Lecture 9:

Controlling Visual Generative AI Part II

Visual Computing Systems
Stanford CS348K, Spring 2026

Text conditioning

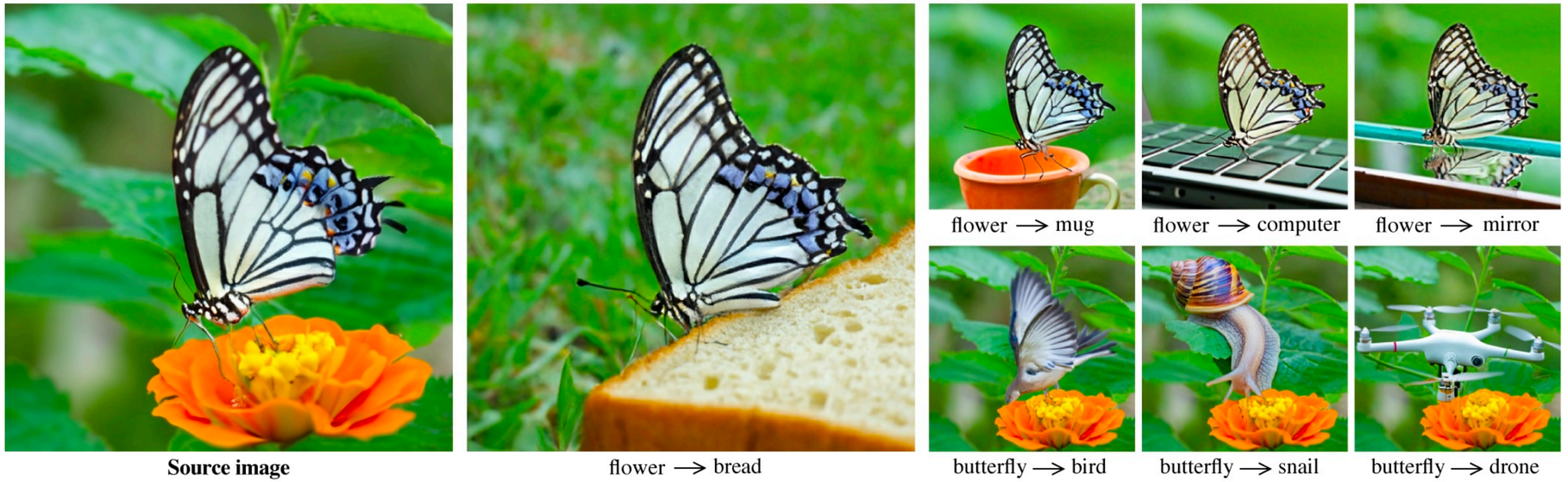
"A classroom of many attentive college students"

Adding controls by tweaking the generative process

Use change in text prompt to trigger change in image

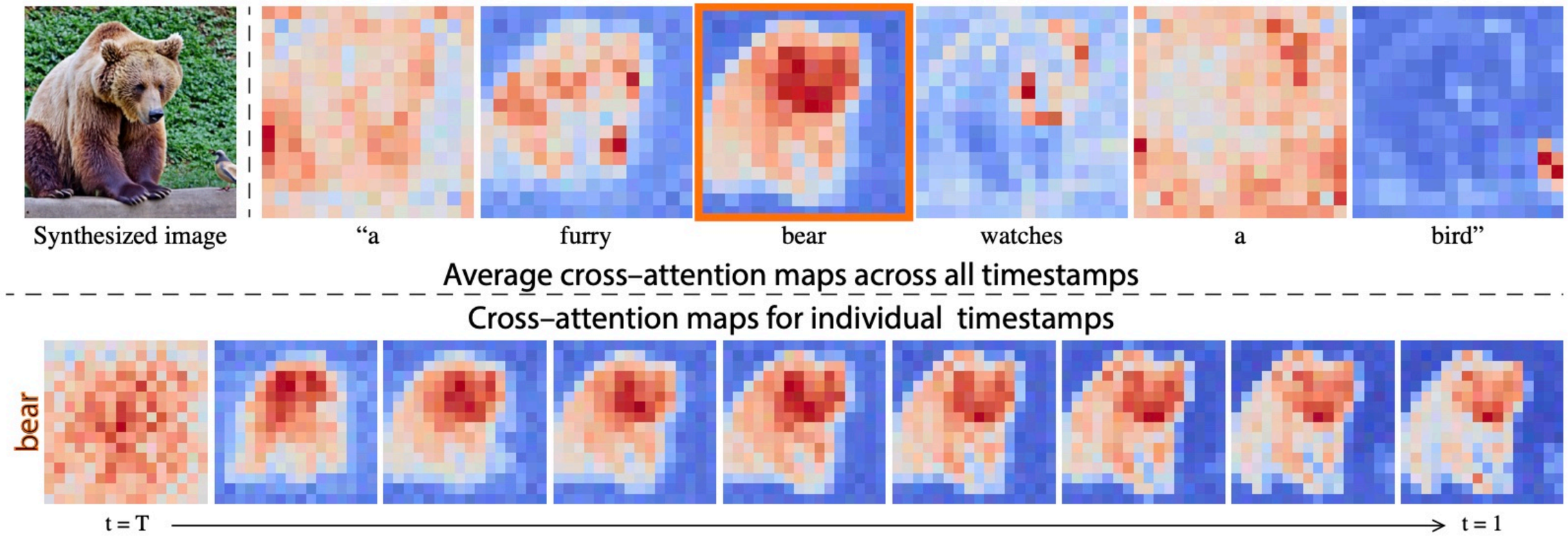
"A basket full of apples."

"A photo of a butterfly on a flower."



“Masks” come from learned attention

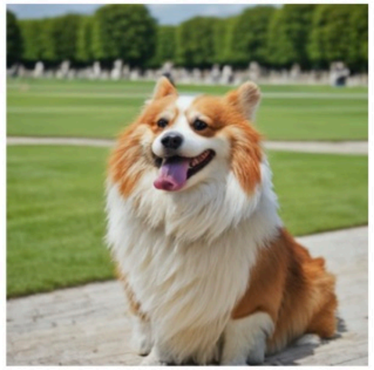
- Use attention masks from original generation process to constrain what pixels can change after prompt is edited



Specialization to a specific concept

Input images

A [V] dog in the Versailles hall of mirrors



A [V] dog in the gardens of Versailles

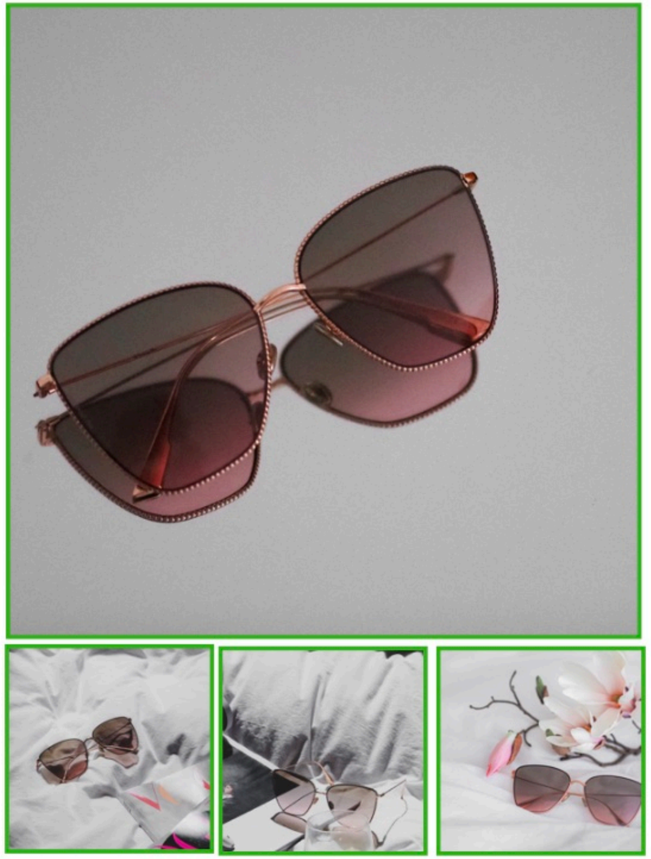
A [V] dog with Eiffel Tower in the background

A [V] dog in Coachella

A [V] dog in mountain Fuji

Specialization to a concept

Input images

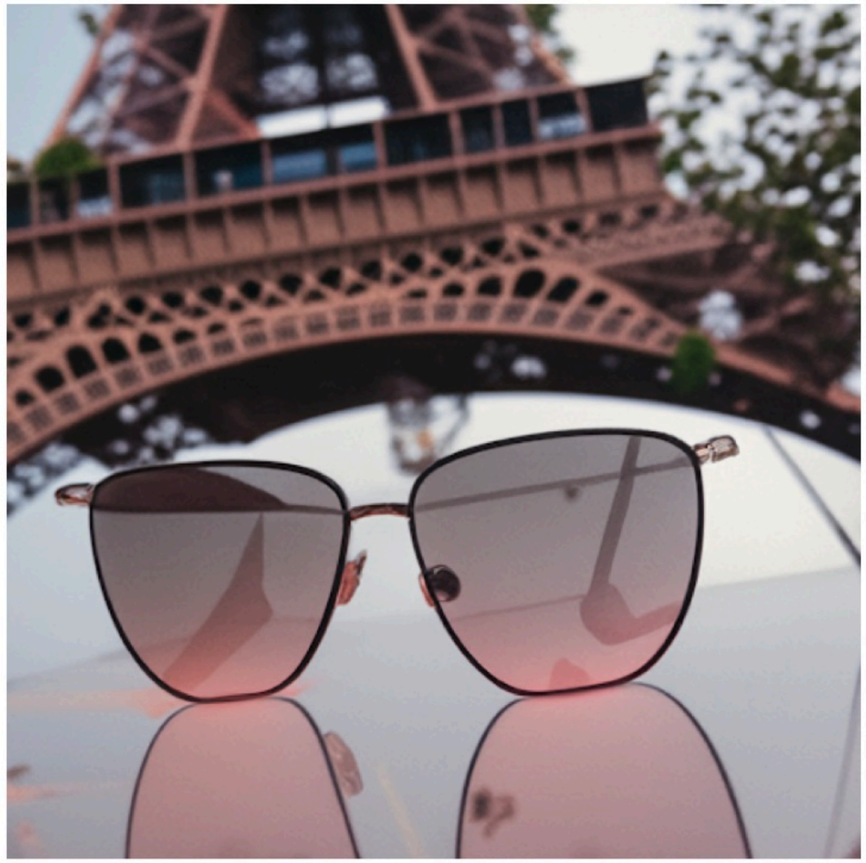


A [V] sunglasses in the jungle

A [V] sunglasses worn by a bear

A [V] sunglasses at Mt. Fuji

A [V] sunglasses on top of snow

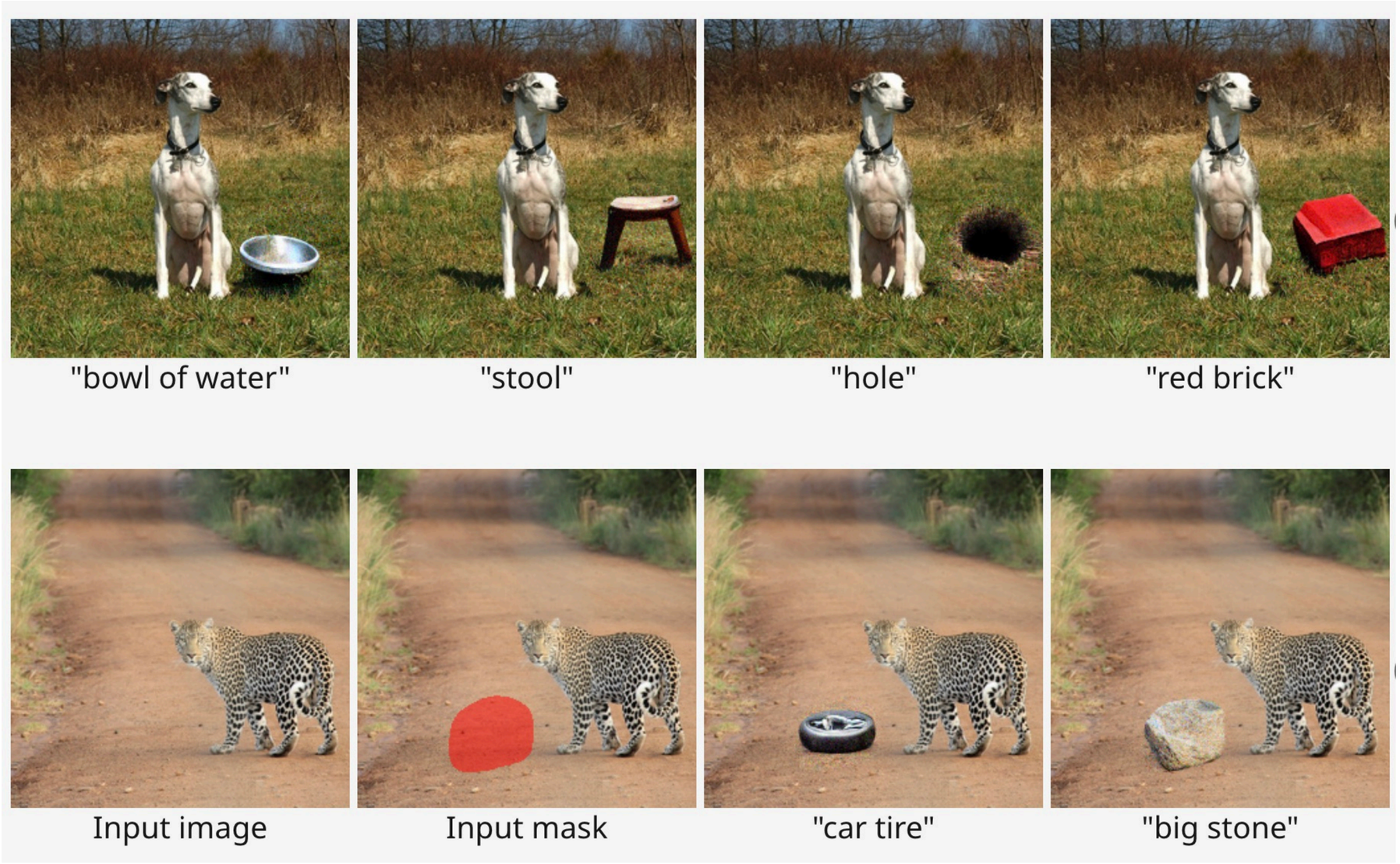


A [V] sunglasses with Eiffel Tower in the background

Inpainting (apply [new] prompt to a region)

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

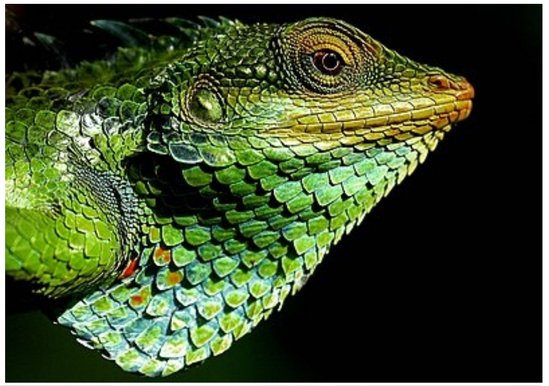
Image outside of region remains almost the same.

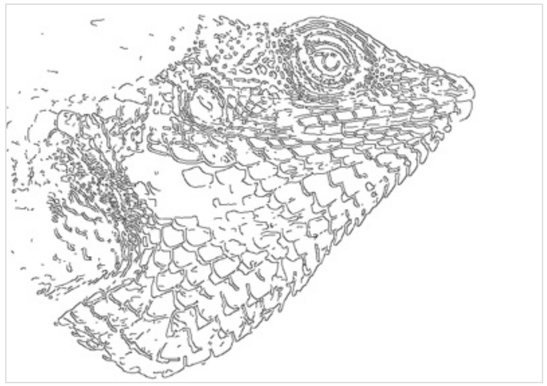


Clever ways to generate input/output examples

Other forms of conditioning

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

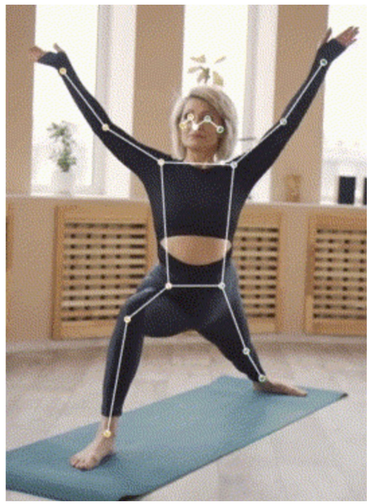




Edge detection

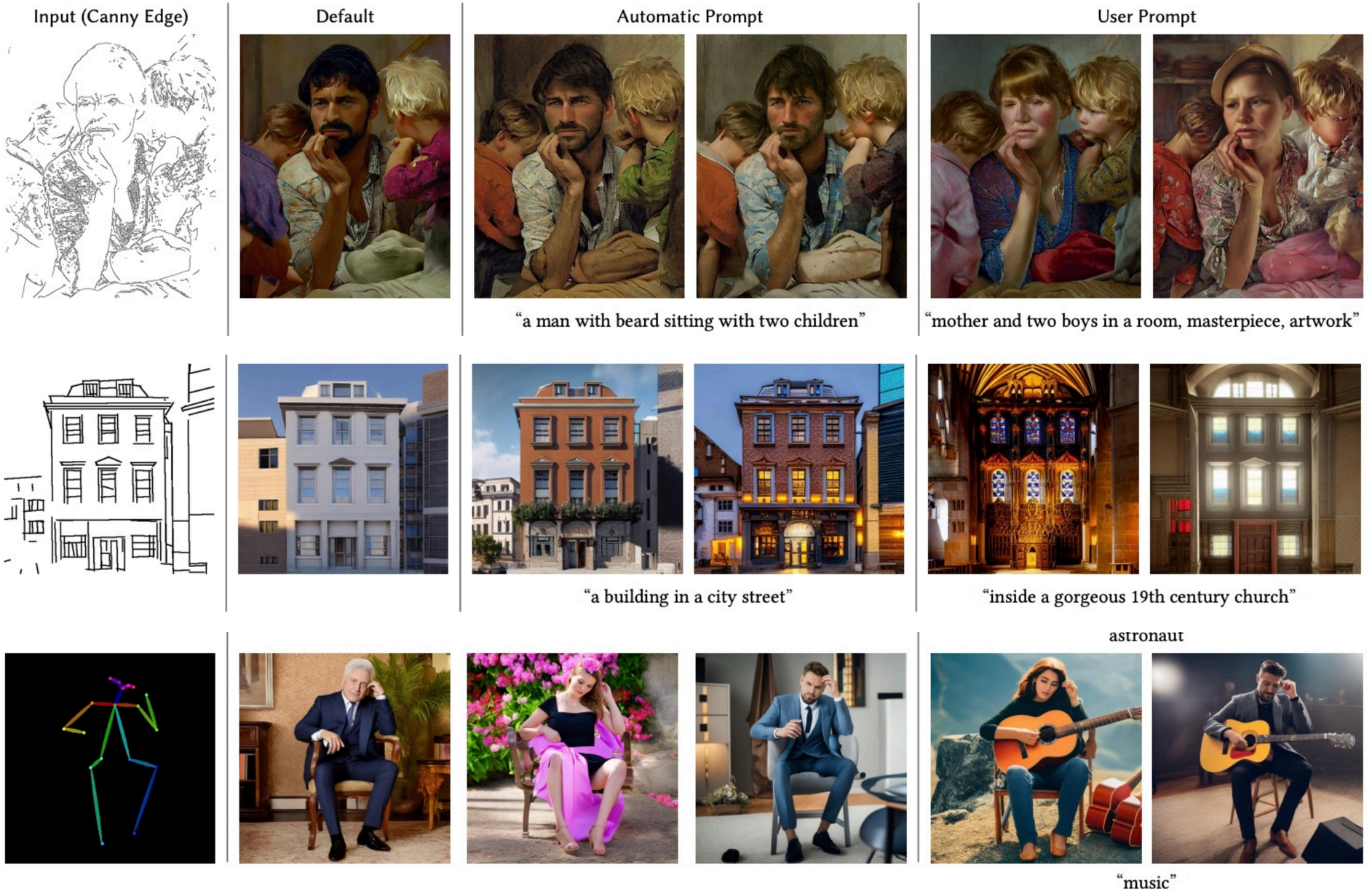
Segmentation

Depth



Another example:
Pose Estimation

Alternative forms of control



Reducing the cost of training

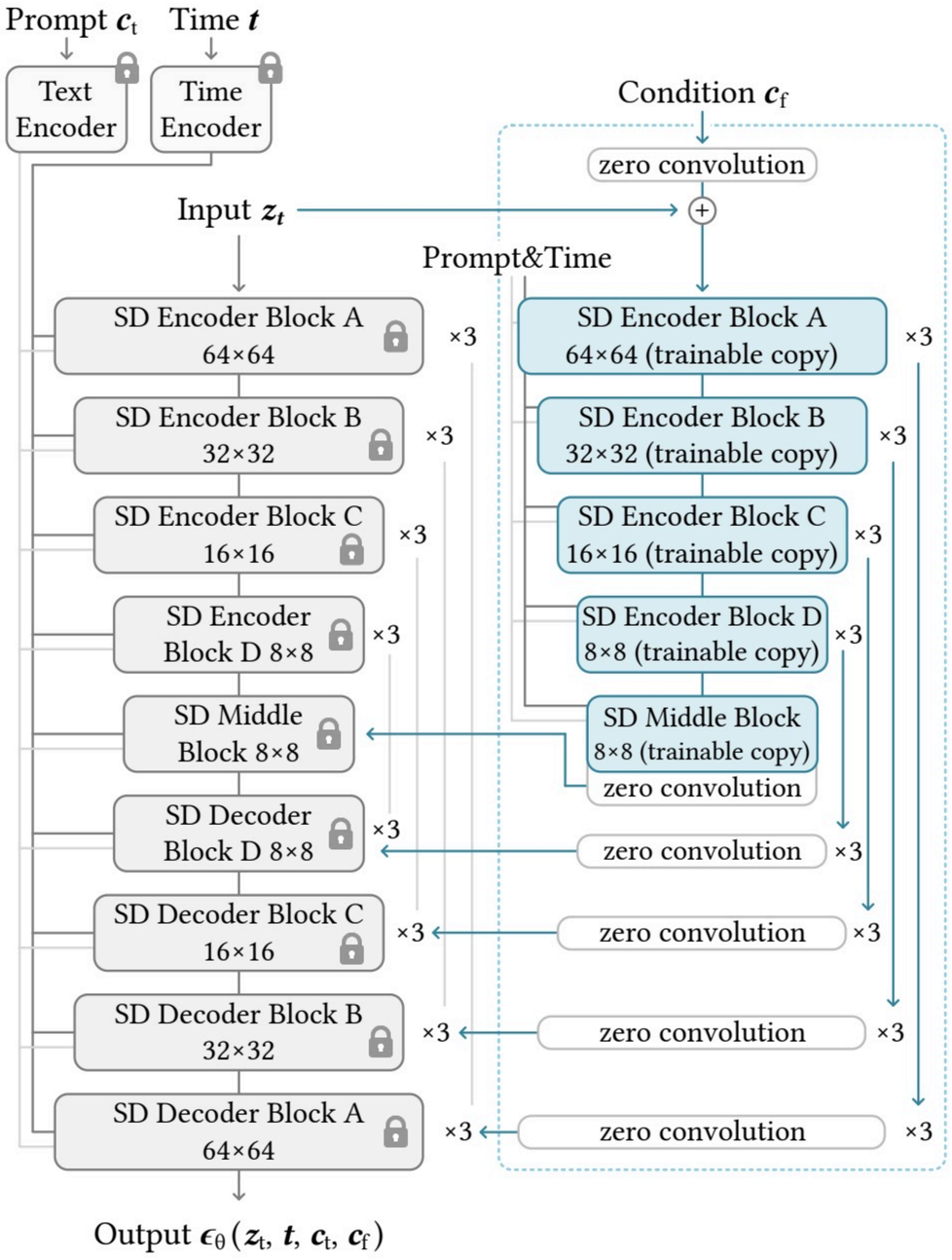
- Example open source text/image training dataset:
  - LAION-5B (5.5B image/test training pairs)

- Significant compute cost to train a model

ControlNet

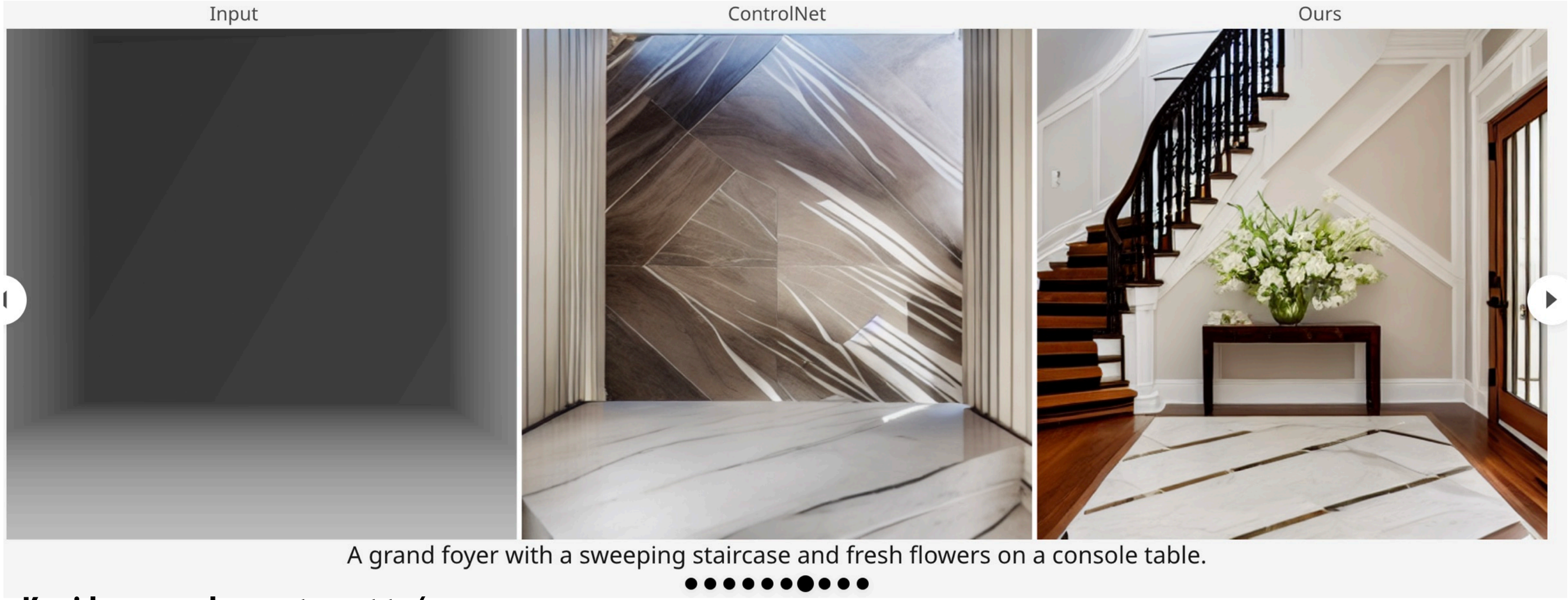
- **Key idea: duplicate diffusion network**
  - **Original copy retains original weights**
  - **ControlNet copy learns how to modify activations to respect new control signals**

- **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)**

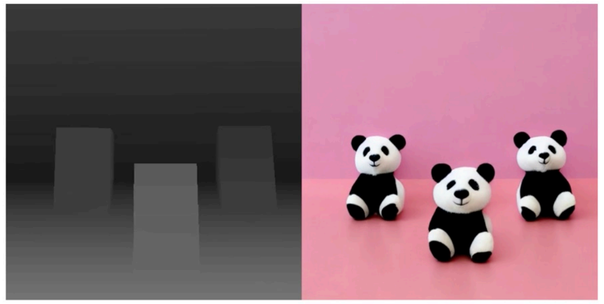


(a) Stable Diffusion (b) ControlNet

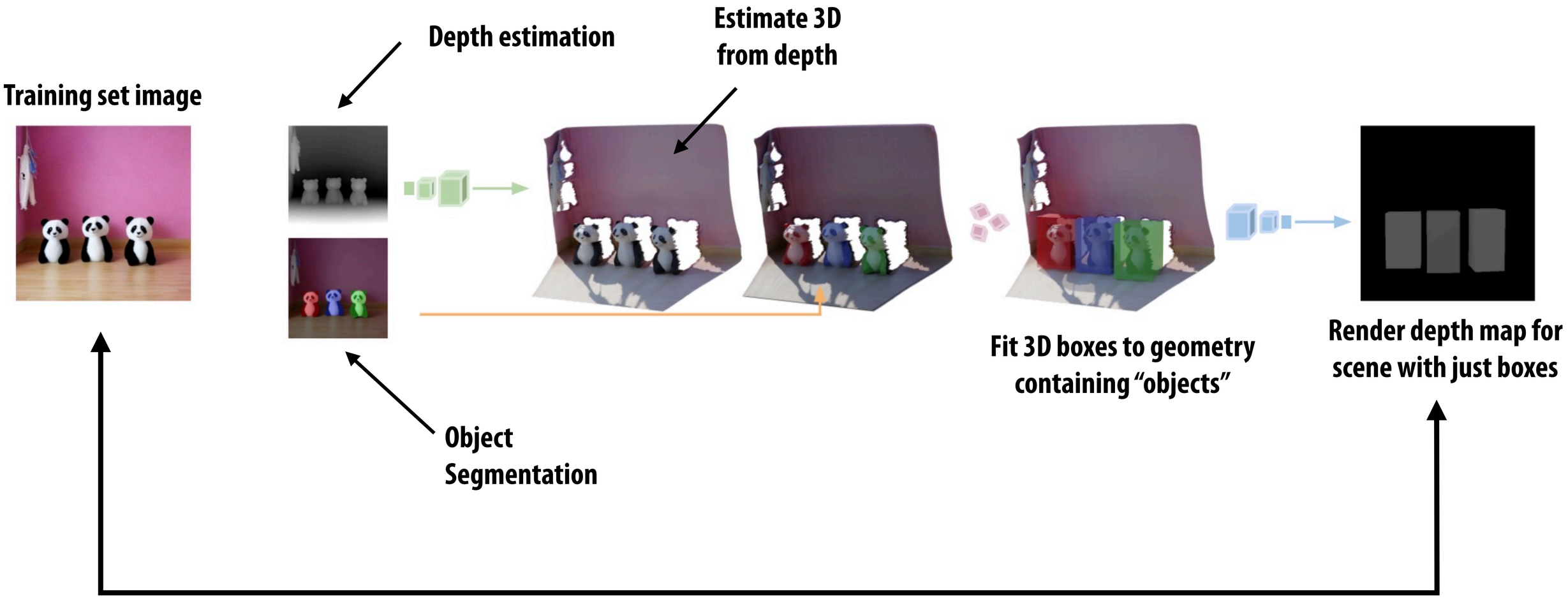
Loose control



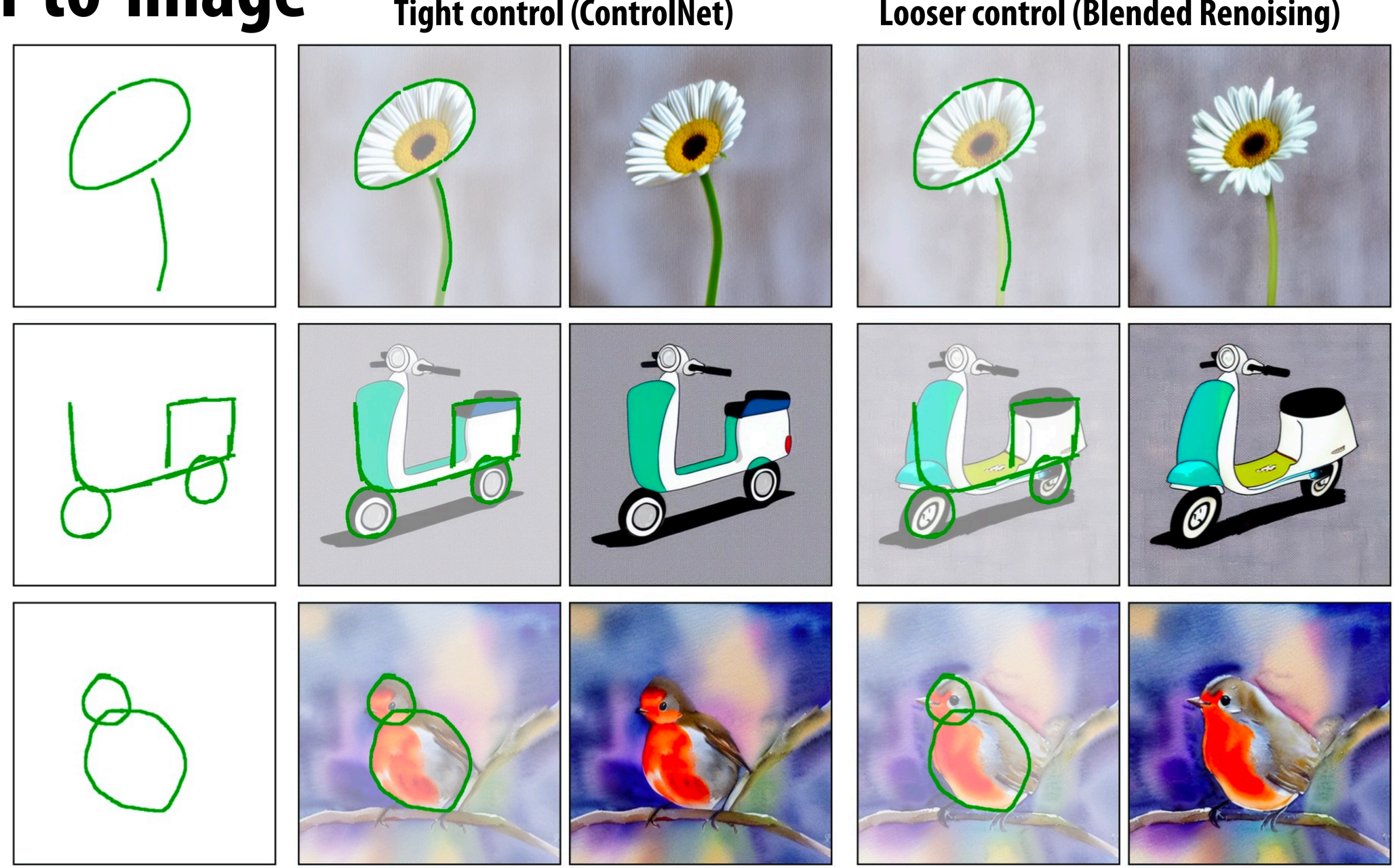
Key idea: user does not want to (or may not have capability to) specific visual controls precisely. Just have the user "block out" the basic shape of the scene.



Generating training data for loose control

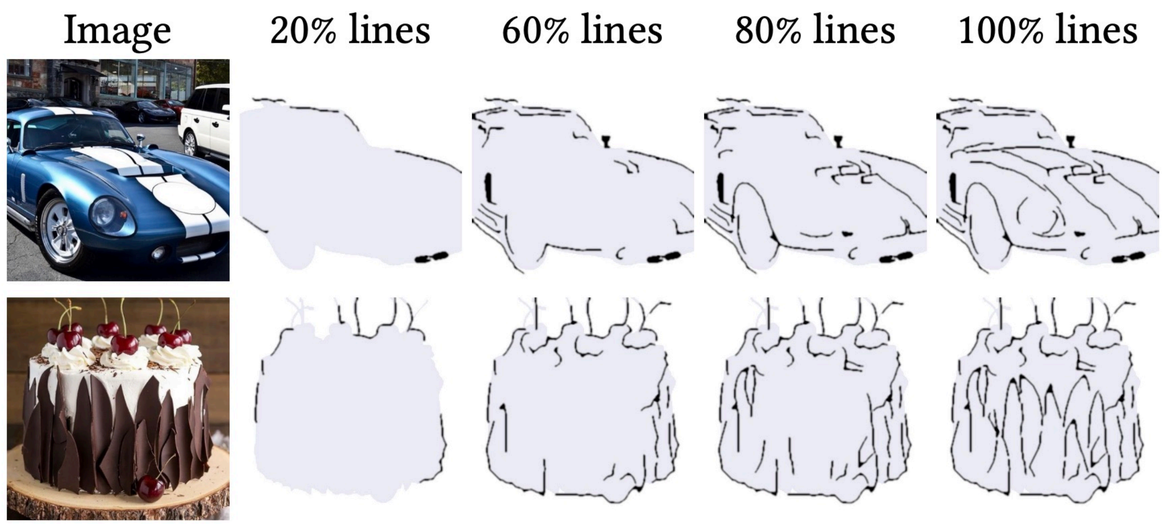


Sketch-to-image



Partial sketch to image training data generation

Creating "partial-sketch" training data



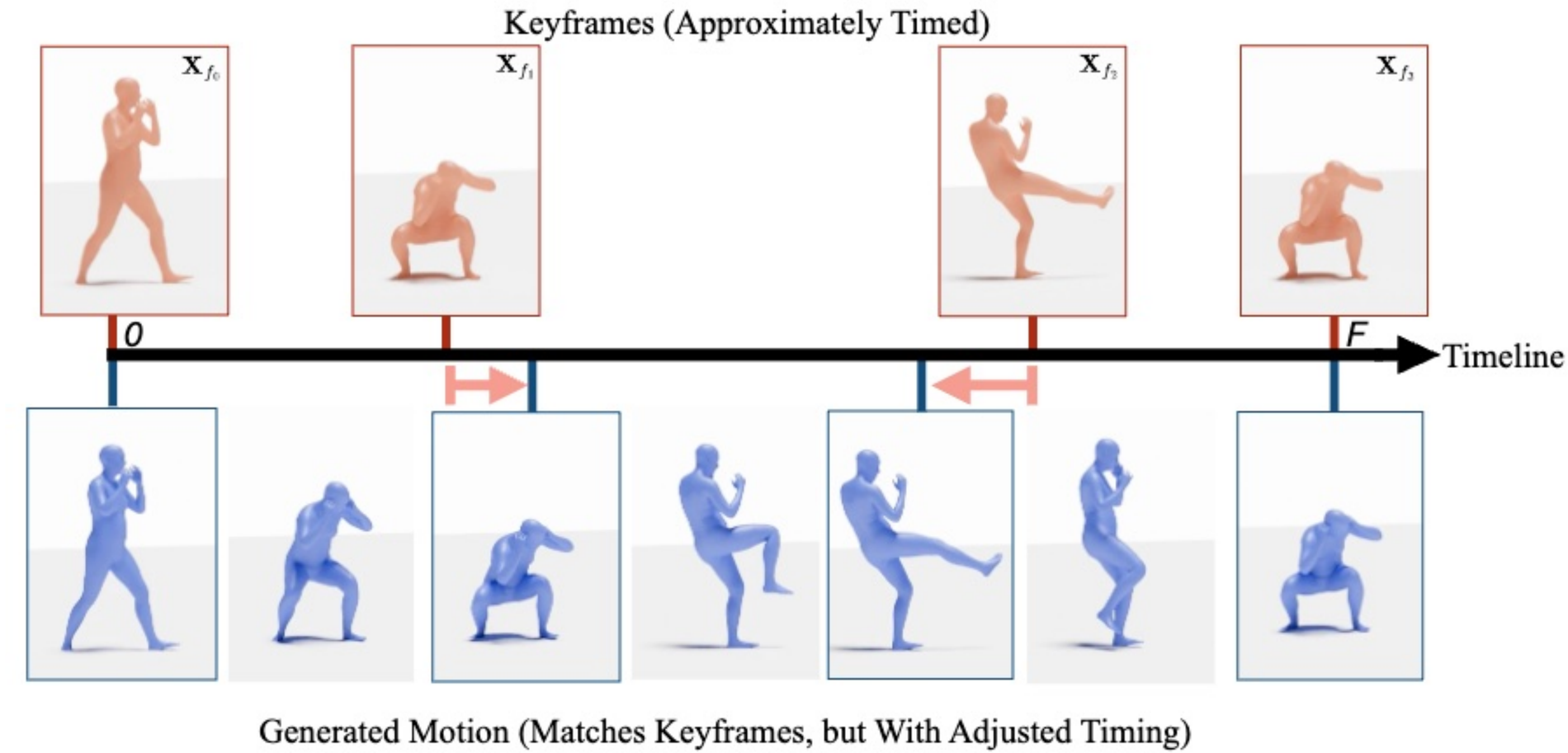
Find edge lines

Identify contour lines (via segmentation)

Progressively remove edge lines that are farthest from contours.

Imprecise keyframe conditioned generation

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



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"

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

Identifying the controls that are most useful to an editing task is the first step toward choosing appropriate representations for generation

- 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

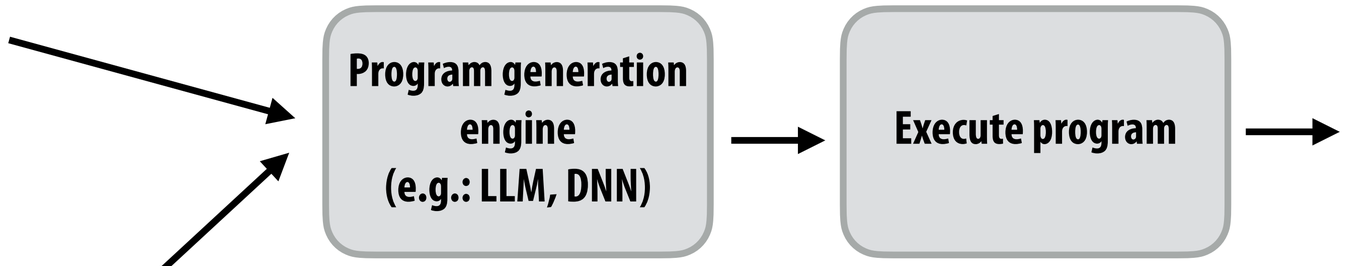
Reducing visual/spatial content generation tasks to LLM-based program generation.

Execute the program to generate the artifact (often very low cost)

High-level specification

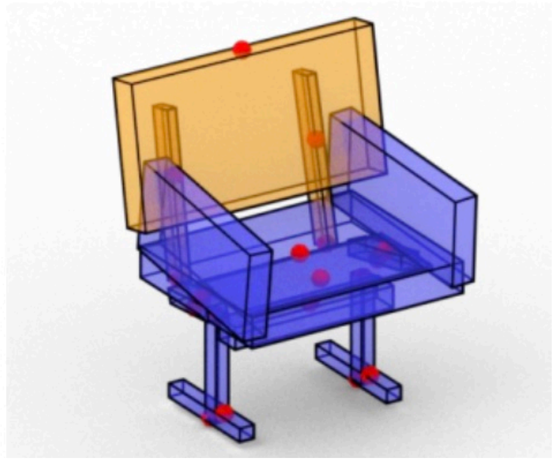
"Chair with a flat back"

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




```
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)
```

Generated output

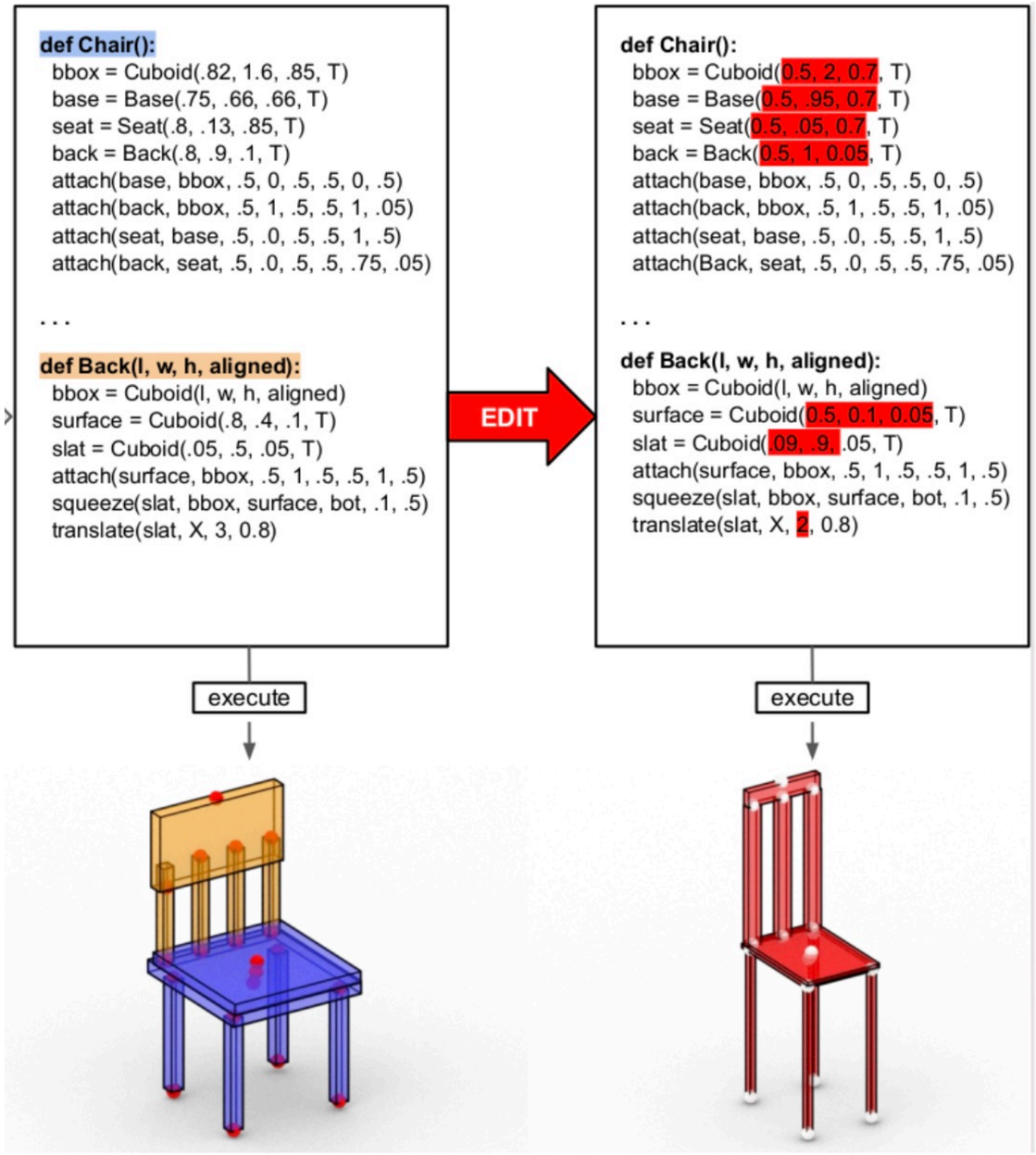


One benefit: a program is a human interpretable representation

Reducing generation tasks to the act of writing programs

Editable/tweakable: we read and edit programs all the time!

Low-cost to rerun edited program to generate edited object.



Another example

Python programs as "plans" to carry out tasks in 3D environments

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():
```

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

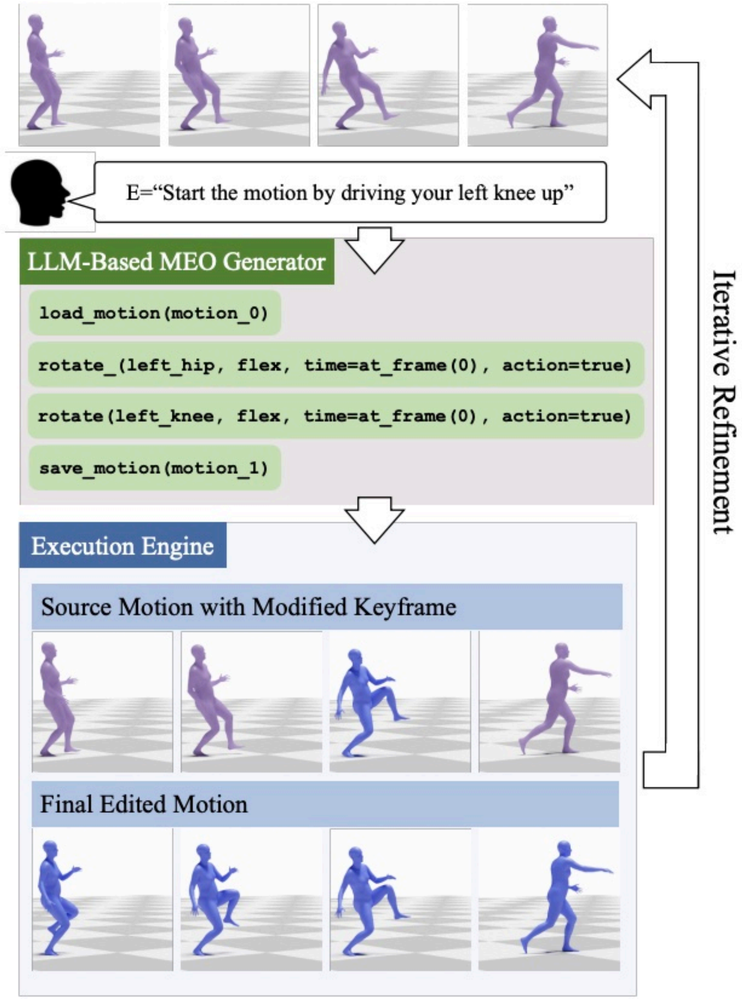
Assertions: State Feedback

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"
```

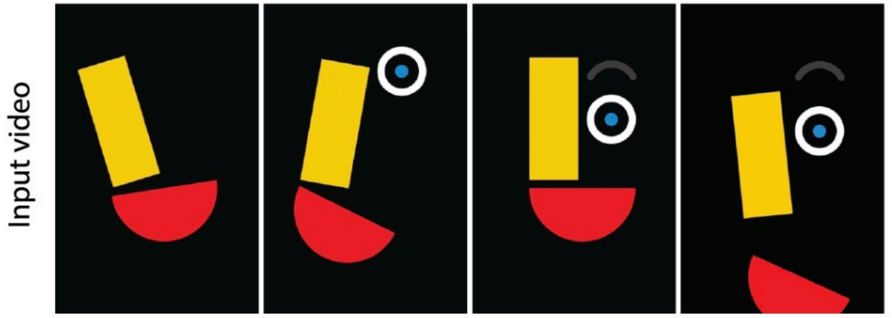
Object Localizer

Simple VQA Method

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

Lifting video to program, then edit, then render video

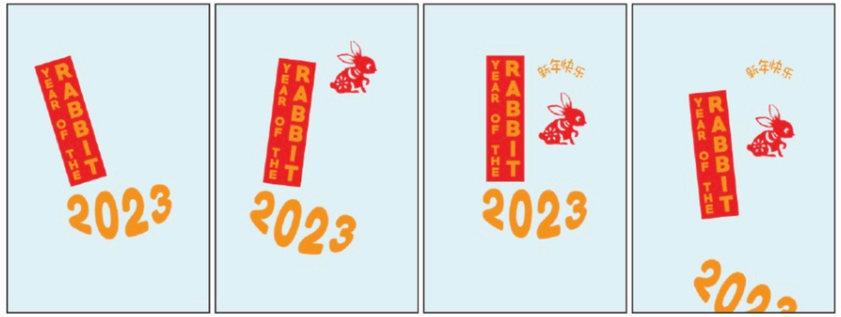
1. Video —> SVG keyframe animation (computer vision)
2. LLM edits the SVH file (e.g., change yellow rectangle to texture map)
3. Re-render animation



Motion vectorization

```
<svg w=200, h=200, dur="5s">
  <g id="0">
    <img href="red_ball.png" />,
    <scale val="1.0 1.0; 1.05 1.0; ..." />,
    <translate val="0.1 0.5; 0.15 0.5..." />,
    <rotate val="1.0; 1.05; ..." />,
    <z-index val="0; 0; ..." />,
    ...
  </g>
  <g id="1">...</g>
  ...
</svg>
```

SVG



SVG

SVG

Program transformer

```
NewYearsProgTransformer(P, args, [frmA, frmB]):
  // Set background color.
  setAppearance("bg", args.background)

  // OBJECT SELECTOR: Query for the red semicircle.
  selObjs = objSelector(P, propQuery, "color", "red", [frmA, frmB])
  // OBJECT TRANSFORMER: Change the appearance to the year.
  changeAppearanceObjTransformer(selObjs, args.year, [frmA, frmB])
  // Repeat obj selection and obj transformation for banner and animal.
  selObjs = objSelector(P, propQuery, "color", "yellow", [frmA, frmB])
  changeAppearanceObjTransformer(selObjs, args.banner, [frmA, frmB])
  selObjs = objSelector(P, propQuery, "color", "white", [frmA, frmB])
  changeAppearanceObjTransformer(selObjs, args.zodiac, [frmA, frmB])

  // OBJECT SELECTOR: Query for the gray curve.
  selObjs = objSelector(P, propQuery, "color", "gray", [frmA, frmB])
  changeAppearanceObjTransformer(selObjs, args.characters, [frmA, frmB])
  // OBJECT TRANSFORMER: Apply an oscillating scale.
  function pulse(t, [sx, sy]):
    return [sx + 0.5 * np.sin(t / 10), sy + 0.5 * np.sin(t / 10)]
  motionTexObjTransformer(selObjs, pulse, args.pulseArgs, [frmA,frmB])

  // OBJECT SELECTOR: Query for the blue circle.
  selObjs = objSelector(P, propQuery, "color", "blue", [frmA, frmB])
  // OBJECT TRANSFORMER: Remove the object.
  removeObj(selObjs, [frmA, frmB])
```

Discussion: MoVer paper

What is the DSL for the following edit?

- Given this photo, make the cat's tail curve back into the frame.

- Let's think about the pros and cons of this program-driven generation/editing approach:

  What are the benefits?

  What are limitations of this approach?

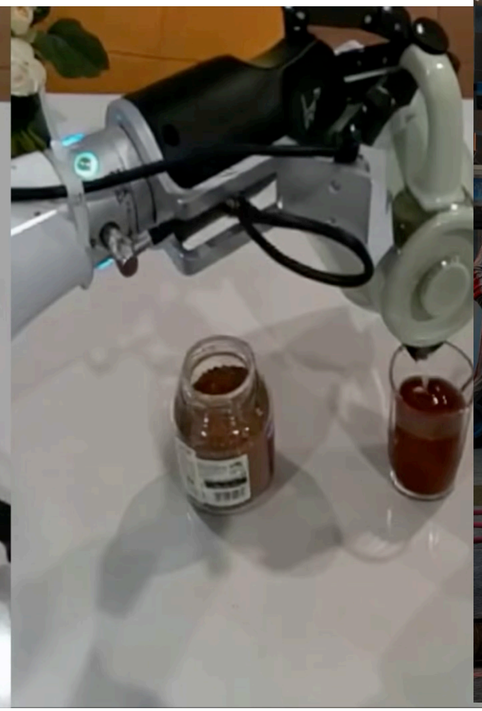
Challenges

- 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)?
  - Verification challenge: Can we predict when a program generator will fail?

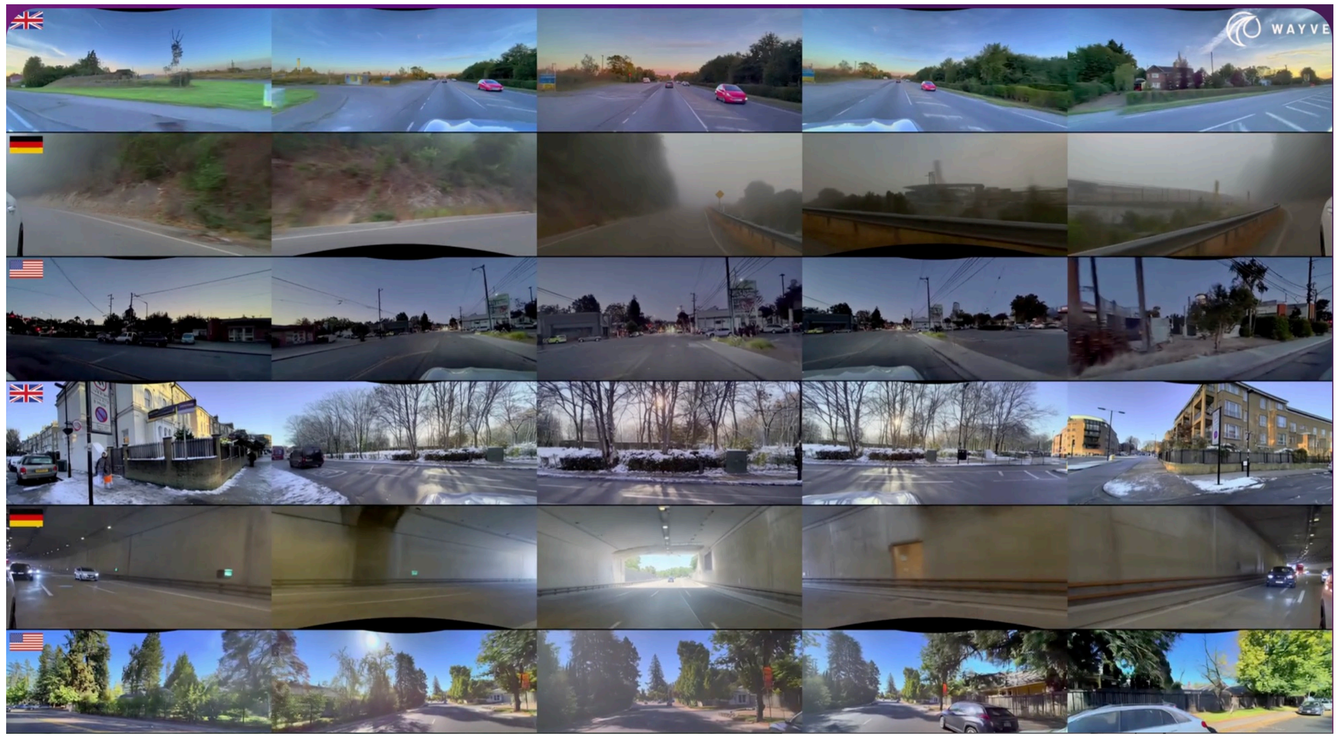
Setup for next class…

- Goal: controllable interactive worlds
- One option: make a computer game using a traditional game engine (a simulation)



Setup for next class…

- Goal: controllable interactive worlds
- One option: make a computer game using a traditional game engine (a simulation)
- Another option: follow trends in modern generative AI (text to “game”)



Middle grounds?

- Use traditional game for modeling interactions / dynamics, use AI for “uplift” to photoreal
- An early example from 2010: cg2real

Middle grounds?

- Use traditional game for modeling interactions / dynamics, use AI for “uplift” to photoreal
- A more recent example: GTA to “photoreal”

Middle grounds?

- Use traditional game for modeling interactions / dynamics, use AI for “uplift” to photoreal
- A very recent example: NVIDIA DLSS5

What should the game engine "provide" to the AI to make enhancement better quality / lower cost / controllable?