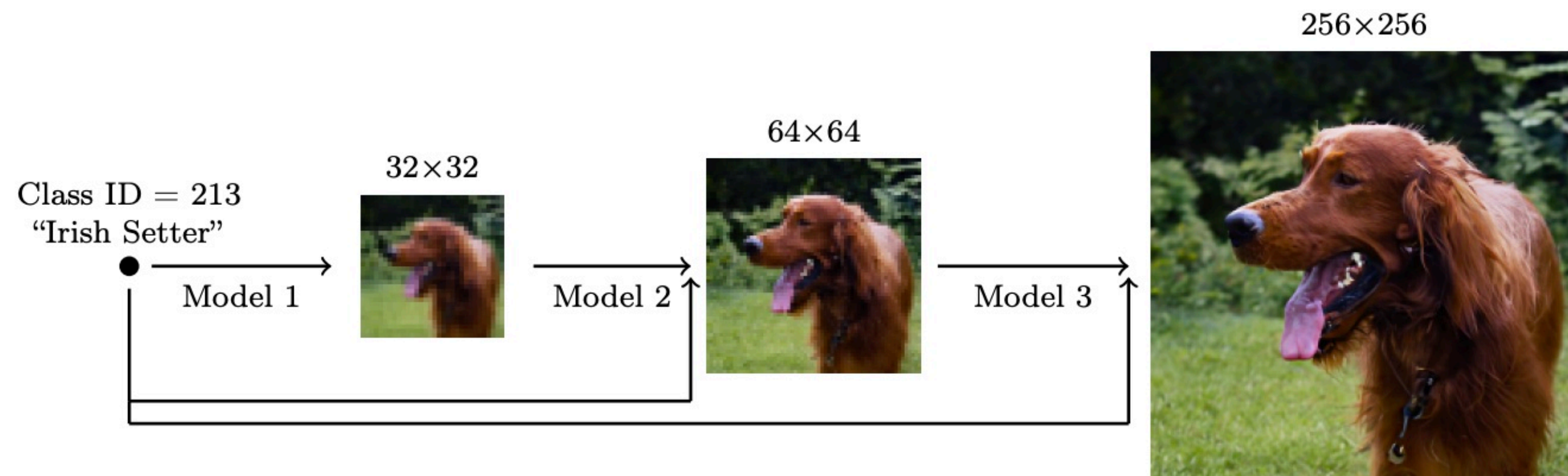
- Diffusion is an iterative process:
  - Requires many steps to convergence



- Ways to improve inference efficiency
  - Diffuse in low-dimensional (latent) space
  - Superresolution techniques
  - Learn to take bigger steps

# Superresolution

- Use diffusion to produce a low-resolution image
- Then subsequent models perform neural superresolution



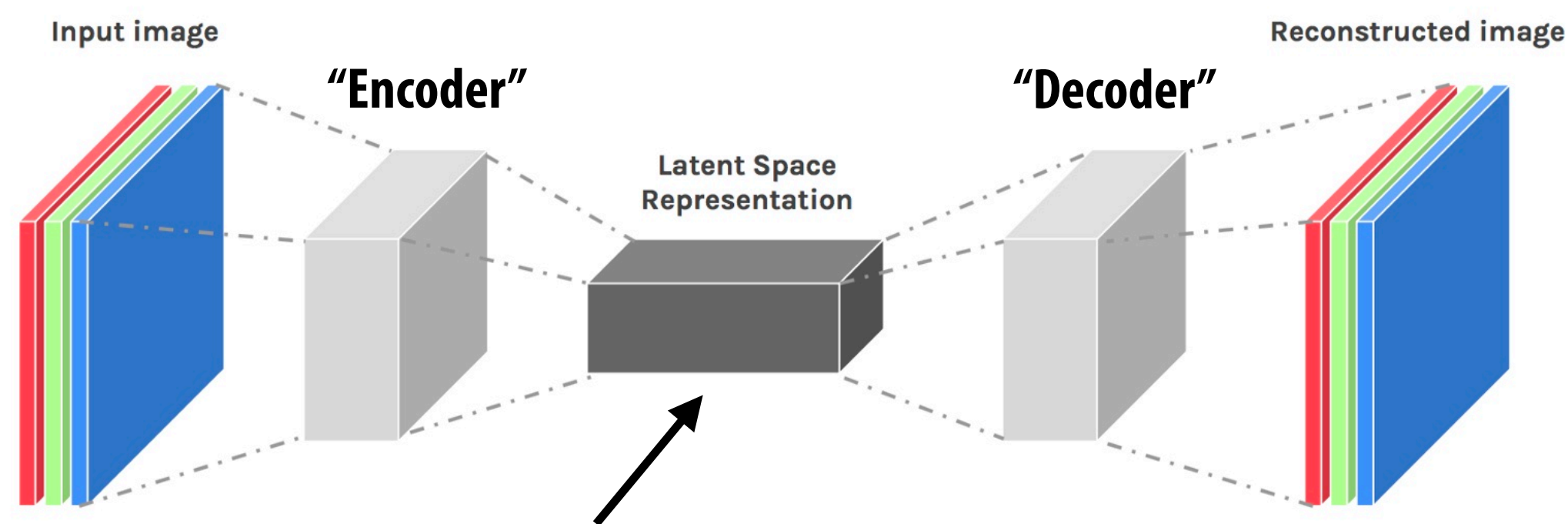
Cascade of diffusion models



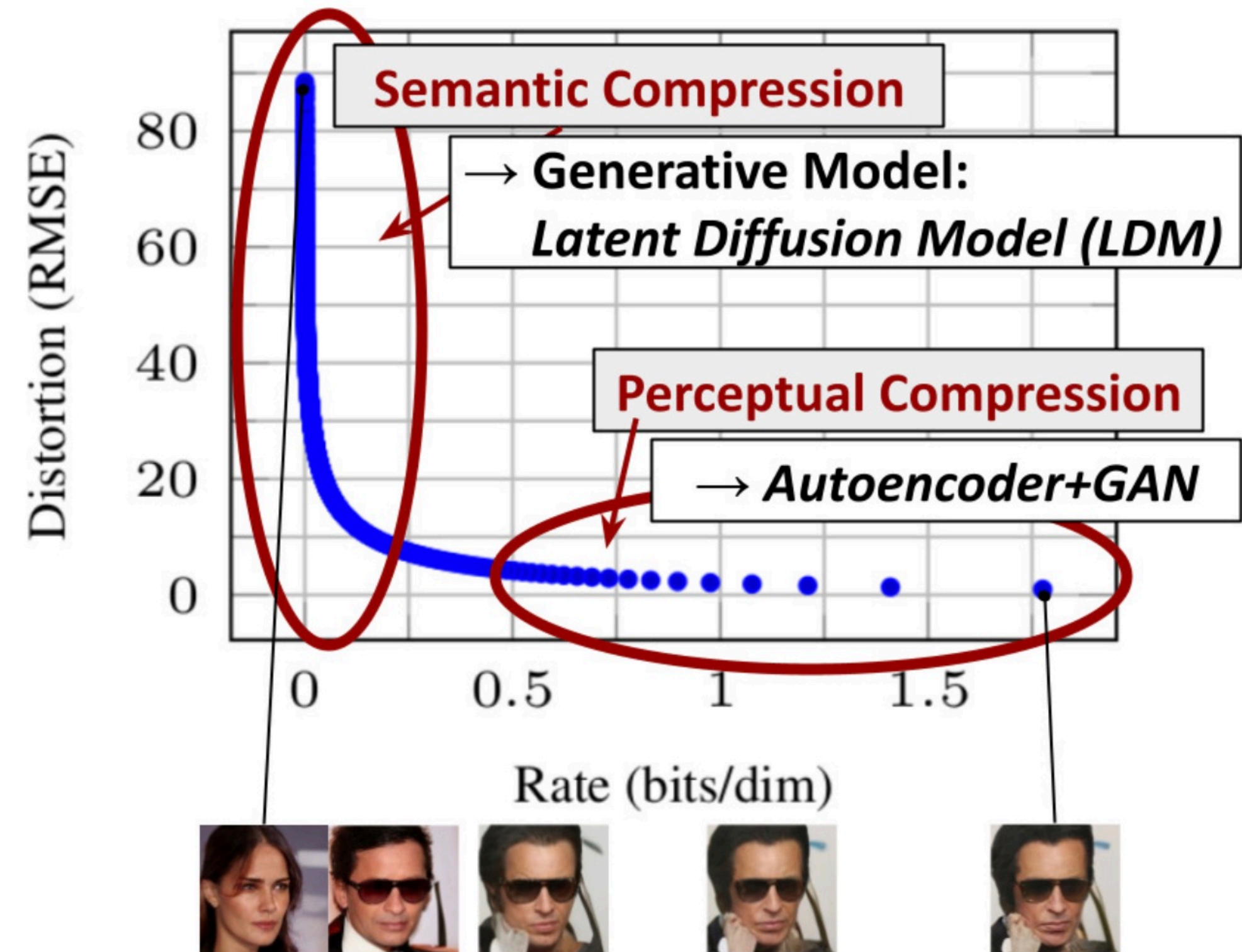
Bicubic upsampling vs. two forms of learned upsampling

# Perform diffusion in lower-dimensional latent space

- Main idea: perform diffusion in the lower dimensional latent space of images, not in high-dimensional RGB pixel space
- After diffusing a latent representation, “decode” latent to final image

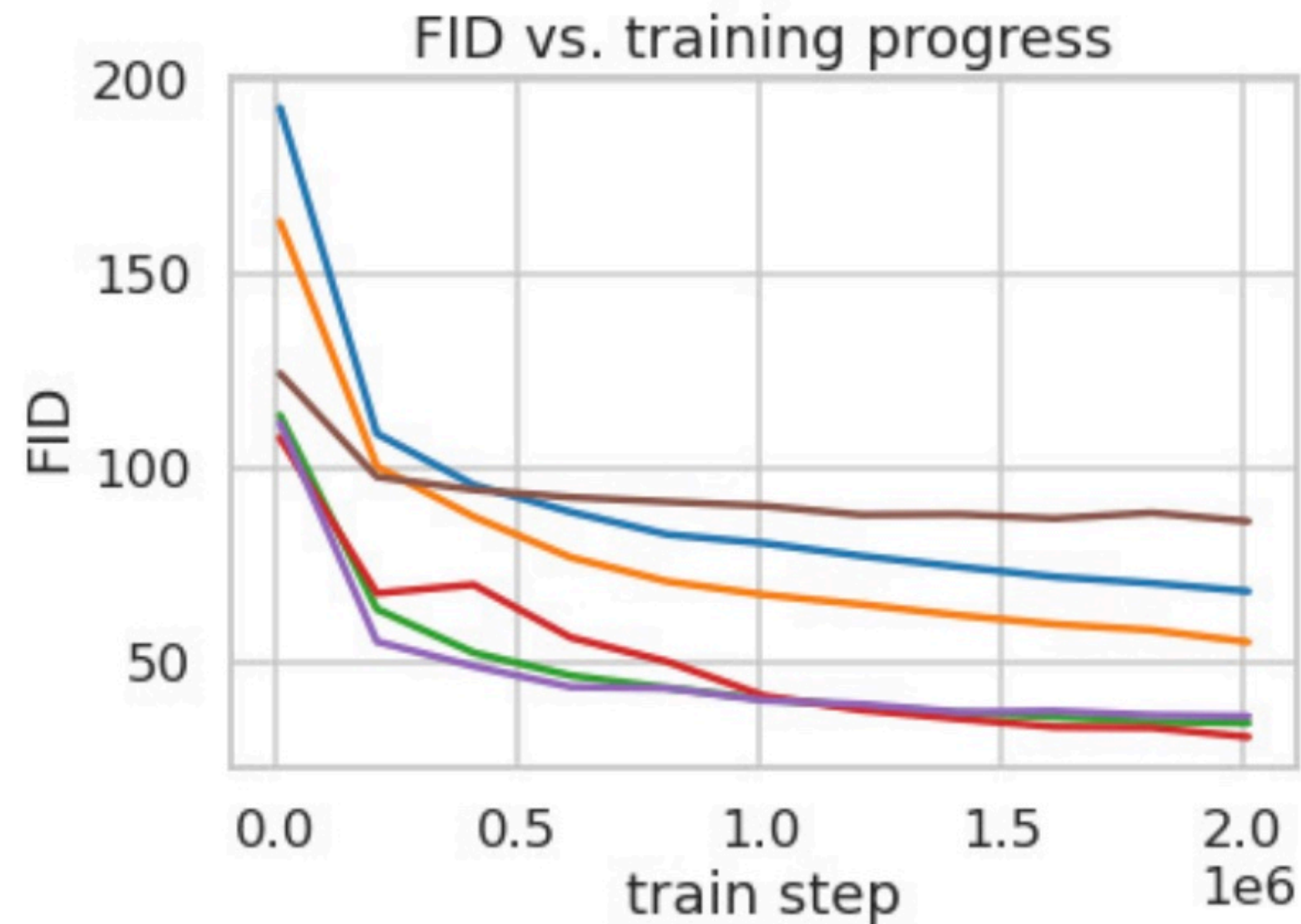


If this latent representation is compact, then it is a compressed representation of the input image



# Perform diffusion in latent space

- Implications to both training efficiency and inference efficiency



Per-pixel representations, can represent data well, but require significant training to learn good models

“Sweet spot”: learns good model + trains quickly

Latent representation too compressed (cannot represent data well)

# Learn to take larger steps

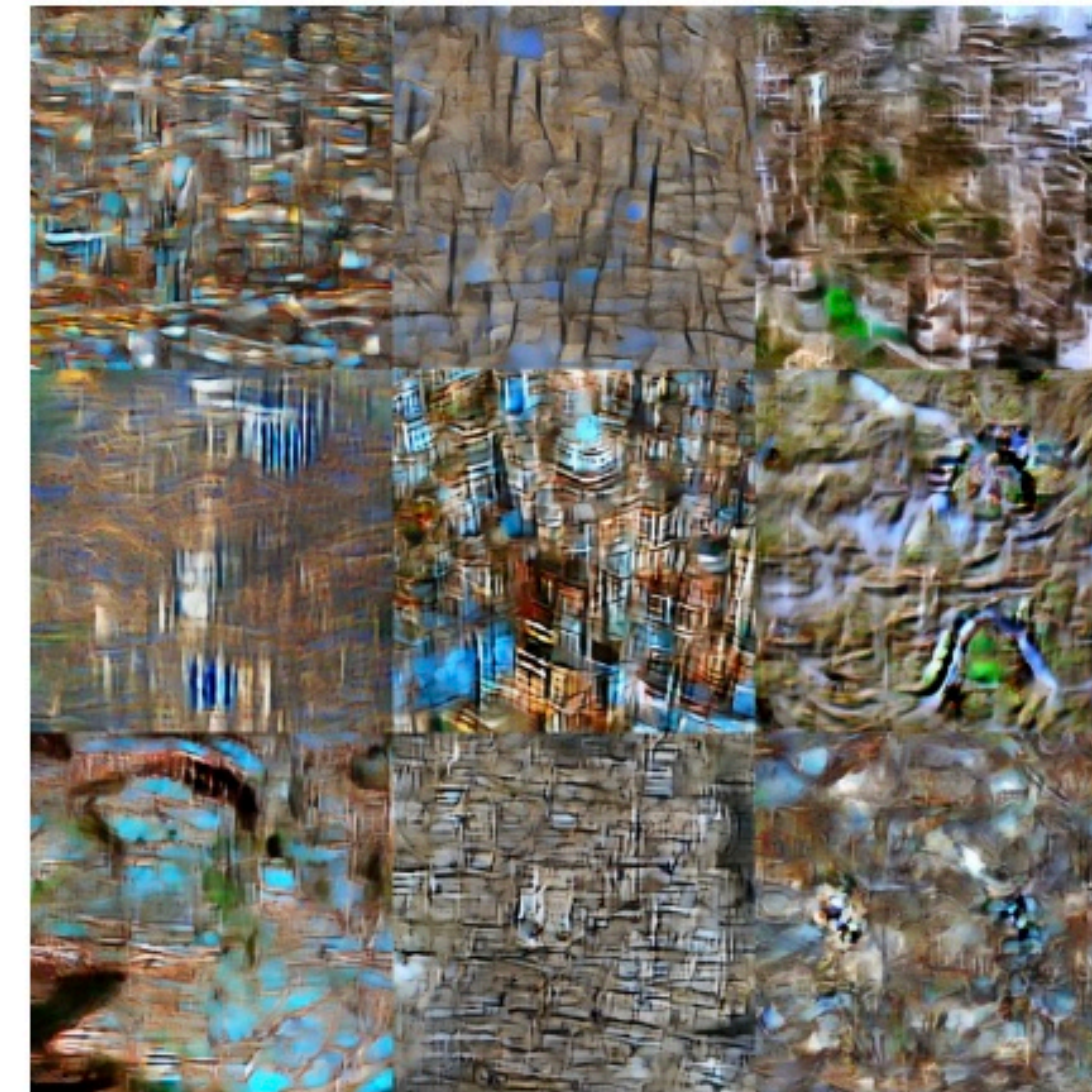
- Given a diffusion model, learn a new (second) diffusion model that reproduces multiple steps of the diffusion process
- This is a form of “model distillation”: training a “student” model to emulate the output of a teacher. Here, the teacher’s output is multiple steps of the diffusion process



Ours 2 steps



Ours 4 steps



DDIM 2x2 steps



DDIM 4x2 steps

Prompt: “A beautiful castle, matte painting.”

# Learn to take larger steps

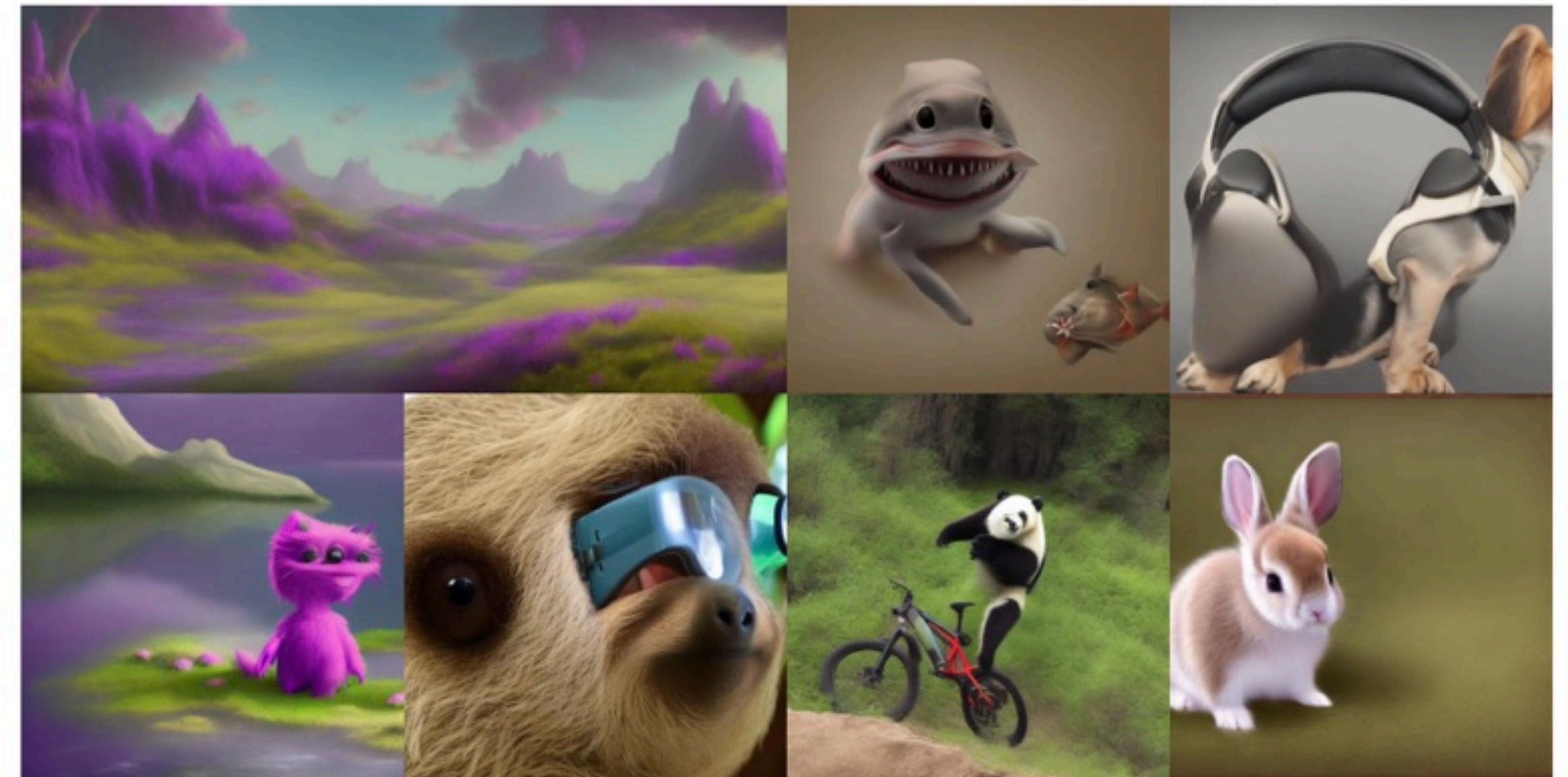
Distilled Text-to-Image samples (4 steps)



Native Text-to-Image samples (4 steps)



Native Text-to-Image samples (8 steps)



# Controlling the output of generative models

# Text conditioning

“A classroom of many attentive college students”

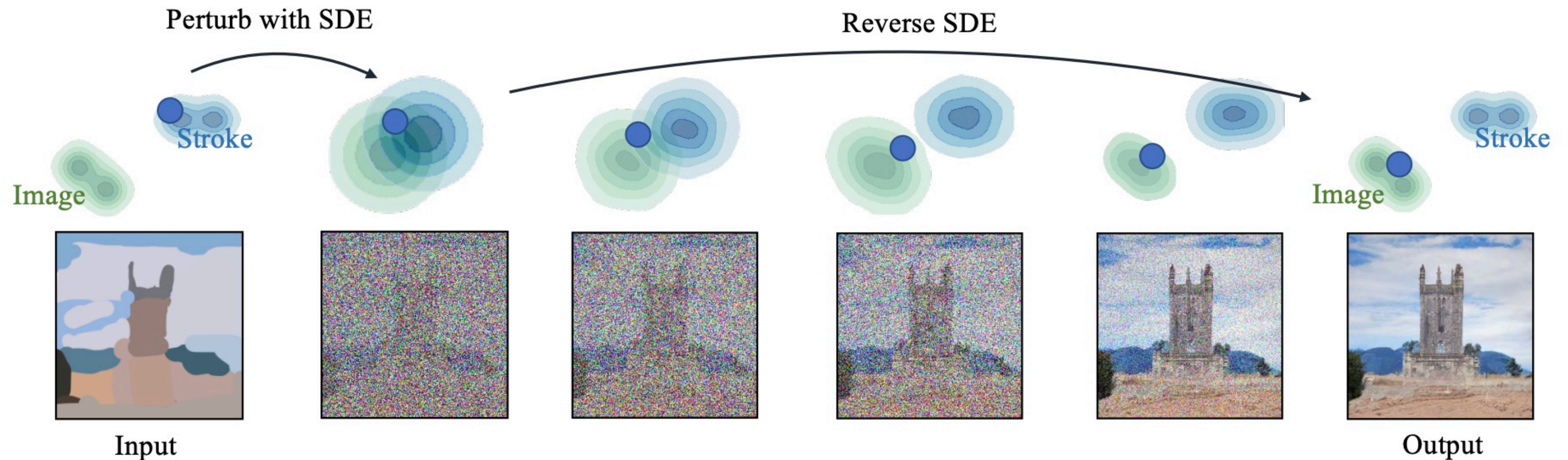


**I want the visual output to look LIKE THIS.  
(Other forms of conditioning)**

# img2img (conditioning based on an image)

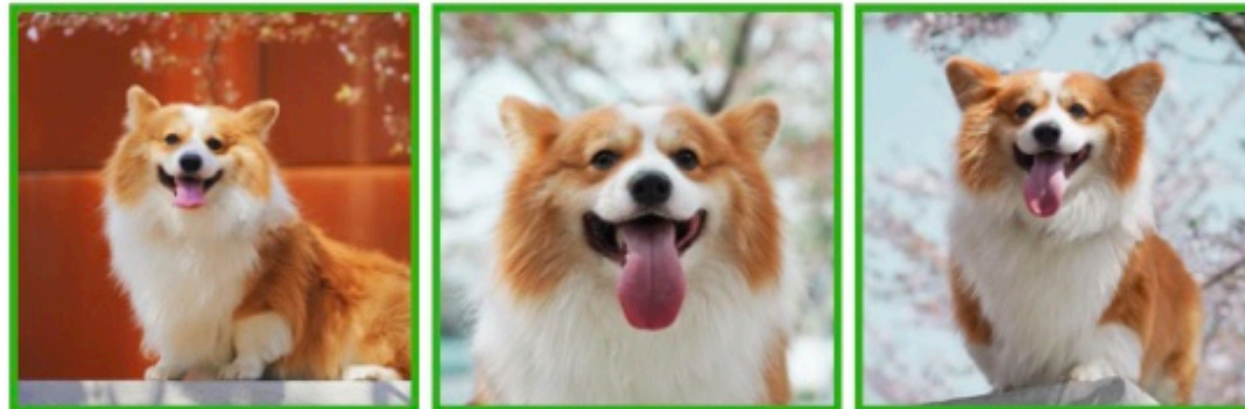
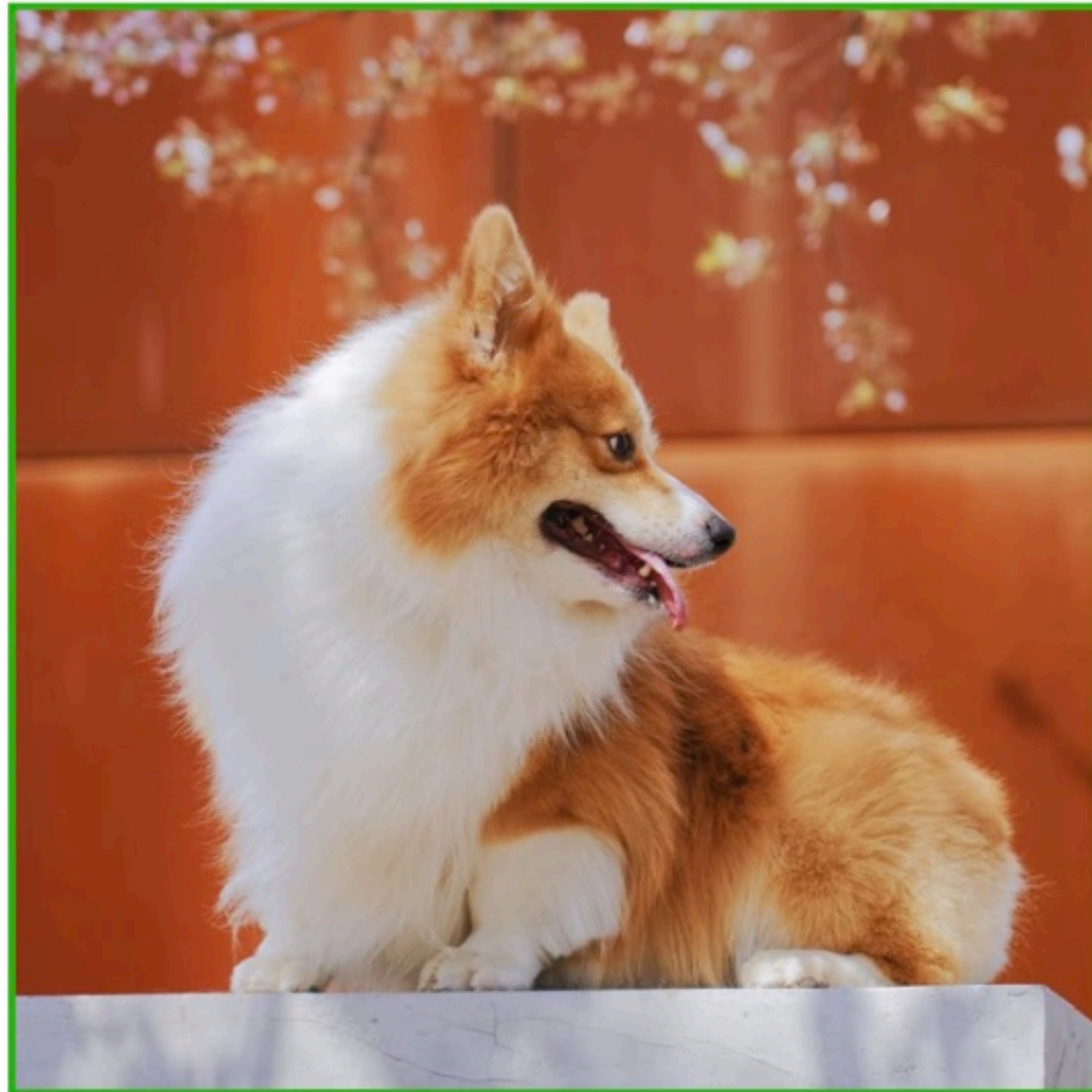
1. Start with a guide image (a target)
2. Add “small” amount of noise
3. Iteratively denoise to produce sample from  $p(x)$

“Guide toward a visual target”

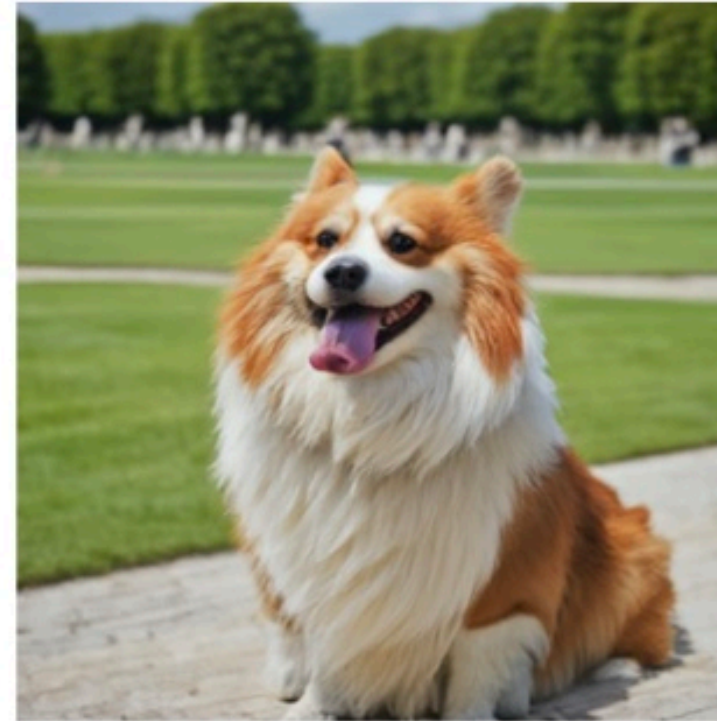


# Specialization to a specific concept

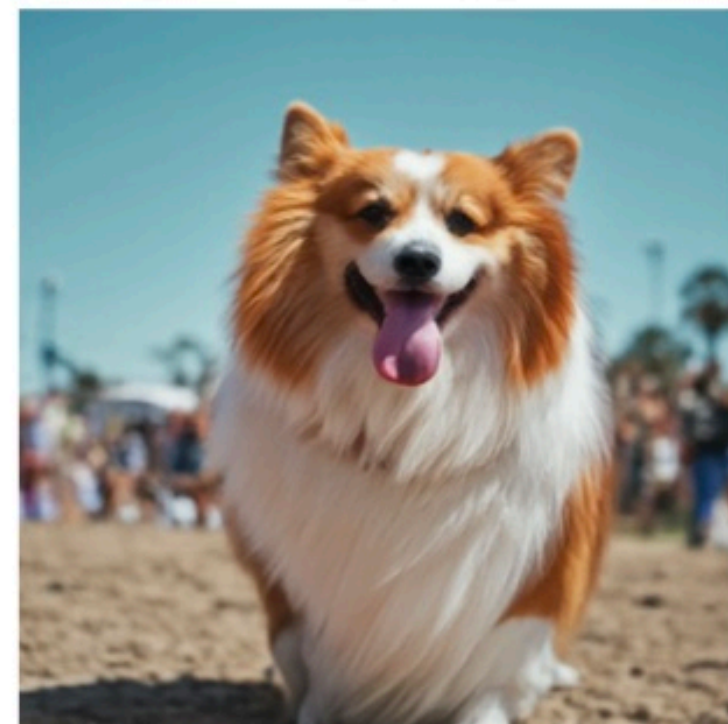
Input images



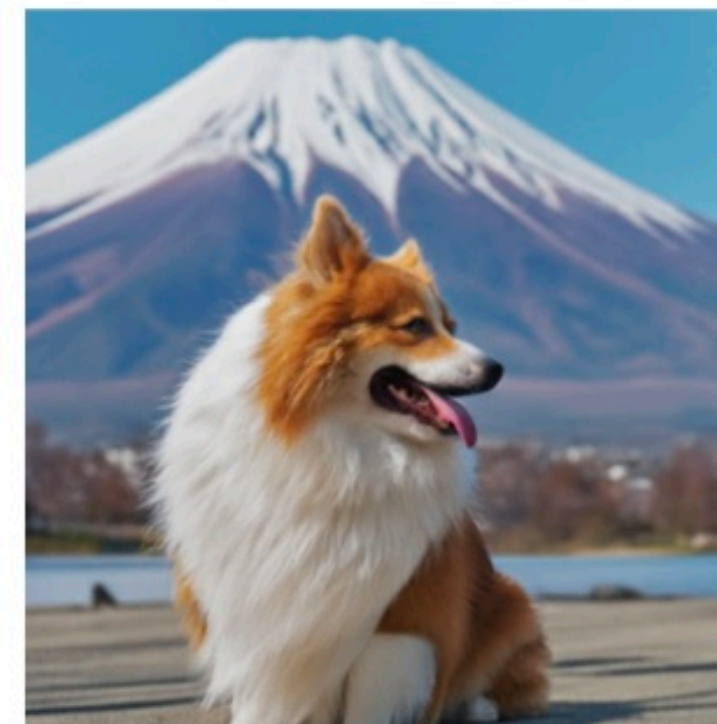
A [V] dog in the Versailles hall of mirrors



A [V] dog in the gardens of Versailles



A [V] dog in Coachella



A [V] dog in mountain Fuji



A [V] dog with Eiffel Tower in the background

# Specialization to a concept

Input images



A [V] sunglasses in the jungle



A [V] sunglasses worn by a bear



A [V] sunglasses at Mt. Fuji



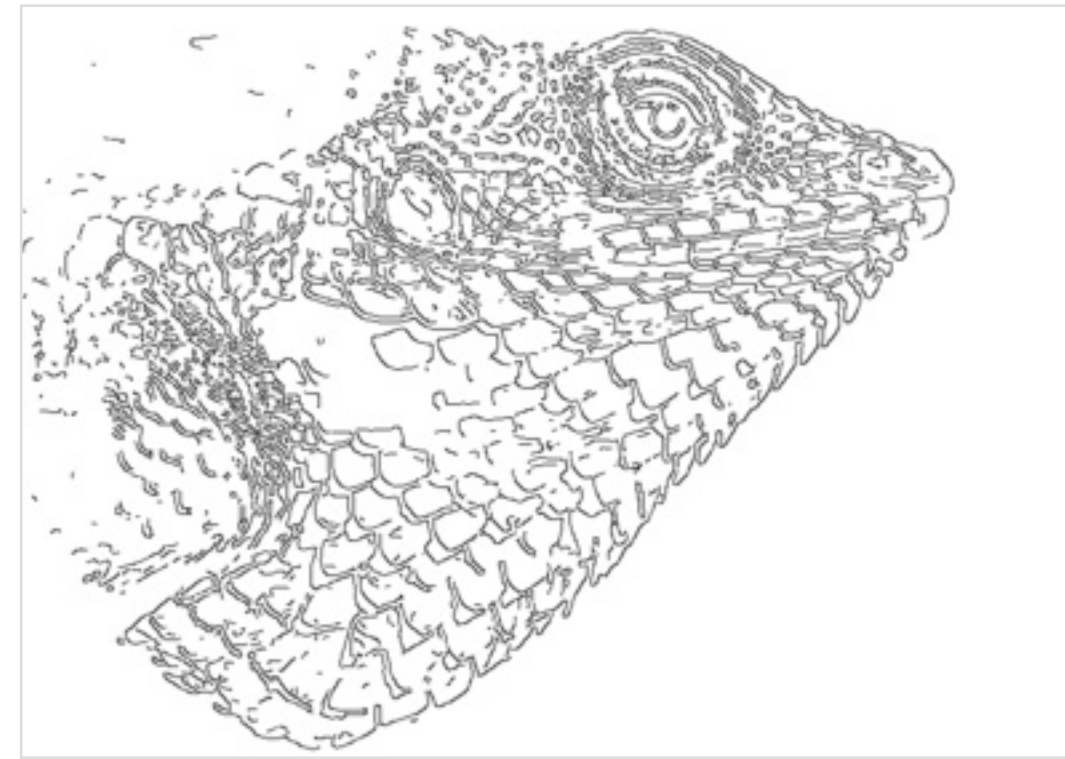
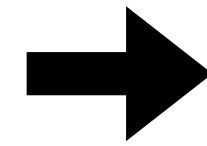
A [V] sunglasses on top of snow



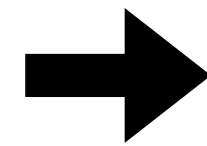
A [V] sunglasses with Eiffel Tower in the background

# Other forms of conditioning

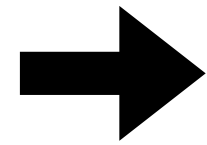
Common tactic: automatically create paired data (via image processing/analysis)



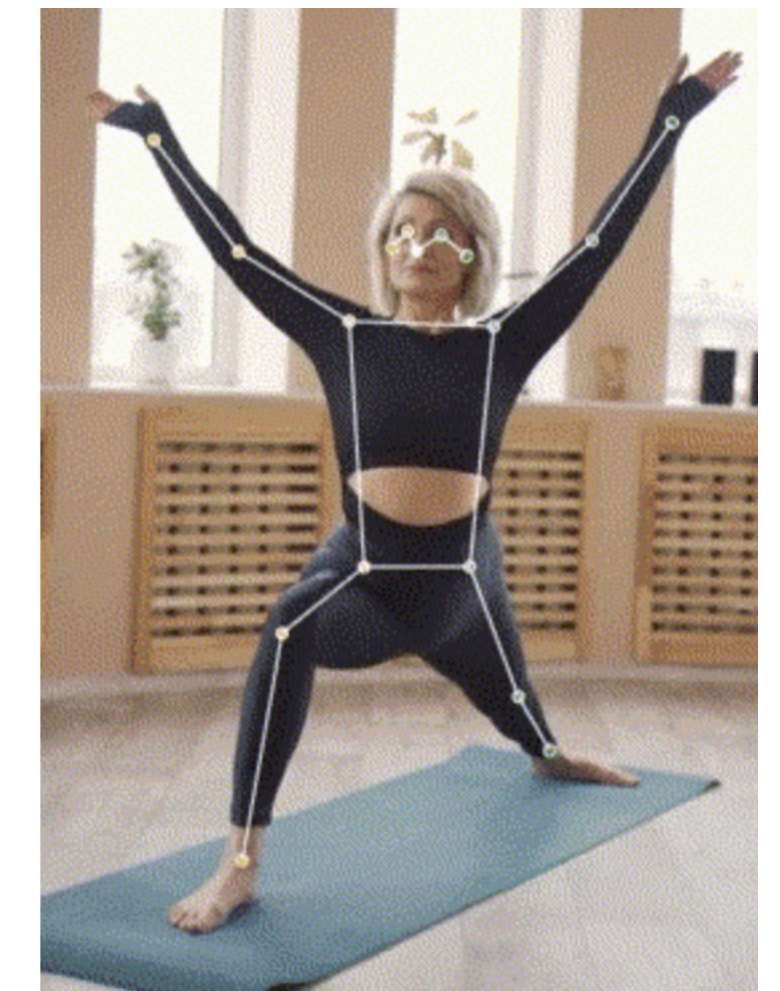
Edge detection



Segmentation



Depth



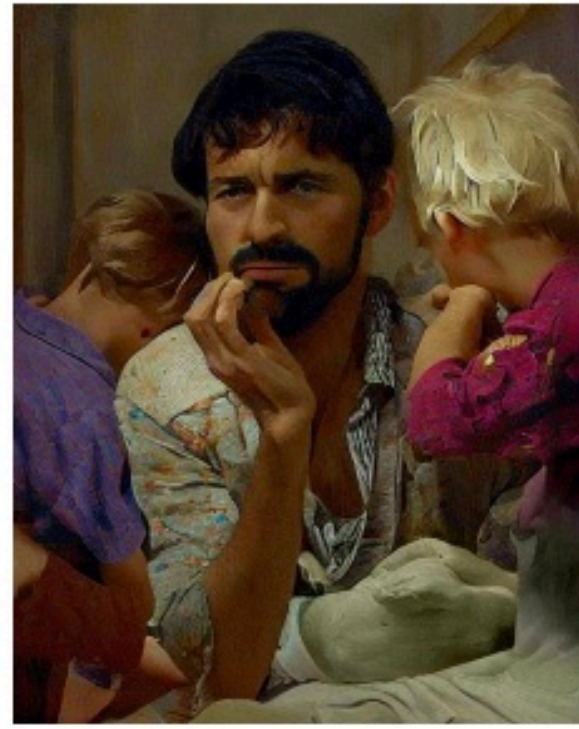
Another example:  
Pose Estimation

# Alternative forms of control

Input (Canny Edge)



Default



Automatic Prompt



“a man with beard sitting with two children”

User Prompt



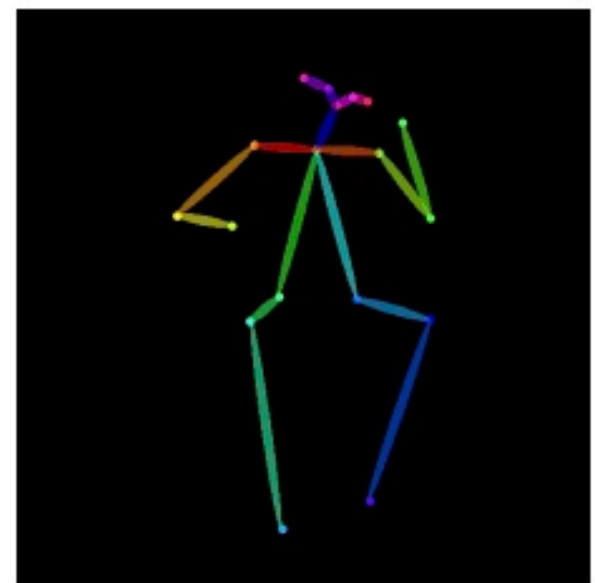
“mother and two boys in a room, masterpiece, artwork”



“a building in a city street”



“inside a gorgeous 19th century church”



astronaut

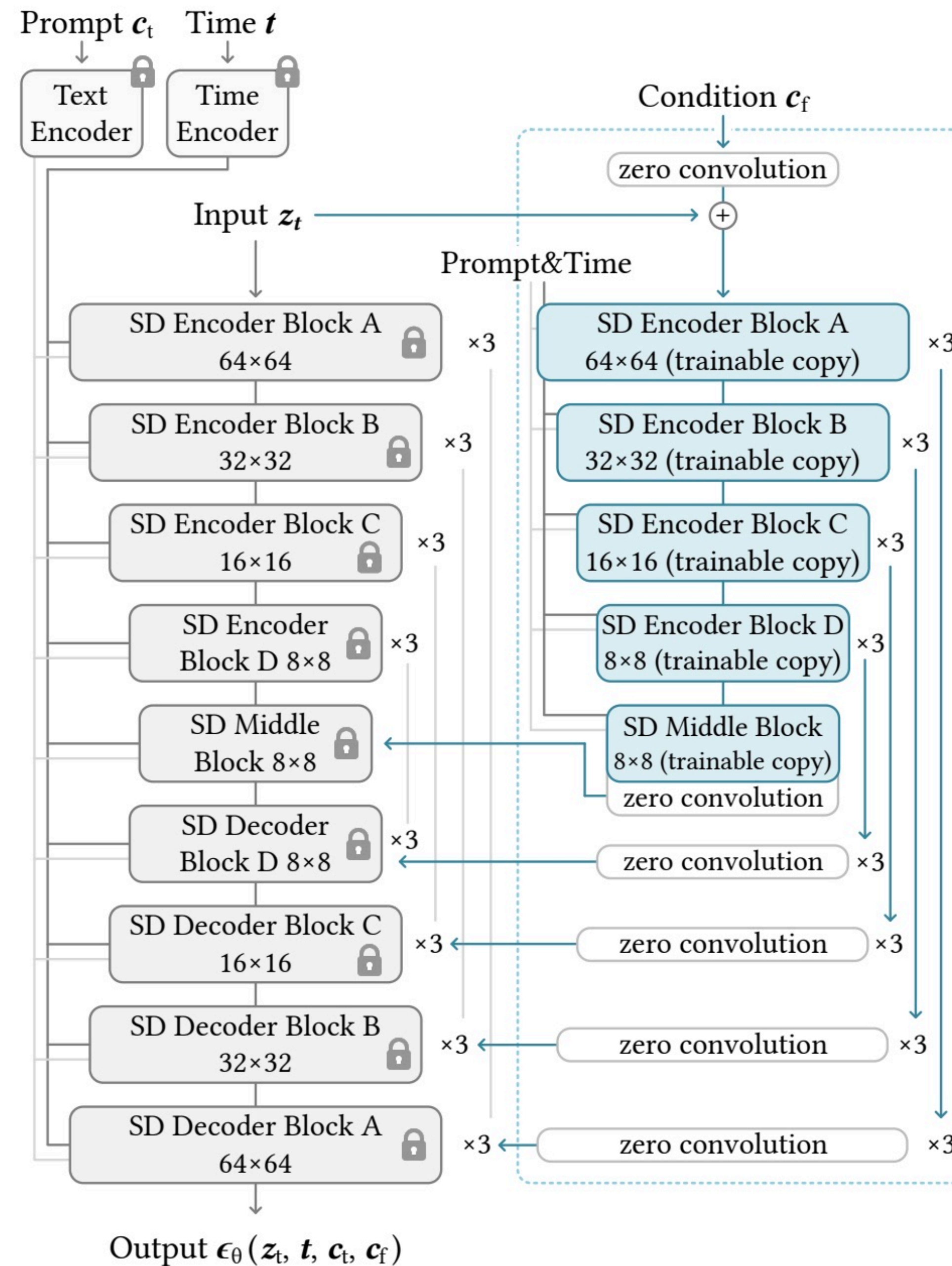
“music”

# Reducing the cost of training

- **Example open source text/image training dataset:**
  - **LAI0N-5B (5.5B image/test training pairs)**
  
- **Significant compute cost to train a model**

# ControlNet

- **Key idea: duplicate diffusion network**
  - **Original copy retains original weights**
  - **ControlNet copy learns how to modify activations to respect new control signals**
- **Intuition:**
  - **Retain strong priors of backbone trained on large body of images (expensive training)**
  - **Learn how to respect new control signal from a much smaller number of images and few optimization steps (inexpensive)**

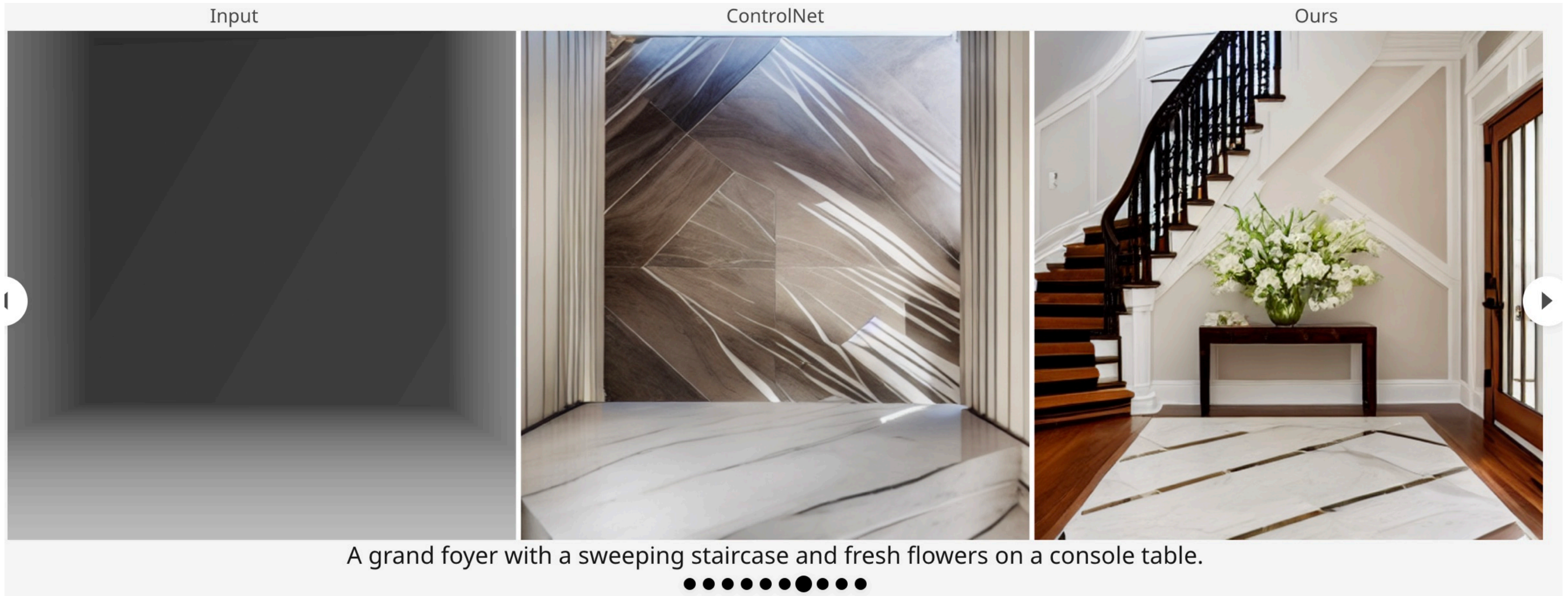


(a) Stable Diffusion

(b) ControlNet

**I kind of have an idea of what I want visually,  
but it is challenging to produce the conditioning**

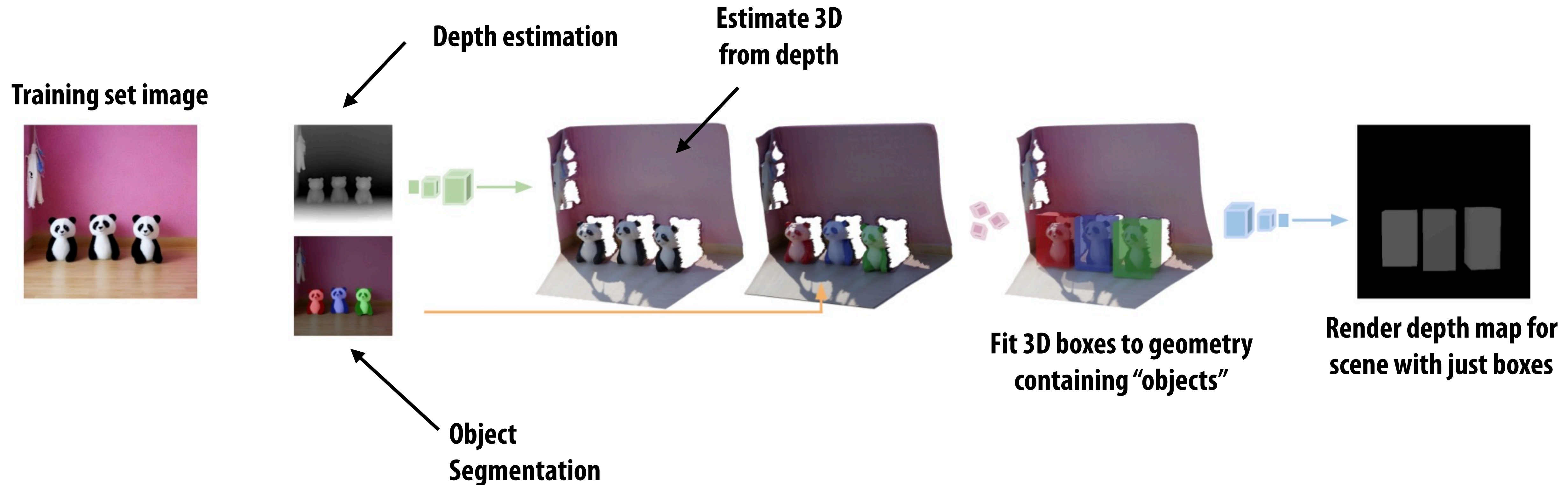
# Loose control



**Key idea: user does not want to (or may not have capability to) specific visual controls precisely. Just have the user “block out” the basic shape of the scene.**



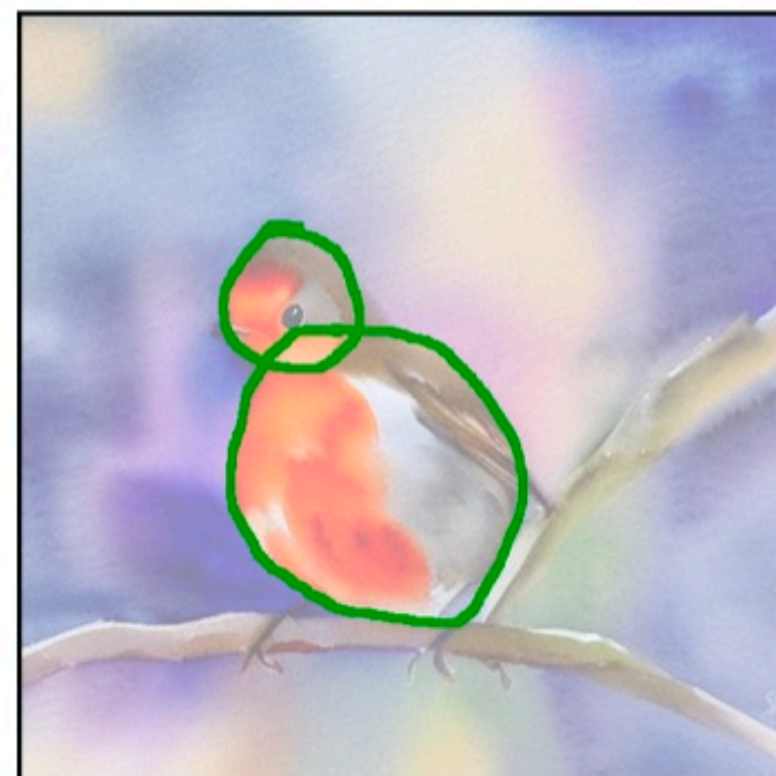
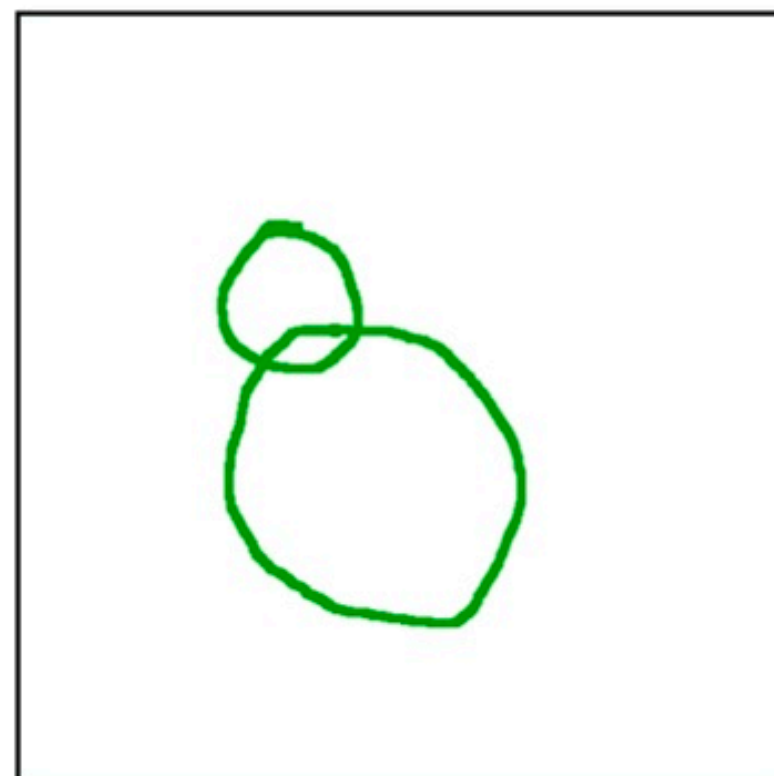
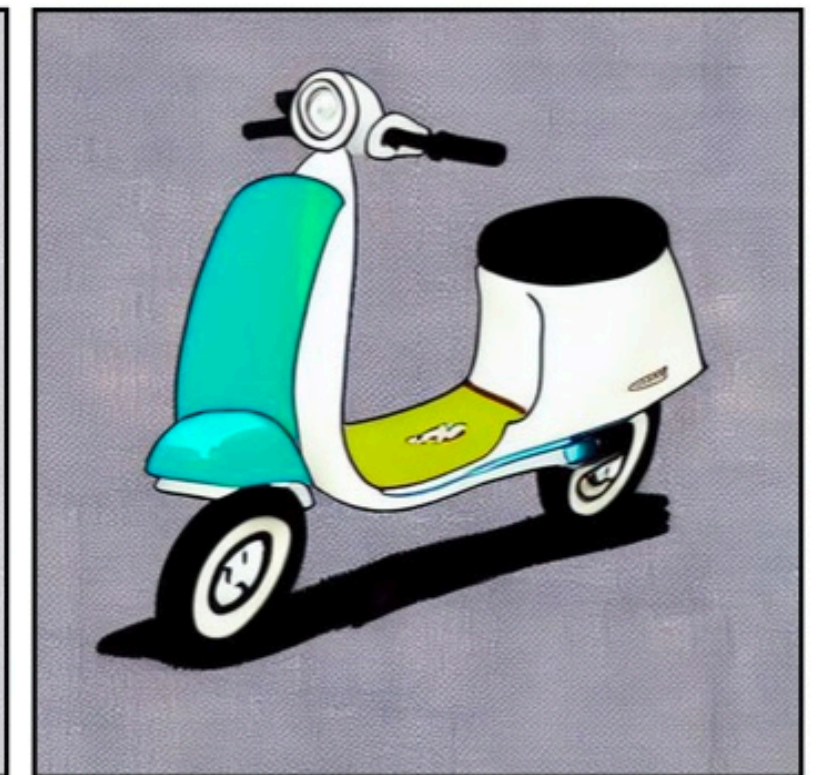
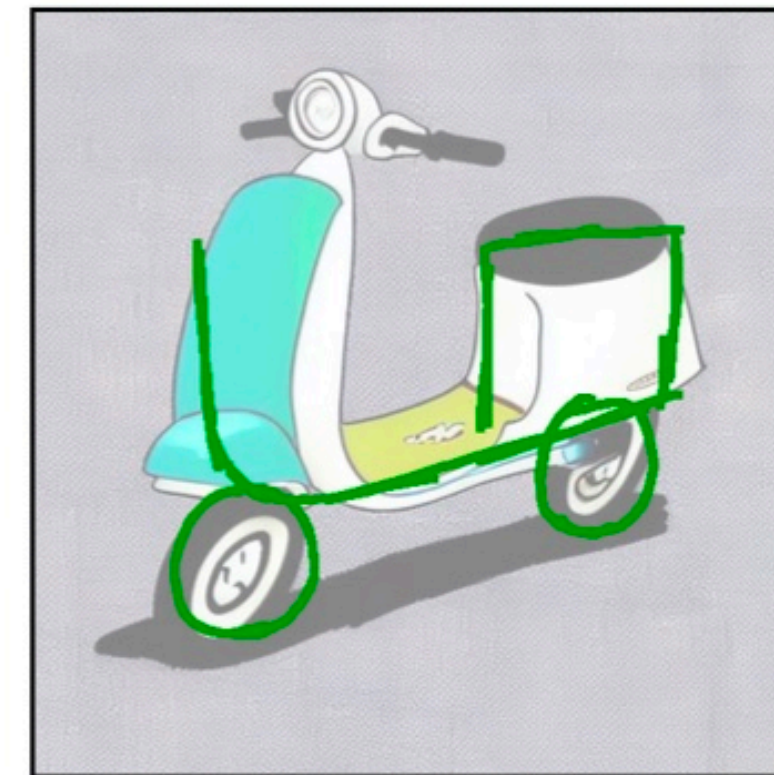
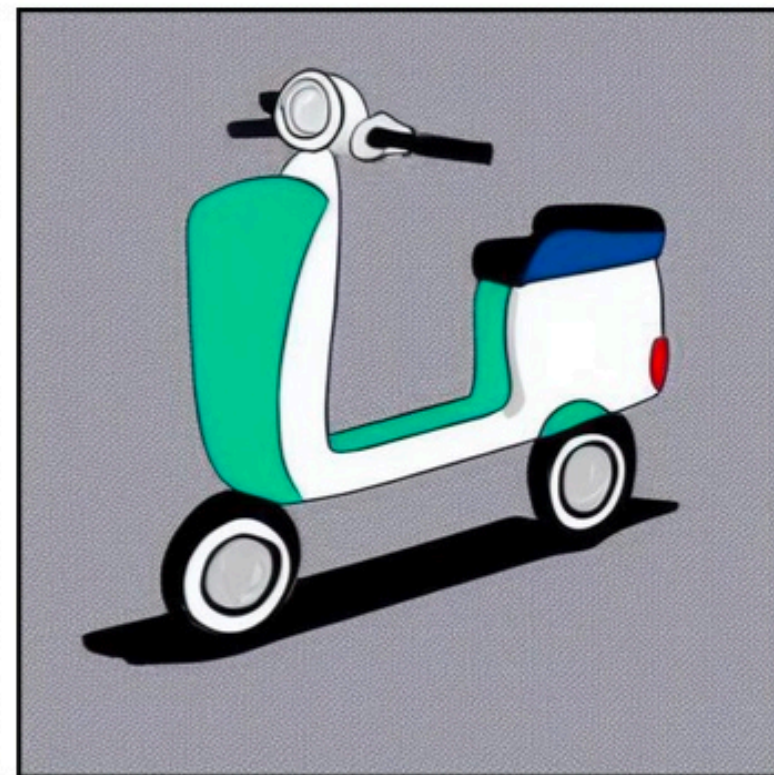
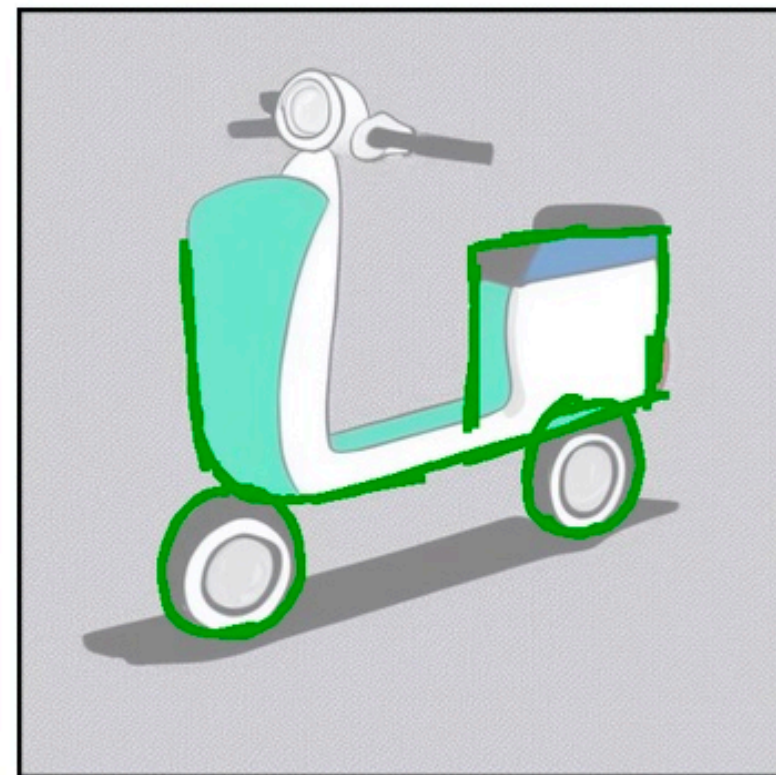
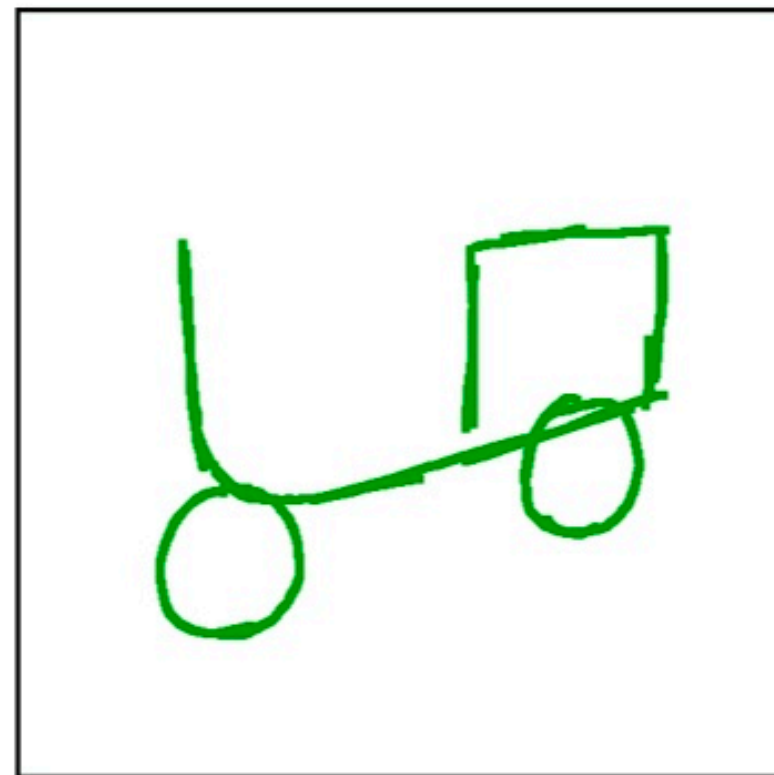
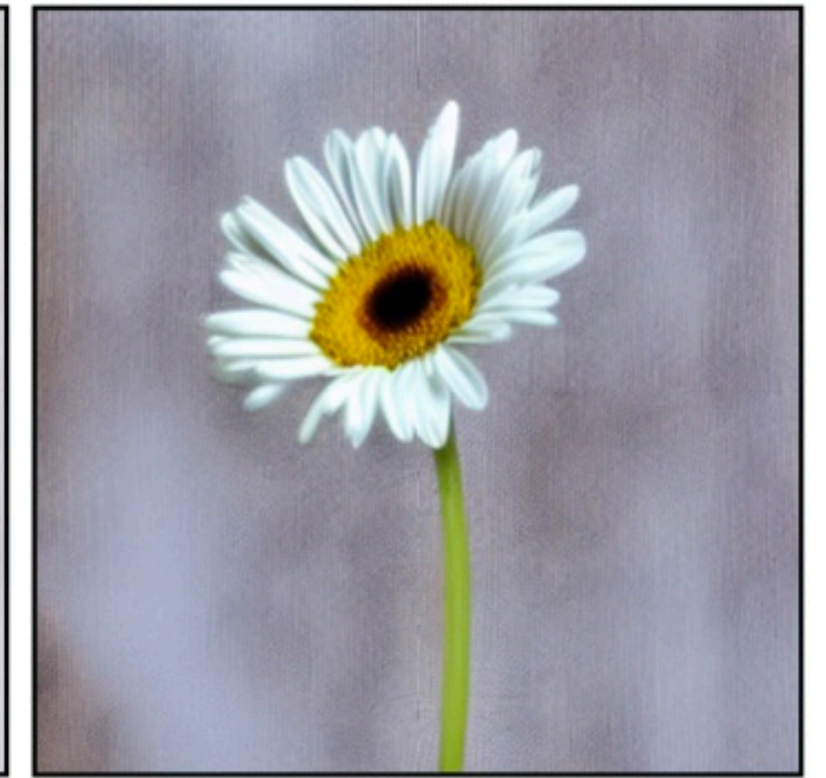
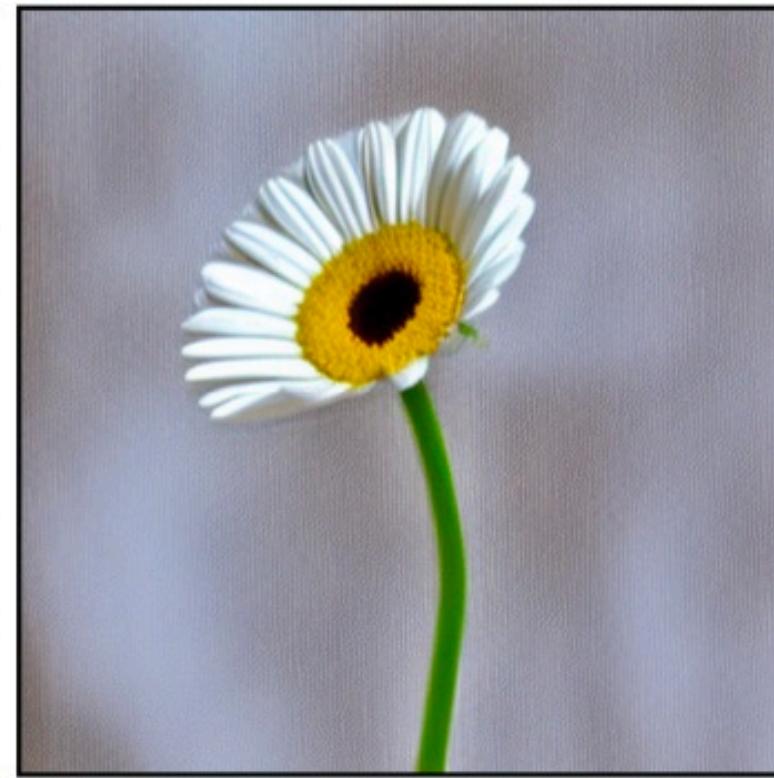
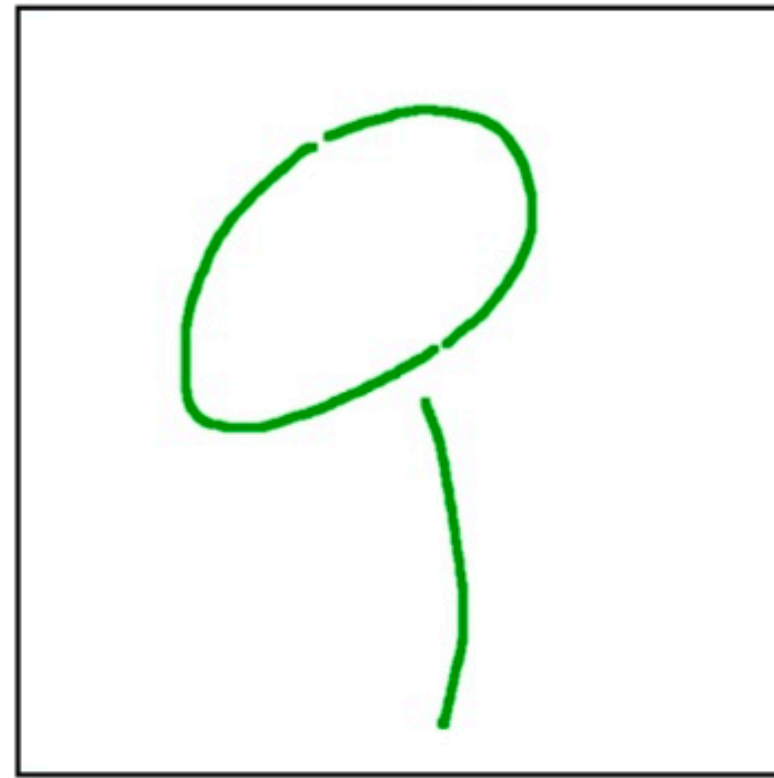
# Generating training data for loose control



# Sketch-to-image

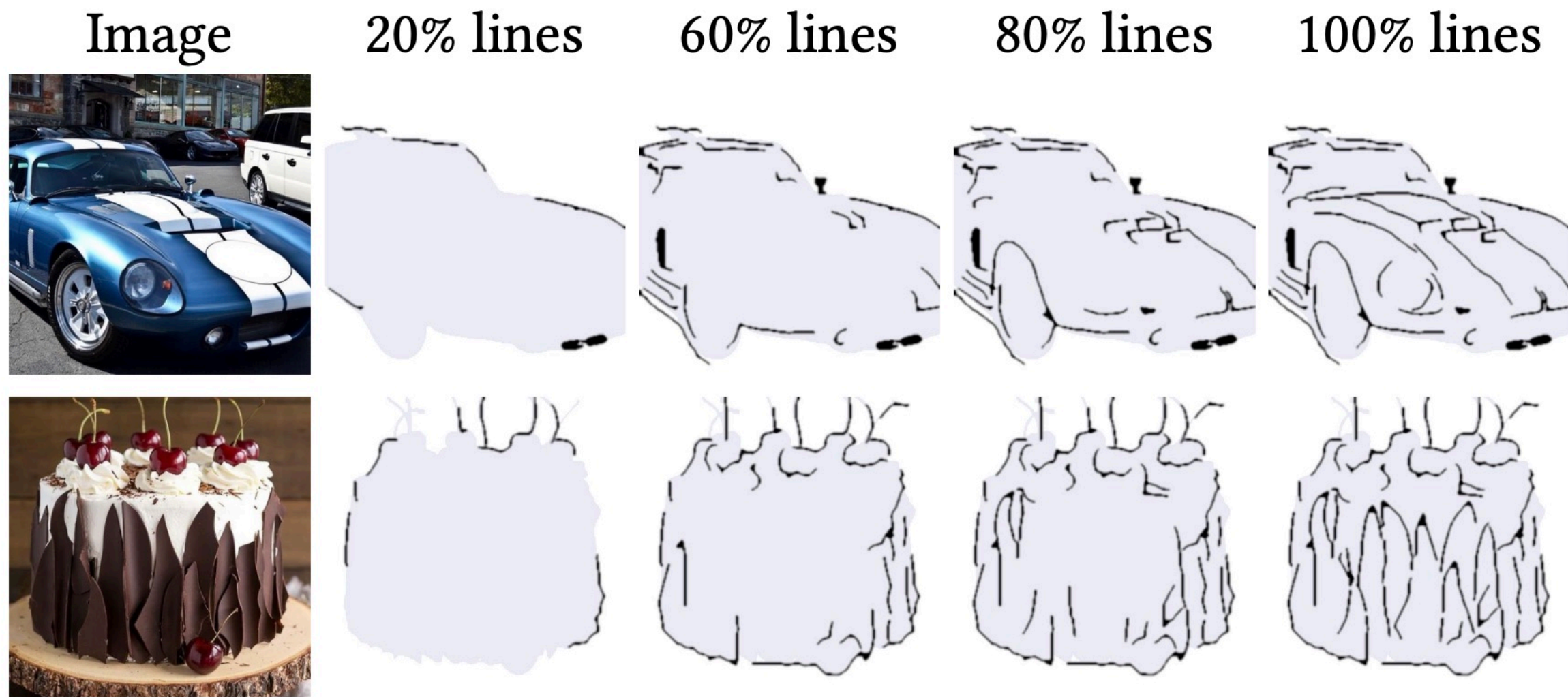
Tight control (ControlNet)

Looser control (Blended Renoising)



# Partial sketch to image training data generation

## Creating "partial-sketch" training data



**Find edge lines**

**Identify contour lines (via segmentation)**

**Progressively remove edge lines that are farthest from contours.**



**I want to modify the output in a SPECIFIC REGION.  
I want to control COMPOSITION.**

# Inpainting (apply [new] prompt to a region)

User specifies mask for region of interest and text prompt for that region.

Image outside of region remains almost the same.



"bowl of water"



"stool"



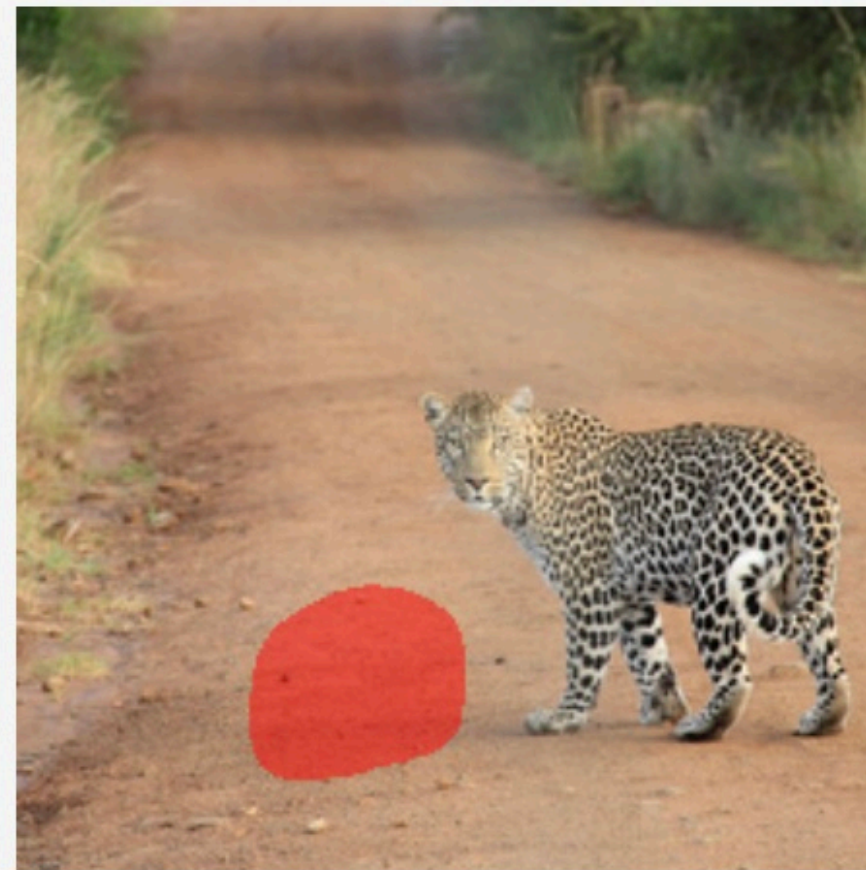
"hole"



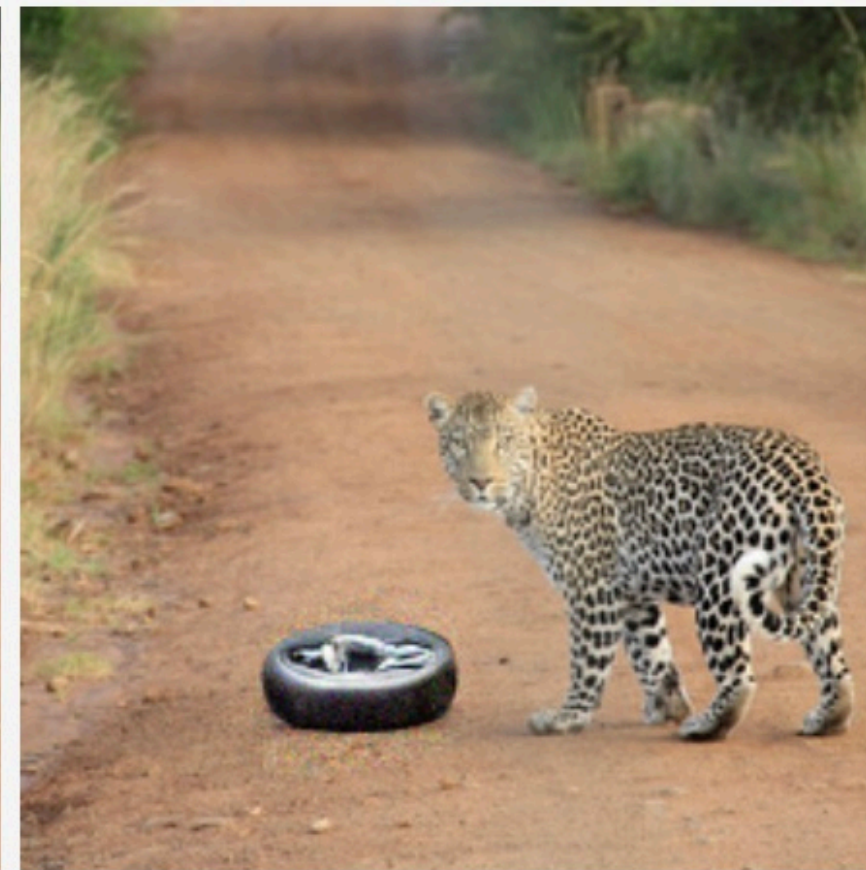
"red brick"



Input image



Input mask



"car tire"



"big stone"

# Use change in text prompt to trigger change in image

“A basket full of apples.”



Source image



apples → cookies



basket → bowl



basket → box



basket → nest



apples → oranges

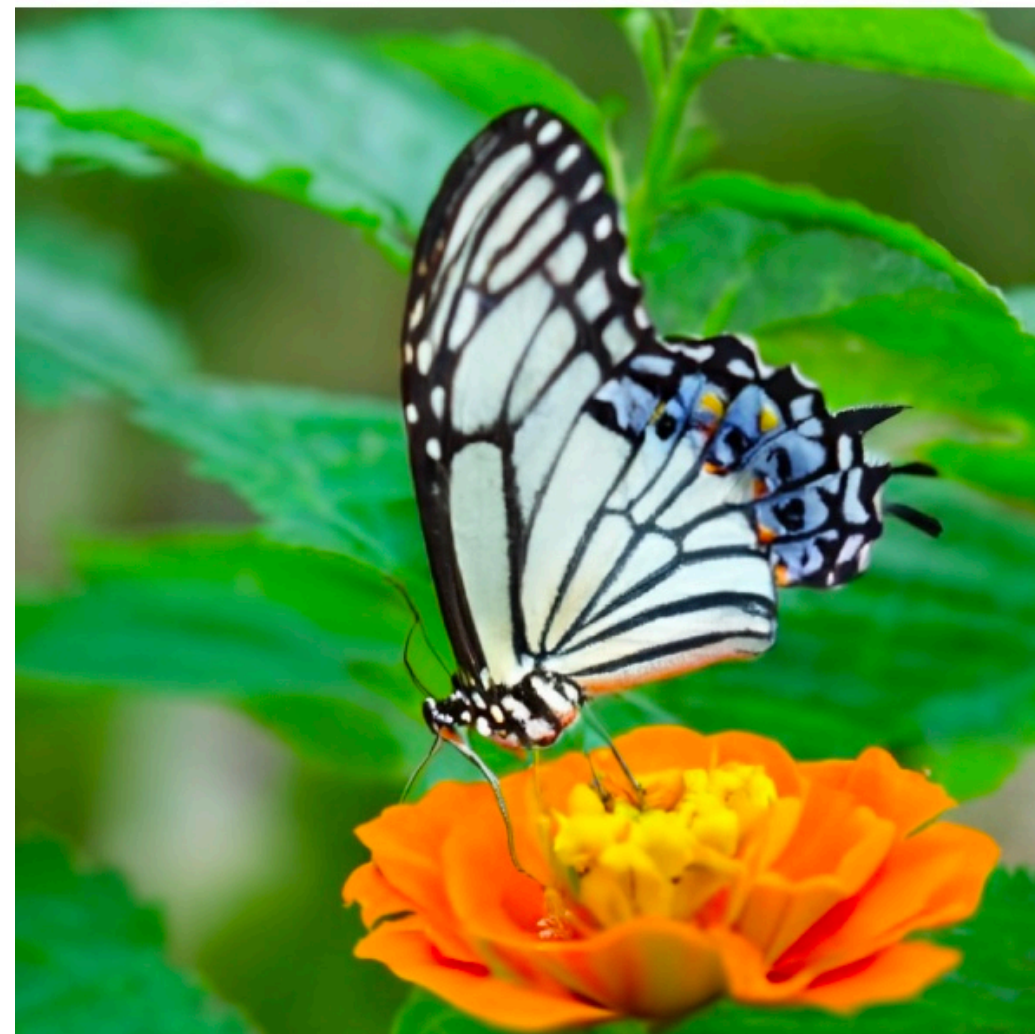


apples → chocolates



apples → kittens

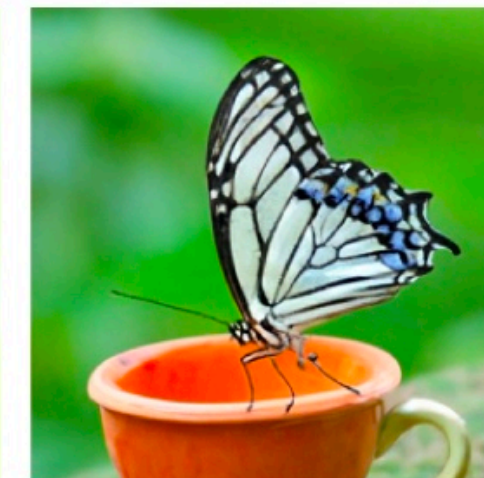
“A photo of a butterfly on a flower.”



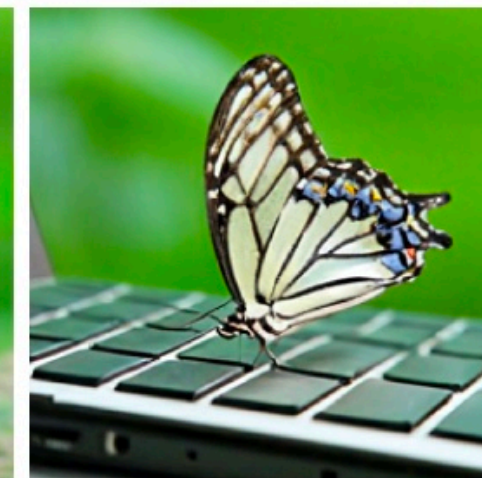
Source image



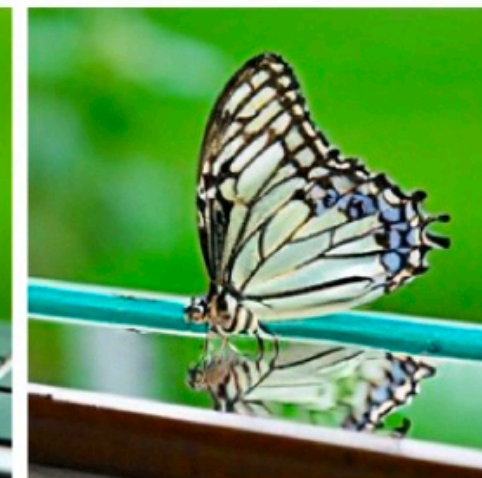
flower → bread



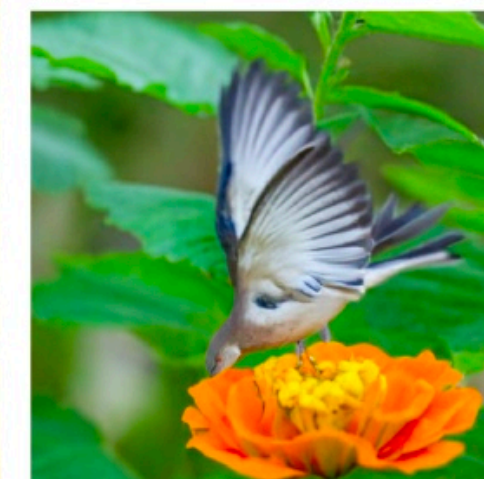
flower → mug



flower → computer



flower → mirror



butterfly → bird



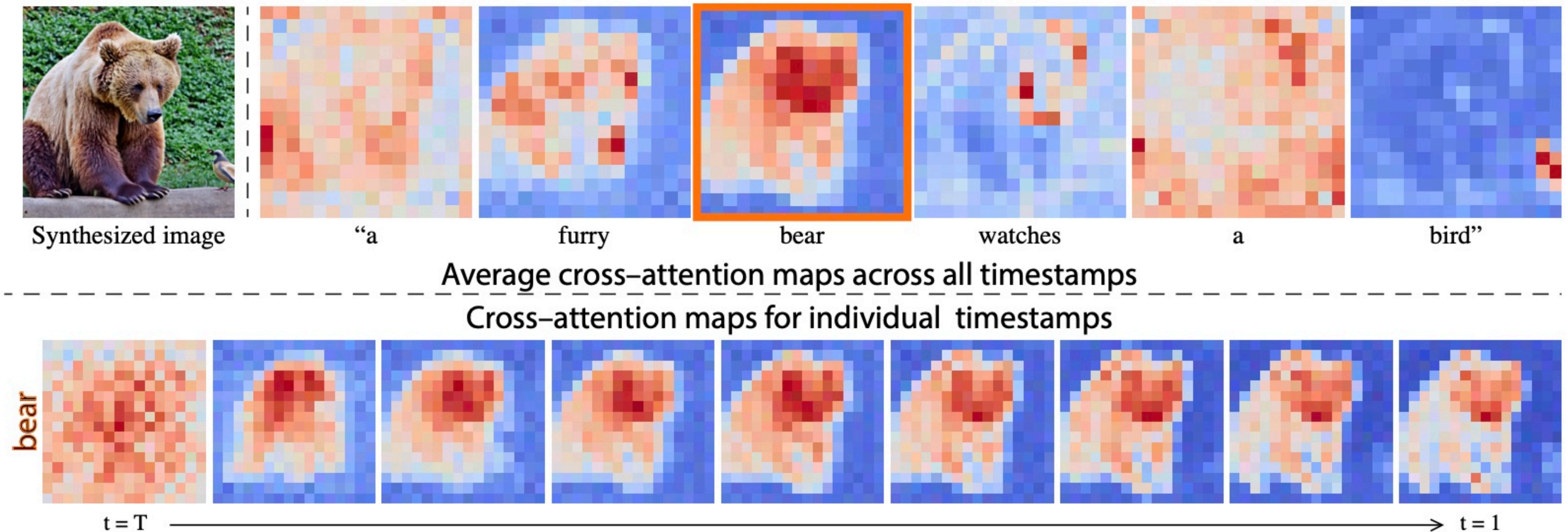
butterfly → snail



butterfly → drone

# “Masks” come from learned attention

- Use attention masks from original generation process to constrain what pixels can change after prompt is edited



# Using text to describe how to change the image

*"Swap sunflowers with roses"*



*"Add fireworks to the sky"*



*"Replace the fruits with cake"*



*"What would it look like if it were snowing?"*



*"Turn it into a still from a western"*



*"Make his jacket out of leather"*

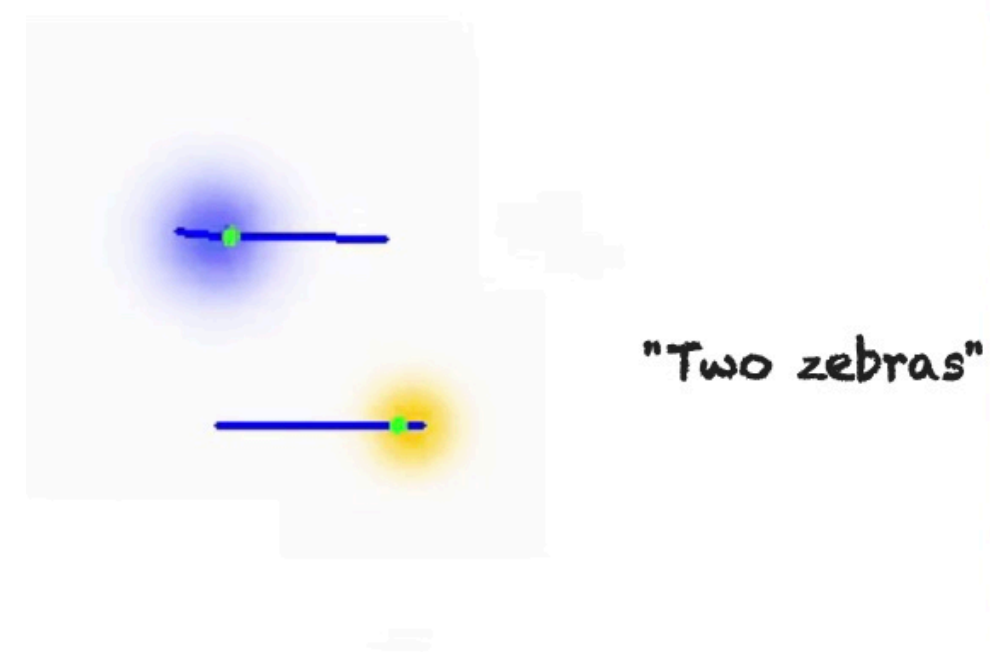


# Challenge of control is spawning similar work in other modalities

- **Video**
- **Character Animation**
- **3D generation**
- **Scene generation**
- **Etc, etc, etc.**

# Controlling video generation

- What are the right controls?
  - Object motion control
  - Camera motion control



# Text-conditioned animation generation

"A person walks forward, bends down to pick something up off the ground."

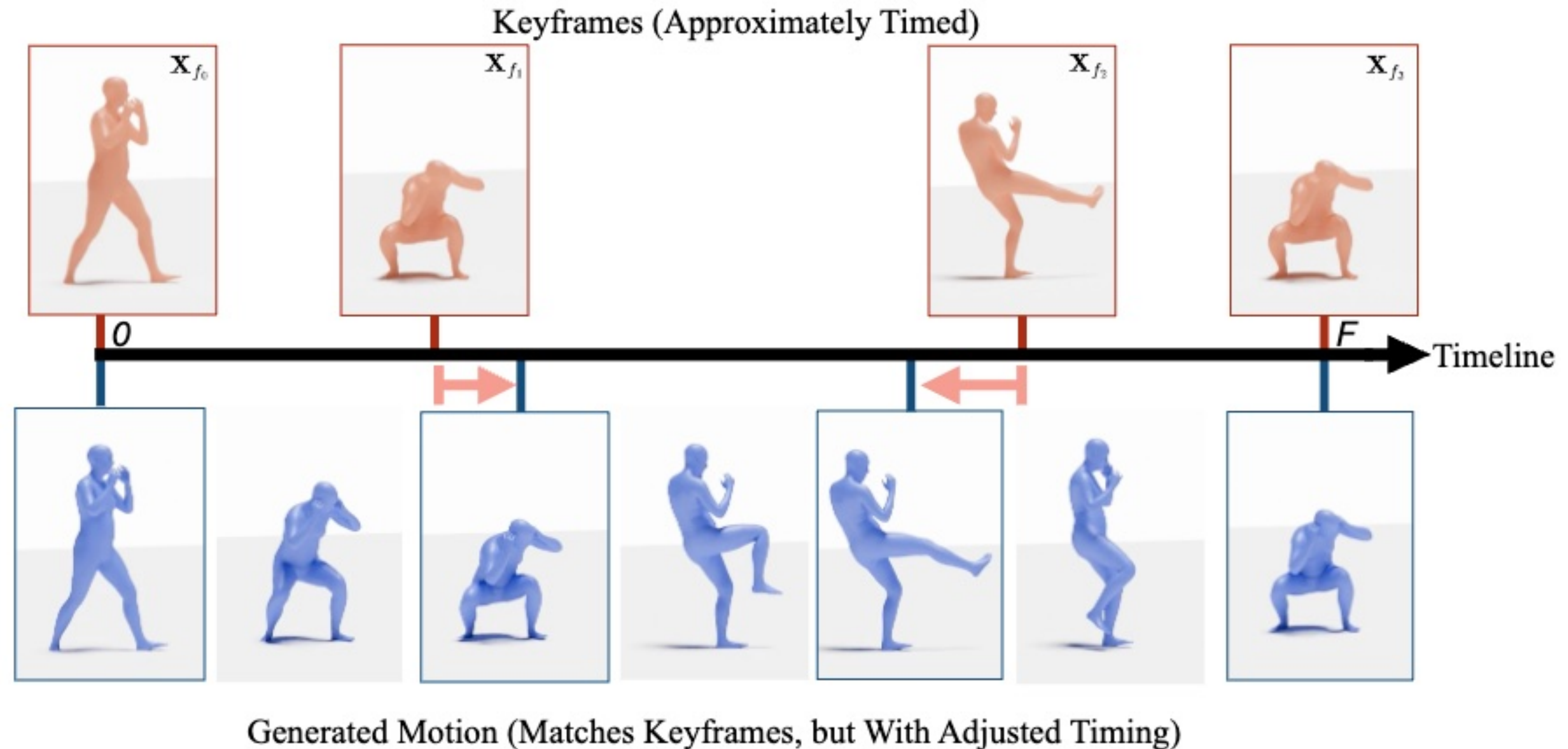


# Audio-conditioned dance animation generation



# Imprecise keyframe conditioned generation

- But keyframes don't have to be correctly placed on the timeline



# Reminder: key aspect in the design of any system

## Choosing the “right” representations for the job

- **Good representations are productive to use:**
  - Embody a “preferred” way of thinking about a problem
- **Good representations enable the system to provide **useful services:****

  - Validating/providing certain guarantees (correctness, resource bounds, conversion of quantities, type checking)
  - Performance optimizations (parallelization, vectorization, use of specialized hardware)
  - Implementations of common, difficult-to-implement functionality (complex array indexing code, texture mapping in 3D graphics, auto-differentiation, etc.)
  - **Provide the user certain types of control over generation**

# **Here: select the controls that are most useful to an editing task is the first step toward choosing appropriate representations**

- **What is the type of control that aligns with the users thought process / mental model?**
  - **Text is often an ambiguous, imprecise, or flat out inefficient way to describe visual intent**
- **Examples:**
  - **Users want to control spatial composition**
    - **“Dog on the left” vs. dragging a layer to the right location**
  - **Users want to “block out” an idea, and have the diffusion model “fill in the details”, “correct proportions”, “harmonize the image”**
  - **Users want to express intent via an example: “I want it to look LIKE THIS!”**

## **Discussion:**

**Propose a type of edit that you would like to make to images/video/animation/3D models**

**How does the user “think” about what they are trying to change (are they worried about details, composition, a particular “axis” of change (e.g, adjust smile but not eyes))**

**How could to generate supervision to train a model to support this type of control?**

# **Neurosymbolic methods: combining traditional symbolic representations with learned representations**

# An increasingly common paradigm for generative AI

Reducing generation tasks to the act of writing programs.

High-level specification

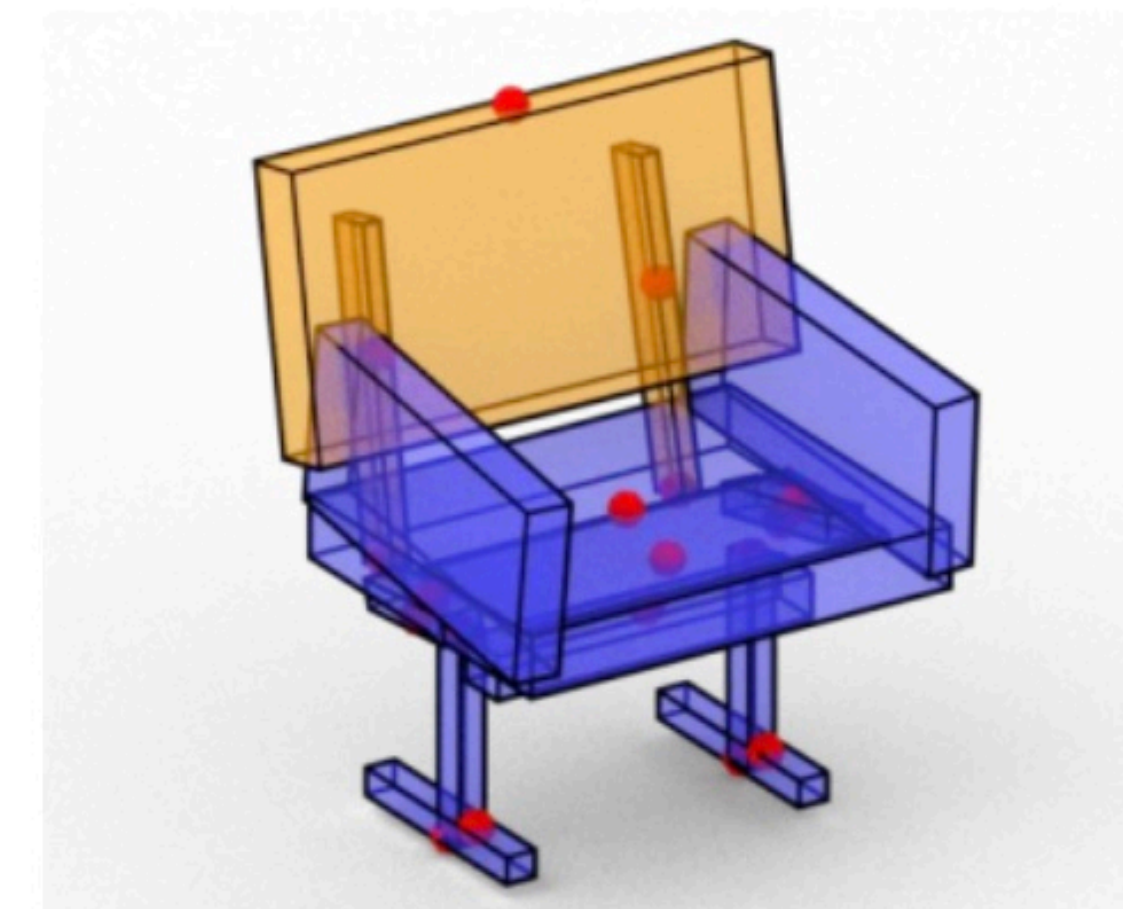
*"Chair with a flat back"*

Domain-specific language providing  
useful primitives  
(often with precise semantics)

Program generation  
engine  
(e.g.: LLM, DNN)

Execute program

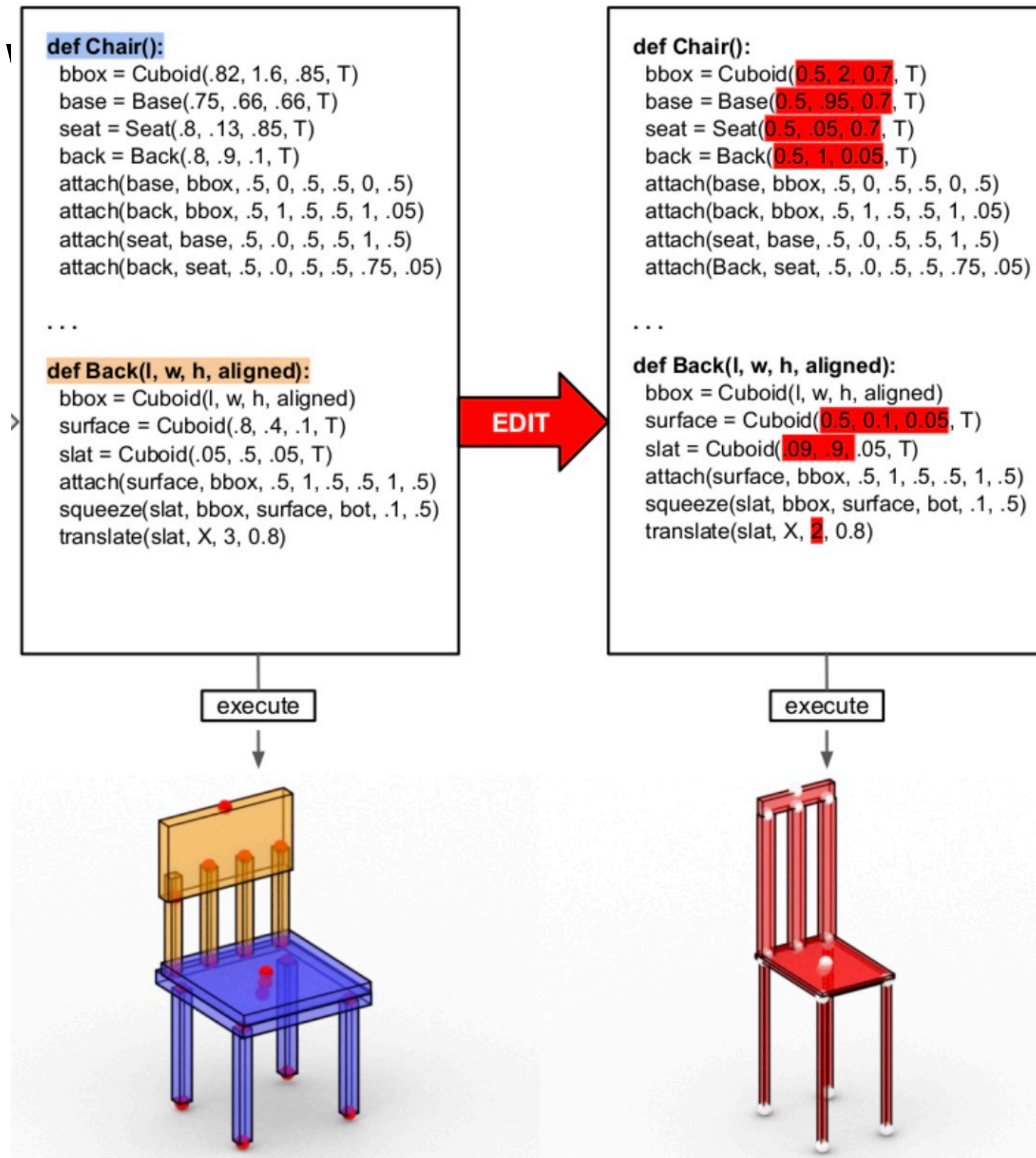
Generated output



```
def Chair():  
  bbox = Cuboid(1.2, 1.4, 1, T)  
  base = Base(.9, .5, .8, T)  
  seat = Seat(1.1, .1, .9, T)  
  back = Back(1.1, .9, .2, F)  
  arm = Cuboid(.1, .4, .7, F)  
  attach(base, bbox, .5, 0, .5, .5, 0, .5)  
  squeeze(back, bbox, base, top, .5, .1)  
  attach(seat, base, .5, 0, .5, .5, 1, .5)  
  attach(arm, back, .5, .5, 0, .1, .3, .5)  
  attach(arm, seat, .5, 0, .5, .1, .7, .5)  
  reflect(arm, X)  
  ...  
def Back(l, w, h, aligned):  
  bbox = Cuboid(l, w, h, aligned)  
  surface = Cuboid(1.16, .64, .13, T)  
  slat = Cuboid(.04, .76, .1, F)  
  attach(surface, bbox, .5, 1, .5, .5, 1, .7)  
  attach(slat, bbox, .5, 0, .5, .2, 0, .45)  
  attach(slat, surface, .5, .6, .8, .2, .3, .2)  
  reflect(slat, X)
```

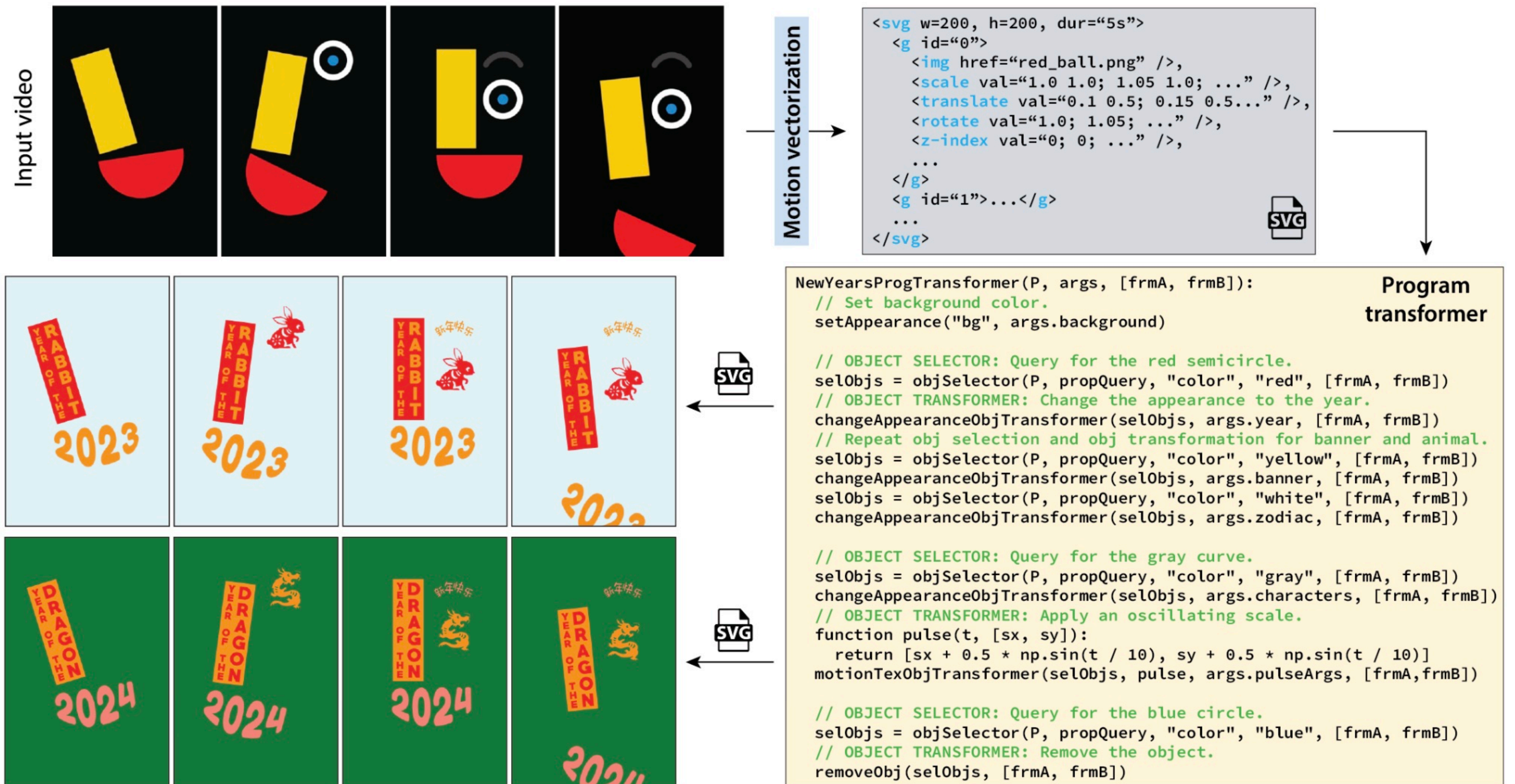
# One benefit: a program is a human interpretable representation

We read and edit programs all the time!  
Reducing generation tasks to the act of



# Editing motion graphics (vector graphics animations)

1. Video —> SVG keyframe animation (computer vision)
2. LLM edits the SVH file (e.g., change yellow rectangle to texture map)
3. Re-render animation



# Another example

Python programs as “plans” to carry c



Notice how the program queries for state of environment and exhibits simple conditional execution.

## PROMPT for Planning

### Import action primitives

```
from actions import walk <obj>,
grab <obj>, switchon <obj>,
open <obj>, standup, find
<obj>, putin <obj> <obj>, ...
```

### Available objects list

```
objects=[clothesshirt' sink,
pie, apple, fridge, garbagecan,
tv, dishwashingliquid, bed,
bookshelf, salmon, stove, plate,
coffeepot, kitchentable,
wineglass, paper, microwave,
toothbrush, toothpaste,
bathroomcabinet, kitchen, lime,
painkillers, barsoap, ...]
```

### Example task(s)

```
def throw_away_lime():
# 0: find lime
find('lime')
...
# 5: close garbagecan
assert('close' to 'garbagecan')
else: find('garbagecan')
assert('garbagecan' is 'opened')
else: open('garbagecan')
close('garbagecan')
...
# 6: Done
```

### Next task prompt

```
def microwave_salmon():
```

its

LLM code generator

## Generated Plan

```
def microwave_salmon():
# 0: walk to kitchen
walk('kitchen')
# 1: find microwave
find('microwave')
# 2: open microwave
assert('close' to 'microwave'
else: find('microwave')
assert('microwave' is 'closed')
else: close('microwave')
open('microwave')
...
# 5: put salmon in microwave
assert('salmon' in 'hands')
else: find('salmon')
else: grab('salmon')
assert('close' to 'microwave')
else: find('microwave')
assert('microwave' is 'opened')
else: open('microwave')
putin('salmon', 'microwave')
...
close('microwave')
...
switchon('microwave')
# 8: wait for salmon to be done
cooking 1-2 minutes
# 9: turn off microwave
...
switchoff('microwave')
...
open('microwave')
...
grab('salmon')
# 12: put salmon on plate
assert('salmon' in 'hands')
...
assert('close' to 'plate')
else: find('plate')
putin('salmon', 'plate')
...
close('microwave')
# 14: Done
```

Comment: Summary

Action API + object(s): Plan Step

Comment: Action(s) Mapping

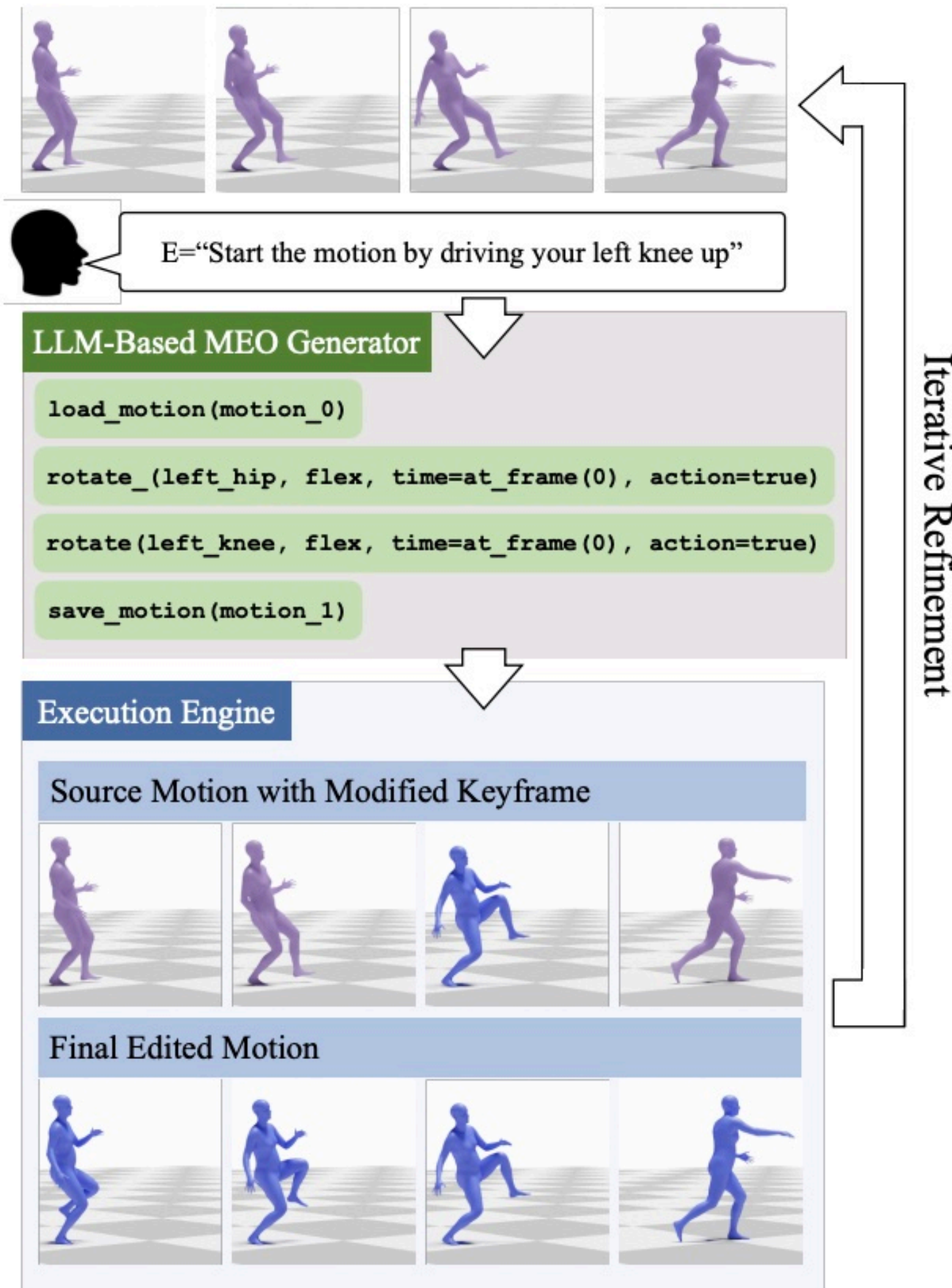
Assertions: State Feedback

Else: Recovery Actions

Optional Steps

# Generating animations

High-level motion editing commands —> programs that perform keyframe edits —> use AI model (diffusion) to interpolate the keyframes



## API for MEO Construction

```
from actions import translate_joint, rotate_joint, ...  
from timing import when_joint, at_frame, ...  
from motion_io import load_motion, save_motion
```

## Available Parameters

```
relative_moments=["highest", "lowest", ...]  
translate_directions=["forward", "backward", "up", ...]  
rotation_directions=["abduct", "adduct", "extend", ...]  
joints_that_rotate=["right_knee", "left_knee", ...]
```

## In-context Learning Example(s)

```
# the person is jumping. Bring the right knee to chest  
# as you jump  
def right_knee_to_chest():  
    # load the motion that needs to be edited  
    load_motion("motion_0")  
    ...  
    # bend the right knee  
    rotate_joint("right_knee", "flex",  
                time=when_joint("waist", "highest"))  
    # flex the right hip to bring the knee higher  
    rotate_joint("right_knee", "flex",  
                time=when_joint("waist", "highest"))  
    ...  
    # save the edited motion  
    save_motion("motion_1")
```

## Editing Instruction E

```
# A person is doing a side kick with the right leg.  
# Can you get that kick higher out?
```

**LLM completes code here**

# Another example

## Programs as “plans” to answer questions

Q: Is the carriage to the right of a horse?



Large Language Model

```
answer = "no" # default answer
horse_exists = query("Is there a horse?")
if horse_exists == "yes":
    x1, y1 = get_pos("carriage")
    x2, y2 = get_pos("horse")
    if x1 > x2:
        answer = "yes"
```

Object  
Localizer

Simple VQA  
Method

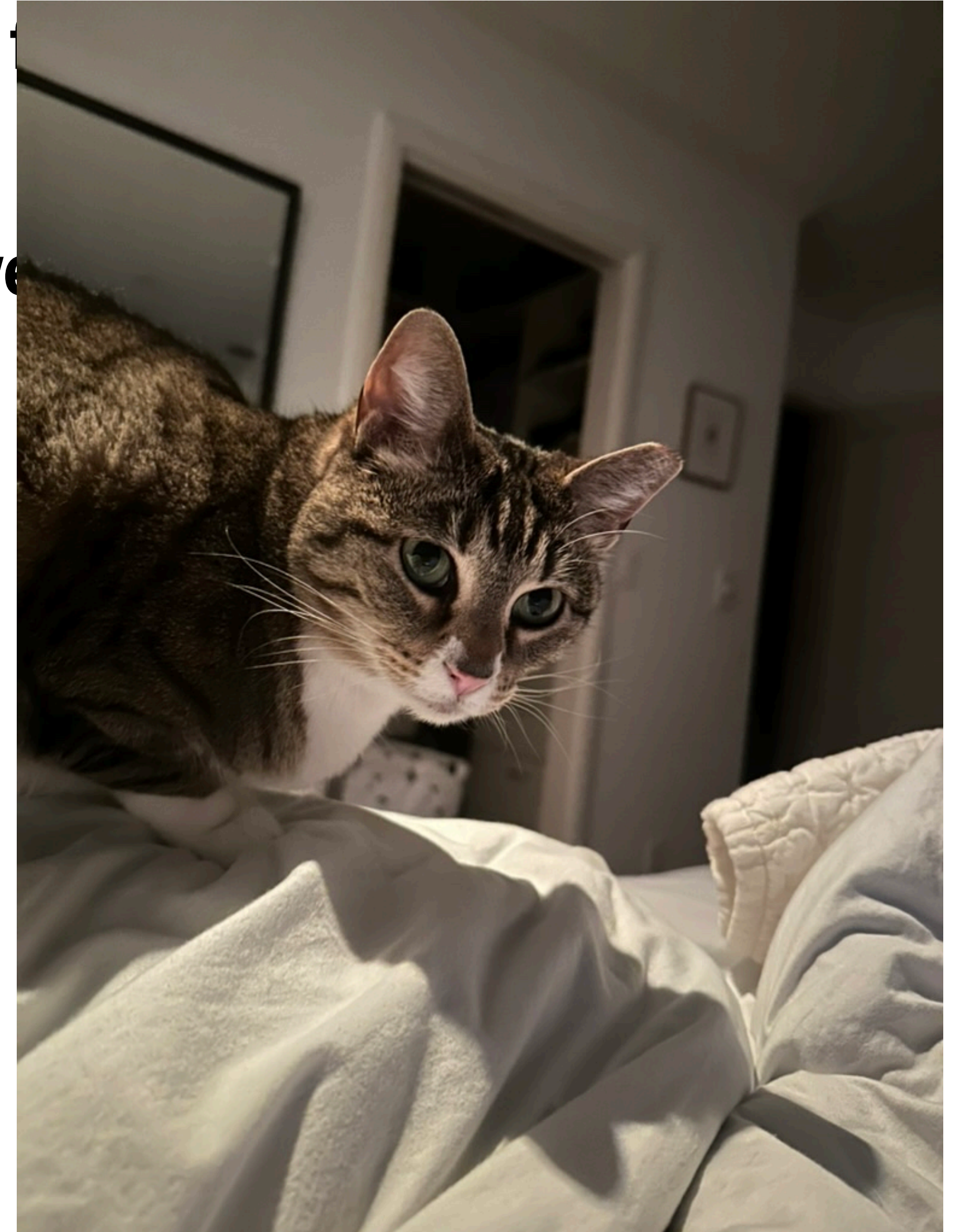
Note how the DSL contains primitives that themselves might be implemented as “Black-box” DNNs

# What is the DSL for the following edit?

- Given this photo, make the cat's tail curve back into the t
- Let's think about the pros and cons of this program-drive

**What are the benefits?**

**What are limitations of this approach?**



# Challenges

- **Designing a domain-specific language for a task can be challenging**
  - **What primitives to include? How to implement these primitives?**
  - **Problems must break down into clearly defined, self-contained steps**
  
- **How do we know when a learned program generator produces a valid program (a program that performs the task specified in the controls)?**
  - **Can we predict when a program generator will fail?**

# MoVER: tonight's reading (SIGGRAPH 2025)

## Text prompt

```
<svg> ... </svg>  
Move the orange circle above the rectangular shape.  
In the meantime, rotate the letter H clockwise by 90 degrees.
```

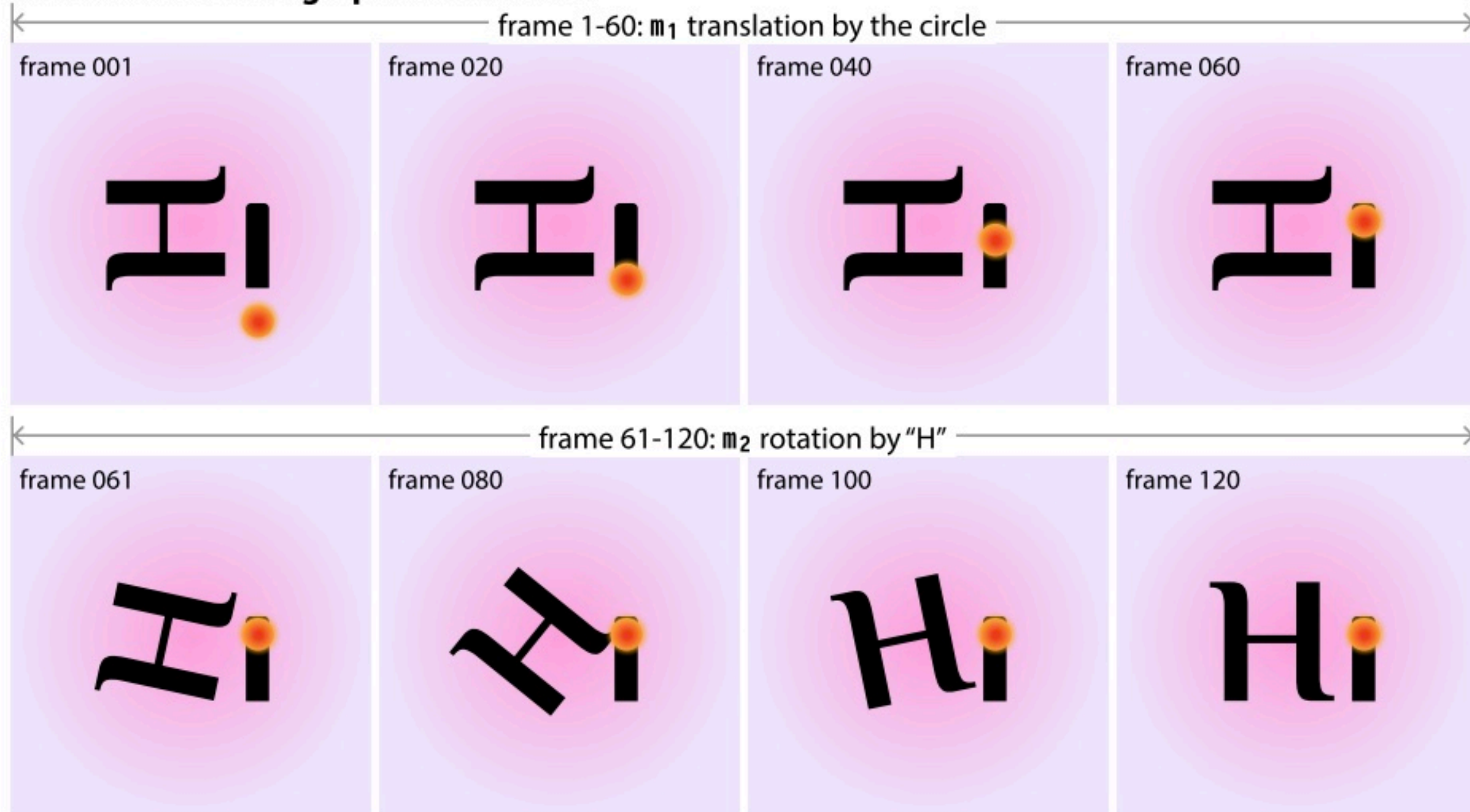
## MoVer program

```
o1 = 10.clr(o,"orange")^shp(o,"circle") # check for an orange circle  
o2 = 10.shp(o,"rectangle") # check for a rectangle  
o3 = 10.id(o,"H") # check for the id "H"  
# check for a translation by the circle to the top of the rectangle  
m1 = 1m.type(m,"trn")^agt(m,o1)^post(m,top(o1,o2))  
# check for a clockwise rotation of 90 degrees by "H"  
m2 = 1m.type(m,"rot")^agt(m,o3)^dir(m,"cw")^mag(m,90)  
# assert that the translation and the rotation overlap in time  
while(m1,m2)
```

## MoVer verification report

```
m1 = 1m.type(m,"trn")^agt(m,o1)^post(m,top(o1,o2)) false  
    top(o1,o2) false  
    post(m,top(o1,o2)) false  
m2 = 1m.type(m,"rot")^agt(m,o3)^dir(m,"cw")^mag(m,90) true  
while(m1,m2) false
```

## Generated motion graphics animation



- Follows an emerging pattern in AI-based program generation
- Given an editing instruction, generate:
  - A program that performs the edit... AND
  - A set of predicates that should be true if the edit was successfully performed (verifiers)
- If any of the predicates fail, have the program generator try again (given information about its prior failures)
- Questions:
  - What is the collection of verifiers?
  - Can the verifiers be "powerful enough" to provide useful checking?