

Shared Representation

CS 347
Jiaju Ma

Announcements

Quiz after lecture today — pencils!

Previously

AI vs. IA

Direct manipulation vs. Agents

Mixed-initiative interaction

Human and machine should collaborate, but *over what substrate?*

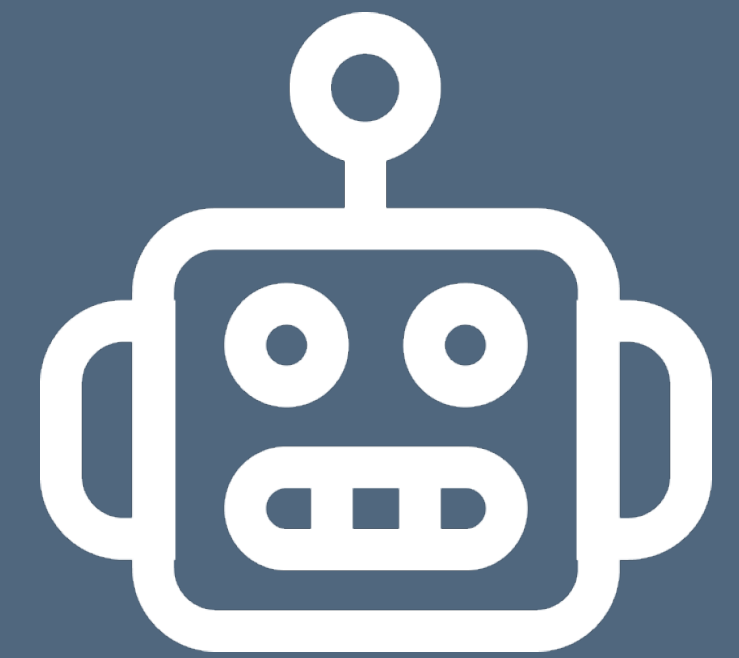
Agency plus Automation

[Heer 2019]

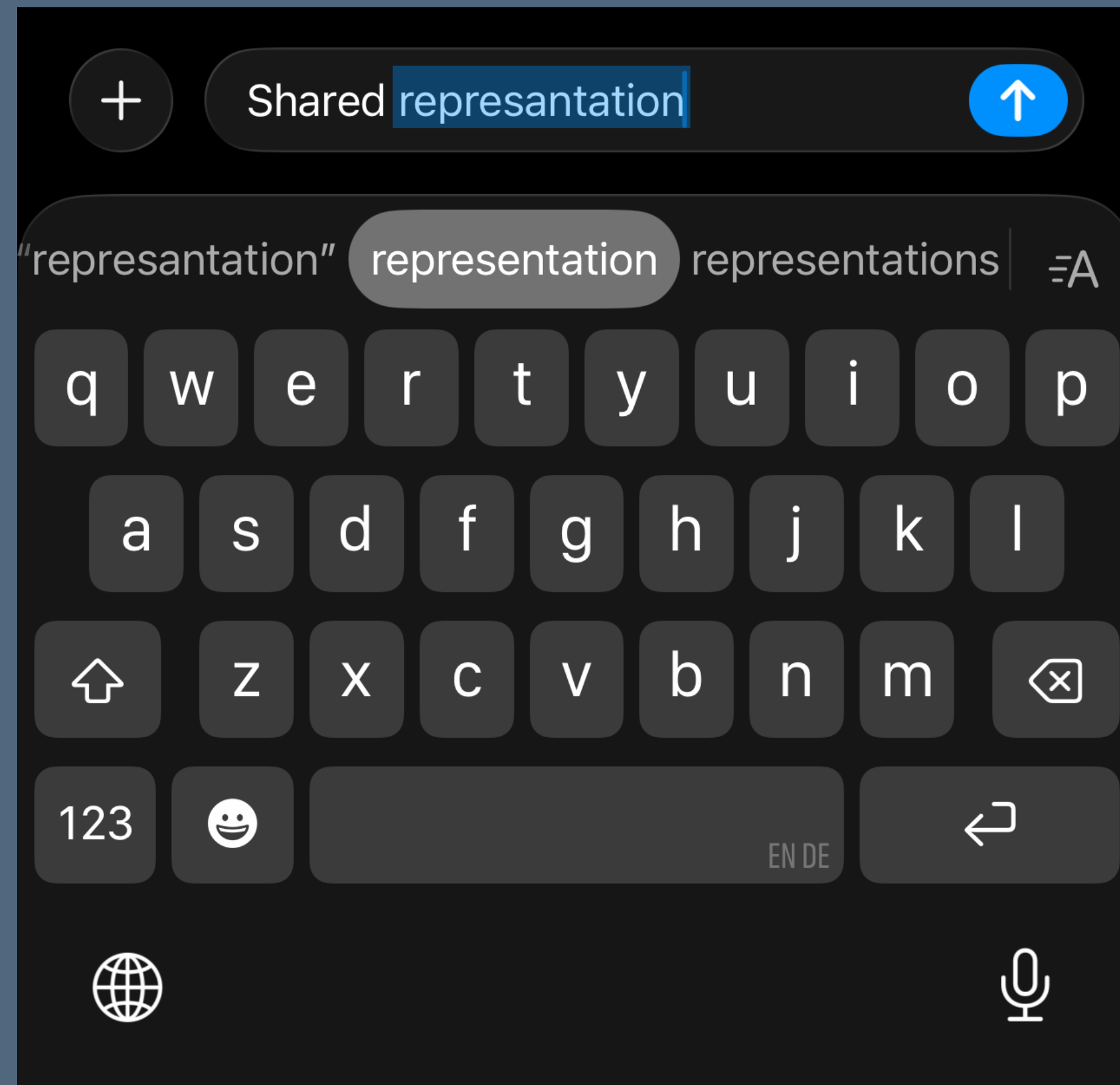


[Shared representation]

A medium that both human and machine can read, write, and modify



Autocorrect



Autocorrect

User intention — The user wants to spell a word correctly

User input — The user has typed a word that might contain typos

Machine output — The machine suggests the correct spelling

Shared representation — The word typed by the user

Designing shared representation

1. The shared representation can directly be *in the same format* as the user input and machine output

ShadowDraw

[Lee 2011]

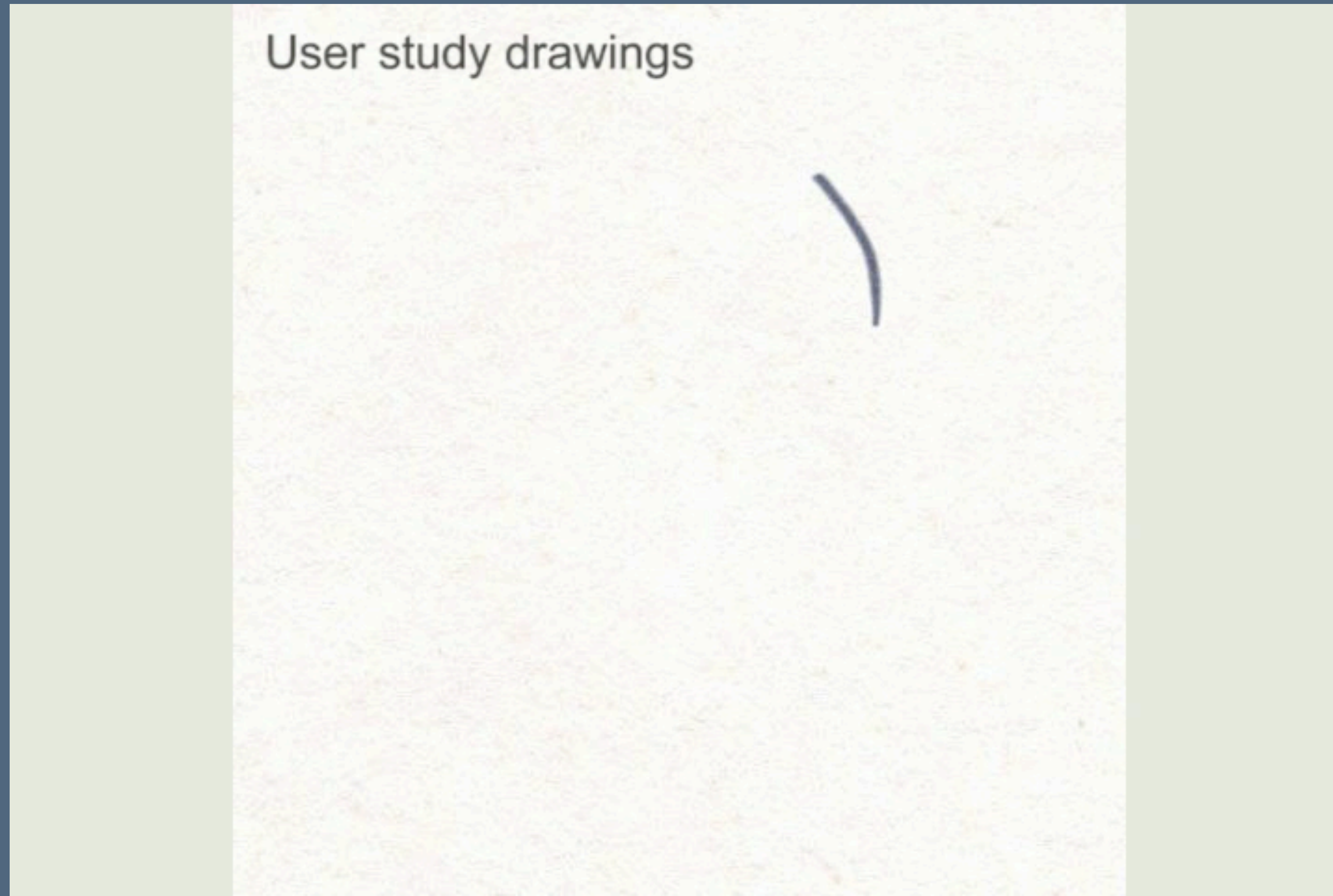
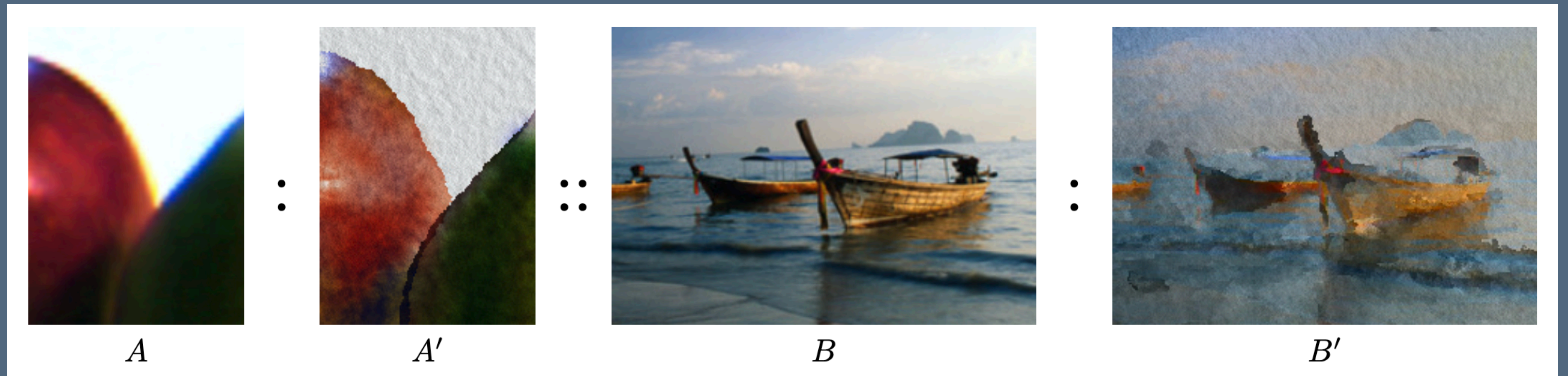


Image Analogies

[Hertzmann 2001]



User input

Machine output

Designing shared representation

I. The shared representation can directly be *in the same format* as the user input and machine output

What if the input and the output are *not* in the same format?

Data Wrangler

[Heer 2019]

The screenshot displays the Data Wrangler interface. On the left is a sidebar with a menu (Split, Cut, Extract, Edit, Fill) and a 'Transform Suggestions' panel. The suggestions include: 'Extract from Year between positions 18, 25', 'Extract from Year on 'Alabama'', 'Extract from Year after 'in'', 'Cut from Year between positions 18, 25', 'Split data repeatedly on newline into rows', 'Split data repeatedly on 'tab'', 'Delete empty rows', 'Promote row 2 to header', and 'Delete rows where Year contains 'Year''. Below the suggestions is a 'Transform Script' section with an 'Export' button.

The main data table has a menu (Translate, Drop, Merge, Wrap, Delete, Promote, Fold, Pivot, Transpose) and a table with the following data:

#	Year	Abc	extract	#	Population	#	Property_crime_rate	#	Burgla
1	Reported crime in Alabama	Alabama							
2	2004			4525375		4029.3		987	
3	2005			4548327		3900		955.8	
4	2006			4599030		3937		968.9	
5	2007			4627851		3974.9		980.2	
6	2008			4661900		4081.9		1080.7	
7	Reported crime in Alaska	Alaska							
8	2004			657755		3370.9		573.6	
9	2005			663253		3615		622.8	
10	2006			670053		3582		615.2	
11	2007			683478		3373.9		538.9	
12	2008			686293		2928.3		470.9	
13	Reported crime in Arizona	Arizona							
14	2004			5739879		5073.3		991	
15	2005			5953007		4827		946.2	
16	2006			6166318		4741.6		953	
17	2007			6338755		4502.6		935.4	
18	2008			6500180		4087.3		894.2	
19	Reported crime in Arkansas	Arkansas							
20	2004			2750000		4033.1		1096.4	

ROWS: 306

User input

Data Wrangler

[Heer 2019]

The screenshot displays the Data Wrangler interface. On the left, a sidebar contains a 'Transform Suggestions' panel with the following options:

- Extract from Year between positions 18, 25
- Extract from Year on 'Alabama'
- Extract from Year after 'in ' (highlighted in green)
- Cut from Year between positions 18, 25

Below the suggestions is a 'Transform Script' panel with an 'Export' button and the following script:

```
▶ Split data repeatedly on newline into rows
▶ Split data repeatedly on 'tab'
▶ Delete empty rows
▶ Promote row 2 to header
▶ Delete rows where Year contains 'Year'
```

The main area shows a data table with the following columns: #, Year, Abc, extract, #, Population, #, Property_crime_rate, #, Burgla. The table contains data for three states: Alabama, Alaska, and Arkansas, with rows for each year from 2004 to 2008. The status bar at the bottom indicates 'ROWS: 306'.

#	Year	Abc	extract	#	Population	#	Property_crime_rate	#	Burgla
1	Reported crime in Alabama	Alabama							
2	2004			4525375		4029.3		987	
3	2005			4548327		3900		955.8	
4	2006			4599030		3937		968.9	
5	2007			4627851		3974.9		980.2	
6	2008			4661900		4081.9		1080.7	
7	Reported crime in Alaska	Alaska							
8	2004			657755		3370.9		573.6	
9	2005			663253		3615		622.8	
10	2006			670053		3582		615.2	
11	2007			683478		3373.9		538.9	
12	2008			686293		2928.3		470.9	
13	Reported crime in Arizona	Arizona							
14	2004			5739879		5073.3		991	
15	2005			5953007		4827		946.2	
16	2006			6166318		4741.6		953	
17	2007			6338755		4502.6		935.4	
18	2008			6500180		4087.3		894.2	
19	Reported crime in Arkansas	Arkansas							
20	2004			2750000		4033.1		1096.4	

Machine output

Data Wrangler

[Heer 2019]

User intention — The user wants to transform data

User input — Data frame + selection

Machine output — The machine suggests potential transformations

Shared representation — A DSL of data transformations

Designing shared representation

1. The shared representation can directly be *in the same format* as the user input and the machine output
2. The shared representation can be an intermediate layer between the user input and the machine output when 1. is not feasible

“three dogs in the image”



Is this what I asked for?

“three dogs in the image”



VPEval

[Cho 2023]

“three dogs in the image”



Visual Program

```
countEval(img, "dog", "==3")
```



VPEval

[Cho 2023]

“three dogs in the image”

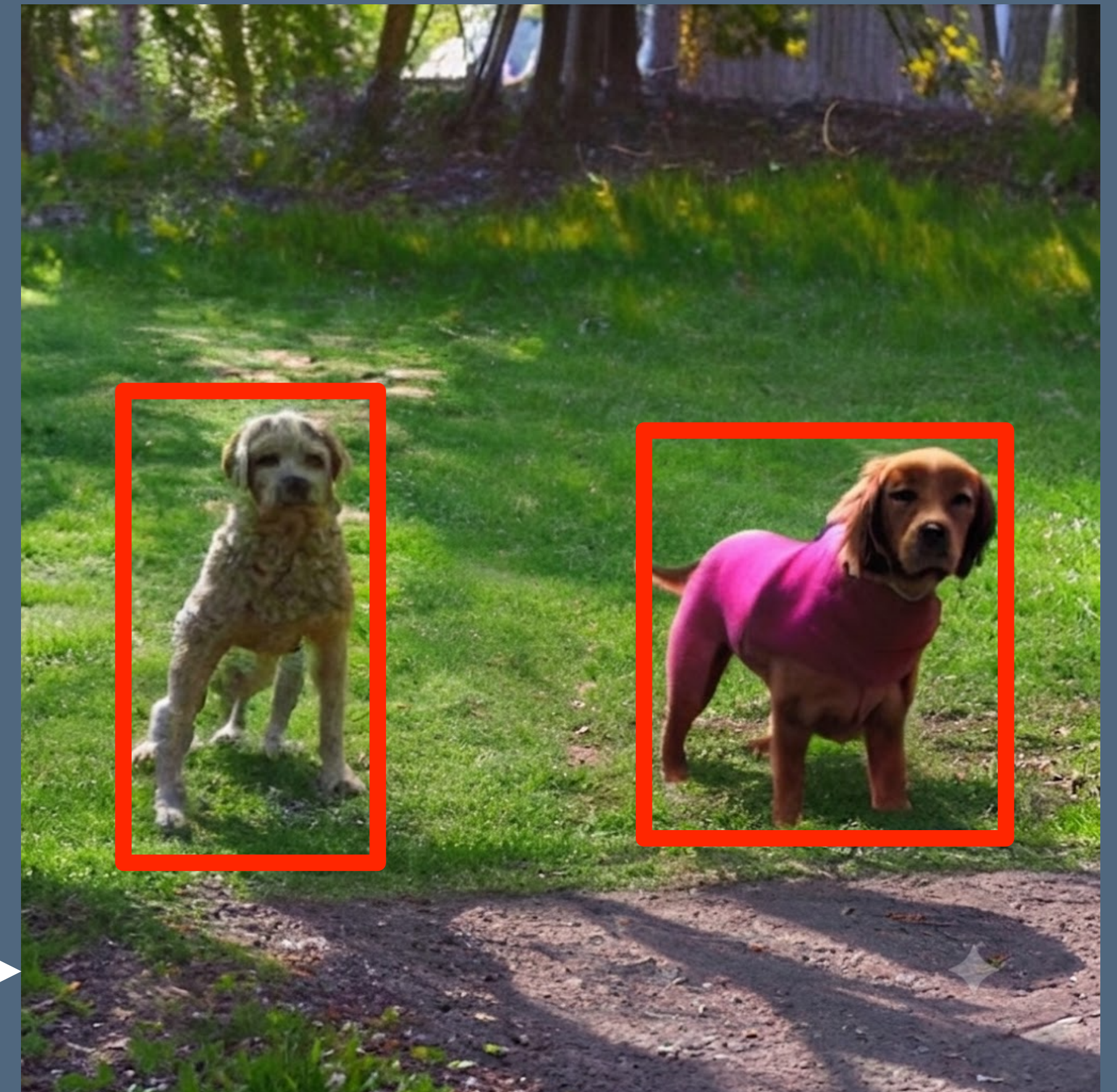


Visual Program

```
countEval(img, “dog”, “==3”)
```



```
>>> countEval(img, “dog”, “==3”)  
False
```



Program as shared representation

Encodes action space — Common set of actions a user can perform and the machine can search over

User understandable and editable — User can look at the program to reason about the output

Machine executable — Unambiguous, formal, directly executable

ViperGPT

[Menon 2023]

Query: how many muffins can each kid have for it to be fair?

Input Image:



ViperGPT

[Menon 2023]

Query: how many muffins can each kid have for it to be fair?

Code Generation



Input Image:



```
1 def execute_command(image)->str:
2     image_patch = ImagePatch(image)
3     muffin_patches = image_patch.find("muffin")
4     n_muffins = len(muffin_patches)
5     kid_patches = image_patch.find("kid")
6     n_kids = len(kid_patches)
7     result = str(n_muffins // n_kids )
8     return result
```

ViperGPT

[Menon 2023]

Query: how many muffins can each kid have for it to be fair?

Code Generation



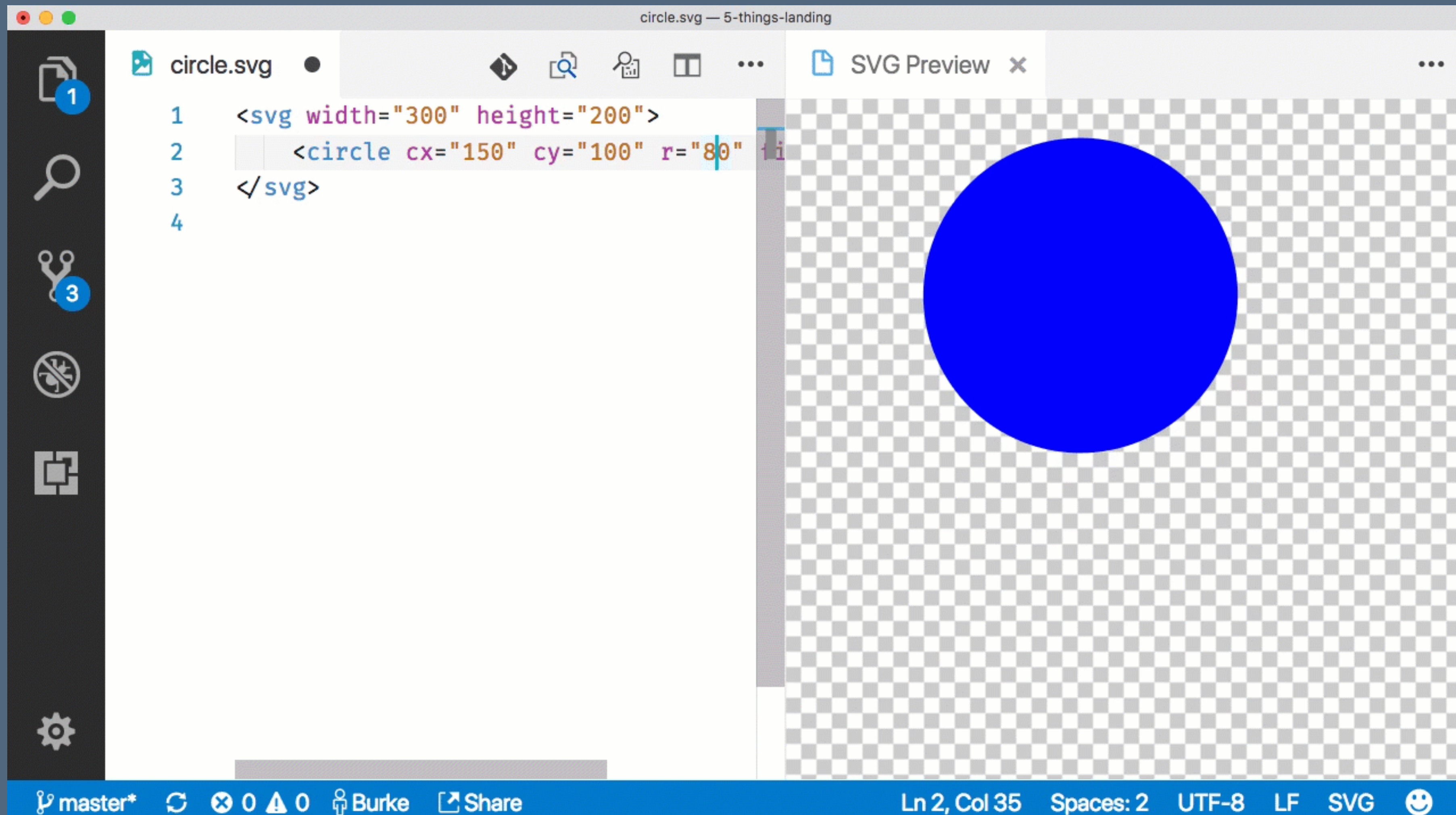
Input Image:



```
1 def execute_command(image)->str:
2     image_patch = ImagePatch(image)
3     muffin_patches = image_patch.find("muffin")
4     n_muffins = len(muffin_patches)
5     kid_patches = image_patch.find("kid")
6     n_kids = len(kid_patches)
7     result = str(n_muffins // n_kids )
8     return result
```

Image as program

[Holland 2018, Medium]



Animation as program

[Gannon 2017, CodePen]

The screenshot shows a CodePen editor for a project titled "SVG Day and Night" by Chris Gannon. The editor is divided into three panels: HTML, CSS, and JS. The HTML panel contains an SVG element with a linear gradient and various attributes. The CSS panel includes an @import for the Raleway font and styles for the body and HTML elements. The JS panel contains JavaScript code that uses document.querySelector to find elements and animate them. Below the code panels is a preview of the resulting animation, which shows a stylized city scene with a yellow sun and a grey moon, and a street with buildings and a street lamp. The preview is labeled "MADE WITH BY CHRIS GANNON".

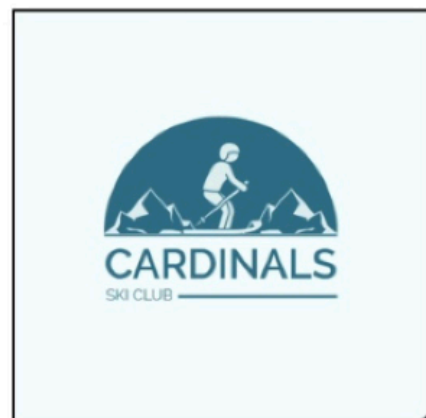
```
HTML
1 <div class="container">
2 <svg class="nightDaySVG" version="1.1"
3   xmlns="http://www.w3.org/2000/svg"
4   xmlns:xlink="http://www.w3.org/1999/xlink"
5   xmlns:a="http://ns.adobe.com/AdobeSVGViewerExtensions/3.0/"
6   x="0px" y="0px" width="400px" height="600px"
7   viewBox="0 0 400 600" enable-background="new 0 0
8   400 600" xml:space="preserve">
9 <defs>
10 <linearGradient id="night"
11   gradientUnits="userSpaceOnUse" x1="199.6667"
12   y1="71.434" x2="199.6667" y2="517.3639">
13   <stop offset="0" style="stop-color:#222B33"/>
```

```
CSS
1 @import url(https://fonts.googleapis.com/css?
2   family=Raleway:700);
3 html{
4   font-family: 'Raleway', sans-serif;
5 }
6 body{
7   background-color:;
8   overflow:hidden;
9 }
10 body, html {
11   height: 100%;
12   width: 100%;
13   margin: 0;
14   padding: 0;
```

```
JS
1 var xmlns="http://www.w3.org/2000/svg"
2 var container =
3   document.querySelector('.container');
4 var nightDaySVG =
5   document.querySelector('.nightDaySVG');
6 var sun = document.querySelector('.sun');
7 var moon = document.querySelector('.moon');
8 var lights = document.querySelector('.lights');
9 var day = document.querySelector('.day');
10 var satDish = document.querySelector('.satDish');
11 var starContainer =
12   document.querySelector('.starContainer');
13 var frameMask =
14   document.querySelector('.frameMask');
```

LogoMotion

[Liu 2025]



STATIC LOGO IMAGE

```
<div> <!-- Group -->


</div>
```

HTML from 2

For the animation of the primary element, the silhouette of the person skiing you could create a motion...



```
var timeline = anime.timeline({ easing:
'easeOutExpo', duration: 750
}); // Begin by animating the background element which
is the sky

timeline.add({ targets: '#WQO', translateY:
[-512, 0], opacity: [0, 1]
}); // Animate the mountains to 'rise' up

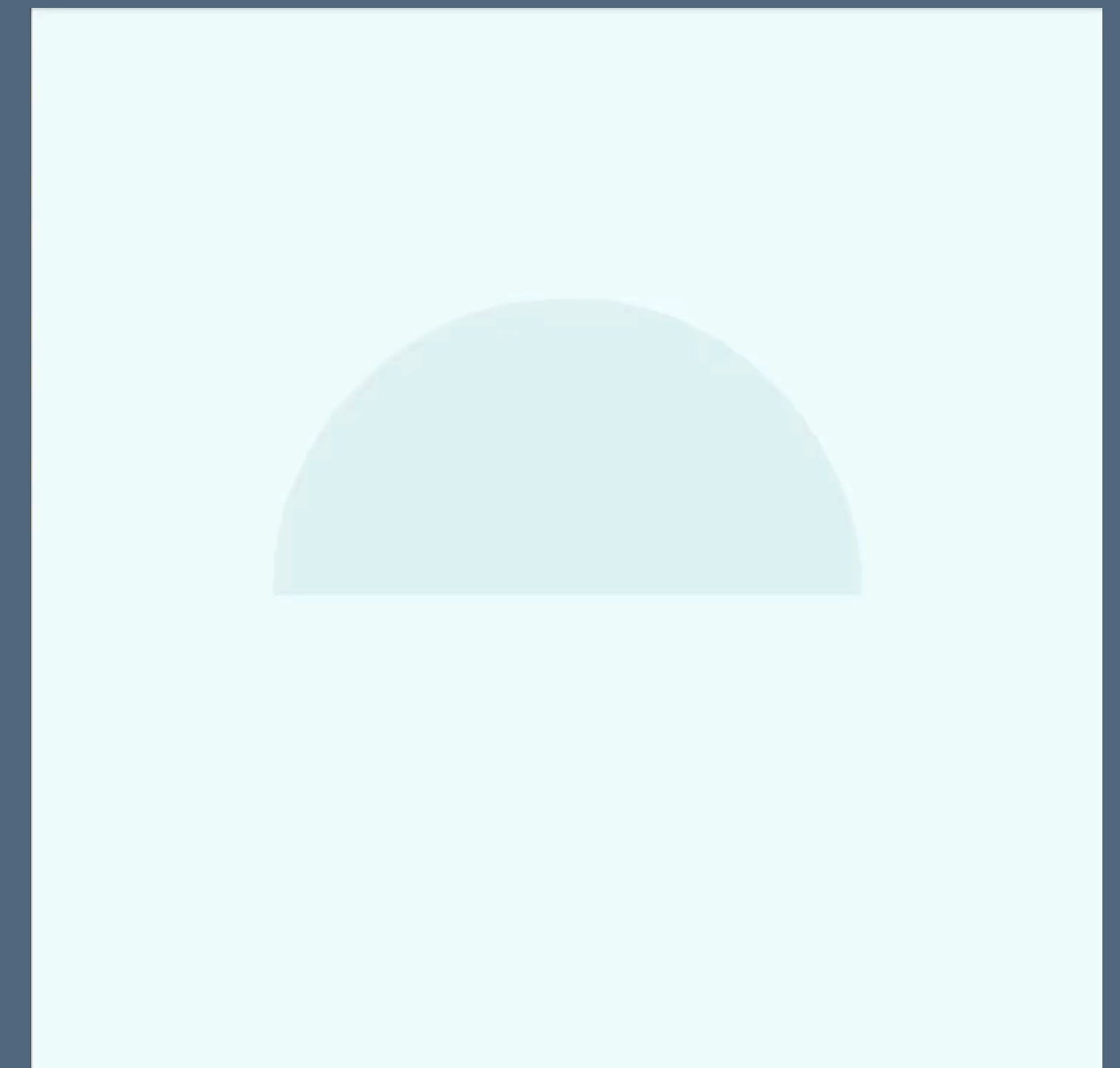
timeline.add({ targets: ['#DXJ', '#KXR'],
translateY: [512, 0], opacity: [0, 1], delay:
anime.stagger(100) // Stagger the start times of
animations for individual targets }, '-=600'); // Overlap
with the previous animation // Slide in the horizontal
lines

timeline.add({ targets: ['#MVX', '#OFQ'],
translateX: [-512, 0], opacity: [0, 1], delay:
anime.stagger(100) }, '-=500'); // Animate the text
"CARDINALS"

timeline.add({ targets: '#CZF', translateX: [512,
0], opacity: [0, 1], delay: anime.stagger(100) },
'-=400');

timeline.add({ targets: '#SNT', translateX: [512,
0], opacity: [0, 1], easing: 'easeOutExpo',
duration: 750, delay: anime.stagger(100) }, '-=400');

timeline.add({ targets: '#OWE', translateX:
[-512, 0], translateY: [anime.random(-10, 10), 0],
scale: [1.5, 1], opacity: [0, 1], rotate: {
value: [anime.random(-5, 5), 0], duration:
1500, easing: 'easeOutCubic' }, duration:
1500, easing: 'easeOutCubic' }, '-=300');
```



Formal representation of natural language

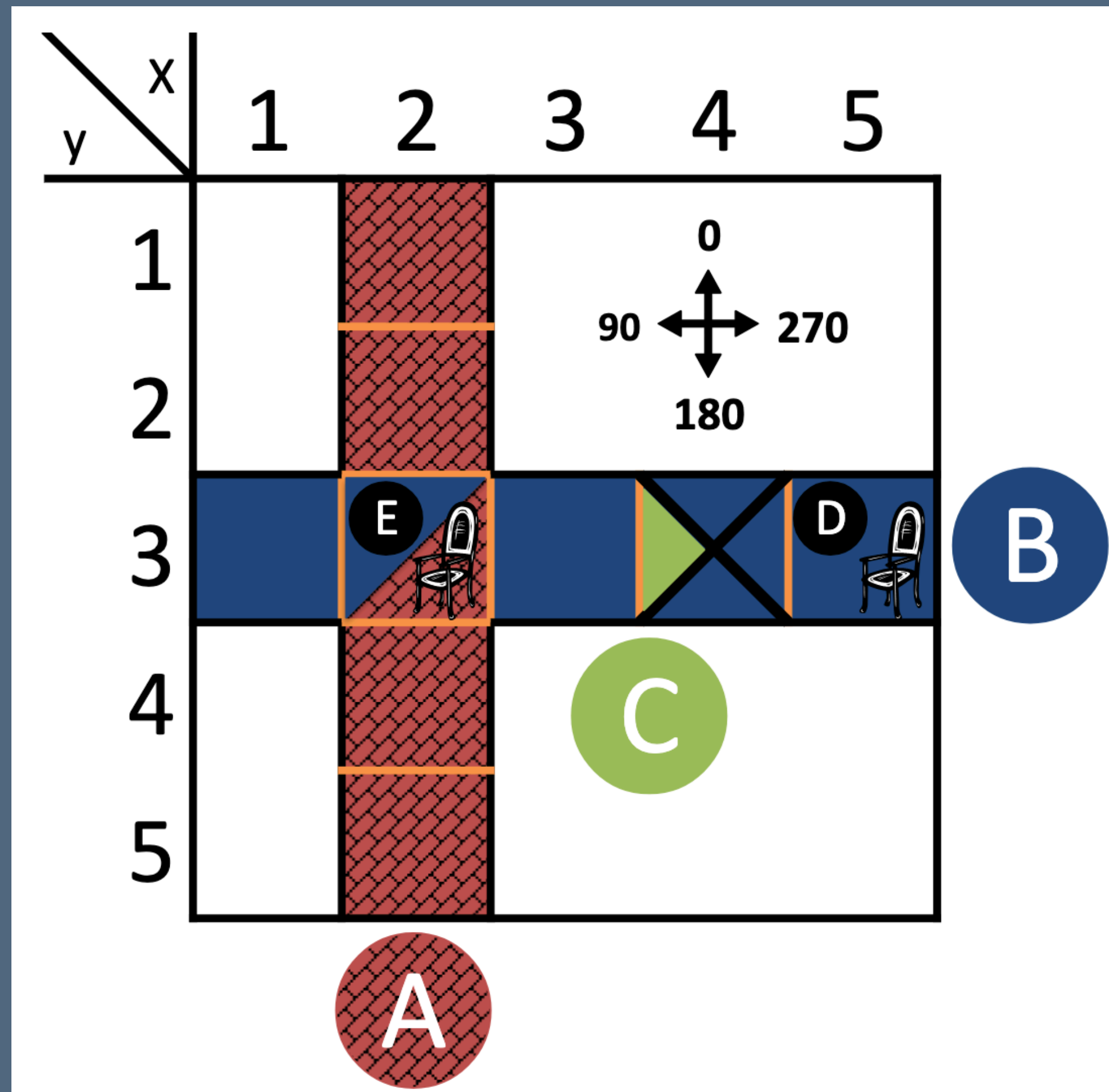
Natural language is ambiguous, underspecified, and hard for machines to reason over directly.

Long-standing approach in linguistics and AI: represent meaning as logical expressions with variables, predicates, and quantifiers.

"A red ball is on the table." $\rightarrow \exists x. \text{ball}(x) \wedge \text{red}(x) \wedge \text{on}(x, \text{table})$

Instructions to Actions

[Artzi and Zettlemoyer 2013]



- $\{ \text{D} \text{ E} \}$ (a) chair
 $\lambda x.chair(x)$
- $\{ \text{A} \text{ B} \}$ (b) hall
 $\lambda x.hall(x)$
- E (c) the chair
 $\iota x.chair(x)$
- C (d) you
 you
- $\{ \text{B} \}$ (e) blue hall
 $\lambda x.hall(x) \wedge blue(x)$
- $\{ \text{E} \}$ (f) chair in the intersection
 $\lambda x.chair(x) \wedge$
 $intersect(\iota y.junction(y), x)$
- $\{ \text{A} \text{ B} \text{ E} \}$ (g) in front of you
 $\lambda x.in_front_of(you, x)$

Instructions to Actions

[Artzi and Zettlemoyer 2013]

move forward twice to the chair

$\lambda a.move(a) \wedge dir(a, forward) \wedge len(a, 2) \wedge$
 $to(a, \iota x.chair(x))$

at the corner turn left to face the blue hall

$\lambda a.pre(a, \iota x.corner(x)) \wedge turn(a) \wedge dir(a, left) \wedge$
 $post(a, front(you, \iota x.blue(x) \wedge hall(x)))$

move to the chair in the third intersection

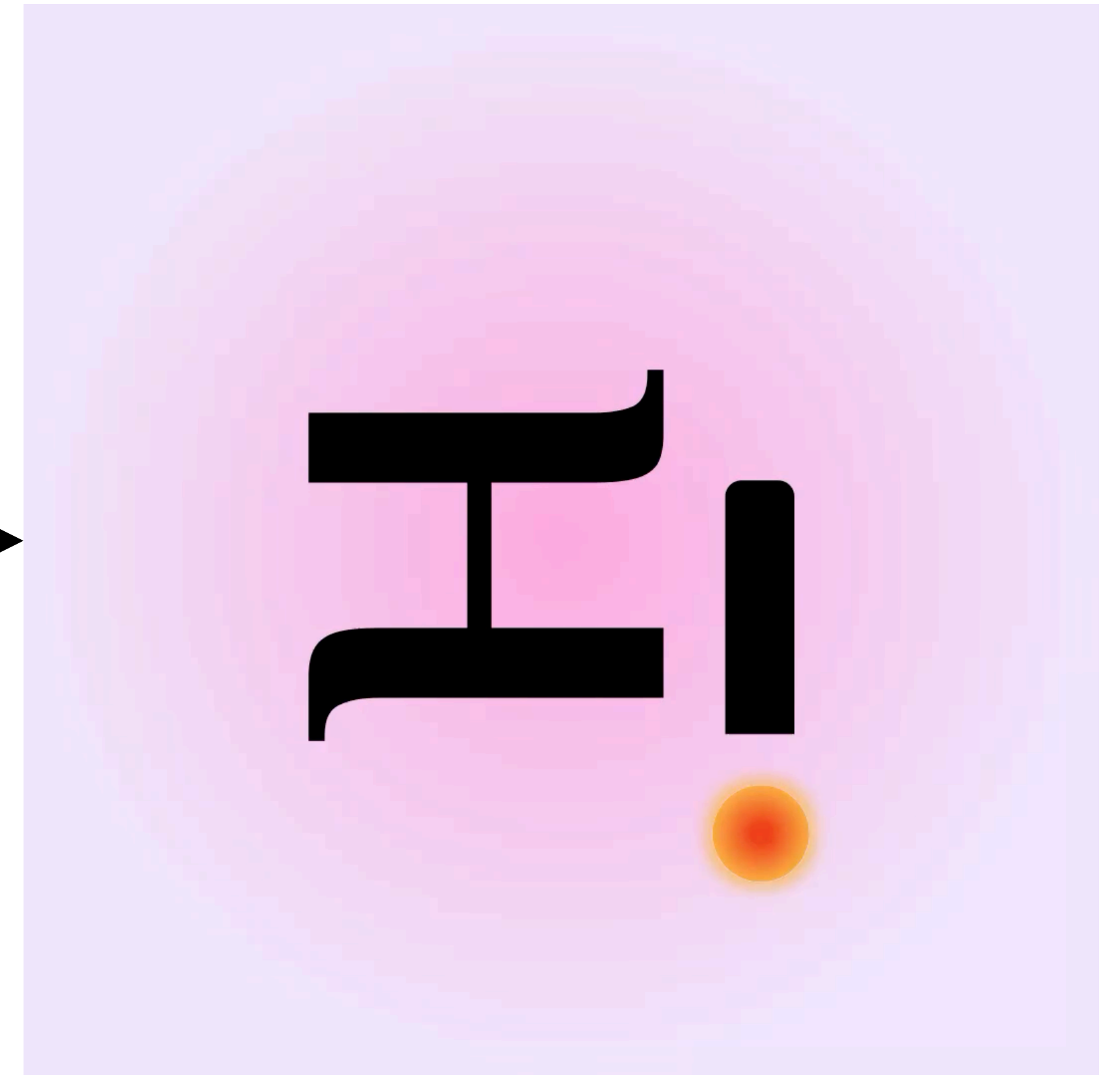
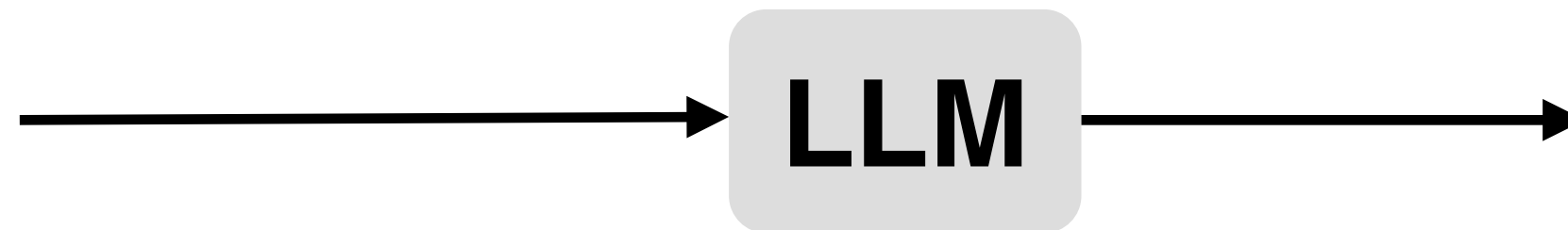
$\lambda a.move(a) \wedge to(a, \iota x.sofa(x)) \wedge$
 $intersect(order(\lambda y.junction(y), frontdist, 3), x)$

MoVer

[Ma and Agrawala 2025]



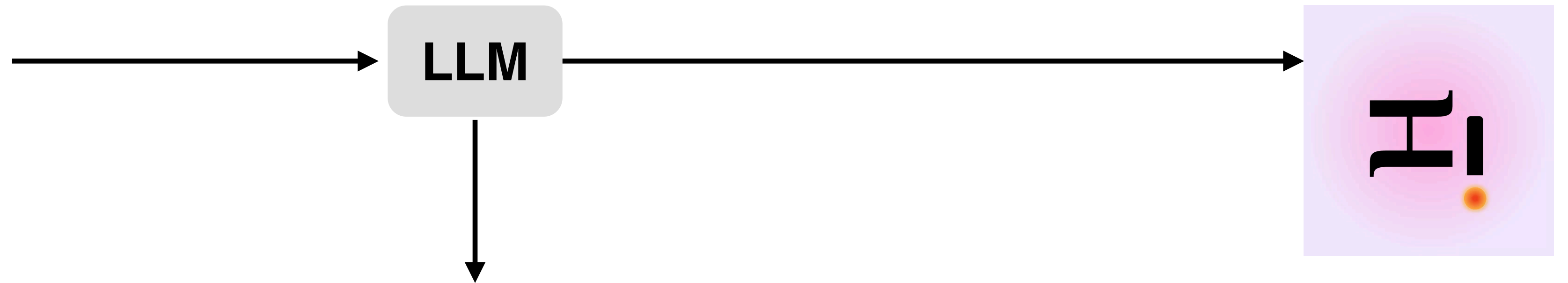
The orange circle translates above the rectangular shape. In the meantime, the letter H rotates 90 degrees clockwise.



MoVer

[Ma and Agrawala 2025]

The orange circle translates above the rectangular shape. In the meantime, the letter H rotates 90 degrees clockwise.



$o_1 = \text{lo.clr}(o, \text{"orange"}) \wedge \text{shp}(o, \text{"circle"})$

$o_2 = \text{lo.shp}(o, \text{"rectangle"})$

$o_3 = \text{lo.id}(o, \text{"H"})$

$m_1 = \text{lm.type}(m, \text{"trn"}) \wedge \text{agt}(m, o_1) \wedge \text{post}(m, \text{top}(o_1, o_2))$

$m_2 = \text{lm.type}(m, \text{"rot"}) \wedge \text{agt}(m, o_3) \wedge \text{dir}(m, \text{"cw"}) \wedge \text{mag}(m, 90)$

$\text{while}(m_1, m_2)$

MoVer

[Ma and Agrawala 2025]

The orange circle translates above the rectangular shape. In the meantime, the letter H rotates 90 degrees clockwise.

LLM

$o_1 = \text{lo.clr}(o, \text{"orange"}) \wedge \text{shp}(o, \text{"circle"})$

$o_2 = \text{lo.shp}(o, \text{"rectangle"})$

$o_3 = \text{lo.id}(o, \text{"H"})$

$m_1 = \text{lm.type}(m, \text{"trn"}) \wedge \text{agt}(m, o_1) \wedge \text{post}(m, \text{top}(o_1, o_2))$

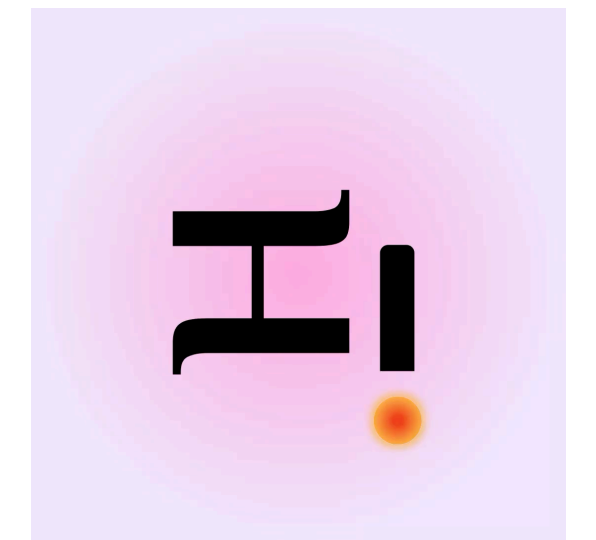
$m_2 = \text{lm.type}(m, \text{"rot"}) \wedge \text{agt}(m, o_3) \wedge \text{dir}(m, \text{"cw"}) \wedge \text{mag}(m, 90)$

$\text{while}(m_1, m_2)$

T
T
T

F
T

F

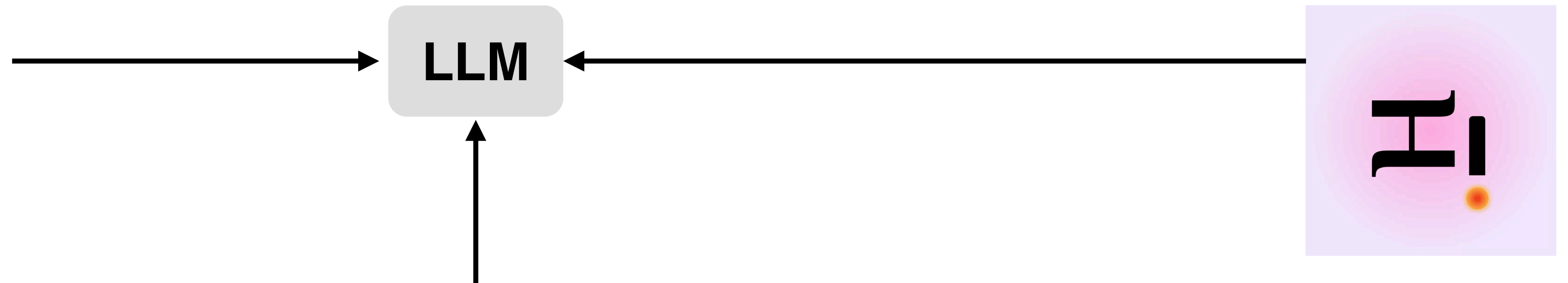


MoVer Execution Engine

MoVer

[Ma and Agrawala 2025]

The orange circle translates above the rectangular shape. In the meantime, the letter H rotates 90 degrees clockwise.



$O_1 = \text{lo.clr}(o, \text{"orange"}) \wedge \text{shp}(o, \text{"circle"})$

$O_2 = \text{lo.shp}(o, \text{"rectangle"})$

$O_3 = \text{lo.id}(o, \text{"H"})$

T
T
T

MoVer
Execution Engine

$m_1 = \text{lm.type}(m, \text{"trn"}) \wedge \text{agt}(m, O_1) \wedge \text{post}(m, \text{top}(O_1, O_2))$

$m_2 = \text{lm.type}(m, \text{"rot"}) \wedge \text{agt}(m, O_3) \wedge \text{dir}(m, \text{"cw"}) \wedge \text{mag}(m, 90)$

F
T

$\text{while}(m_1, m_2)$

F

MoVer

[Ma and Agrawala 2025]

The orange circle translates above the rectangular shape. In the meantime, the letter H rotates 90 degrees clockwise.

LLM

```
o1 = lo.clr(o, "orange") ^ shp(o, "circle")
```

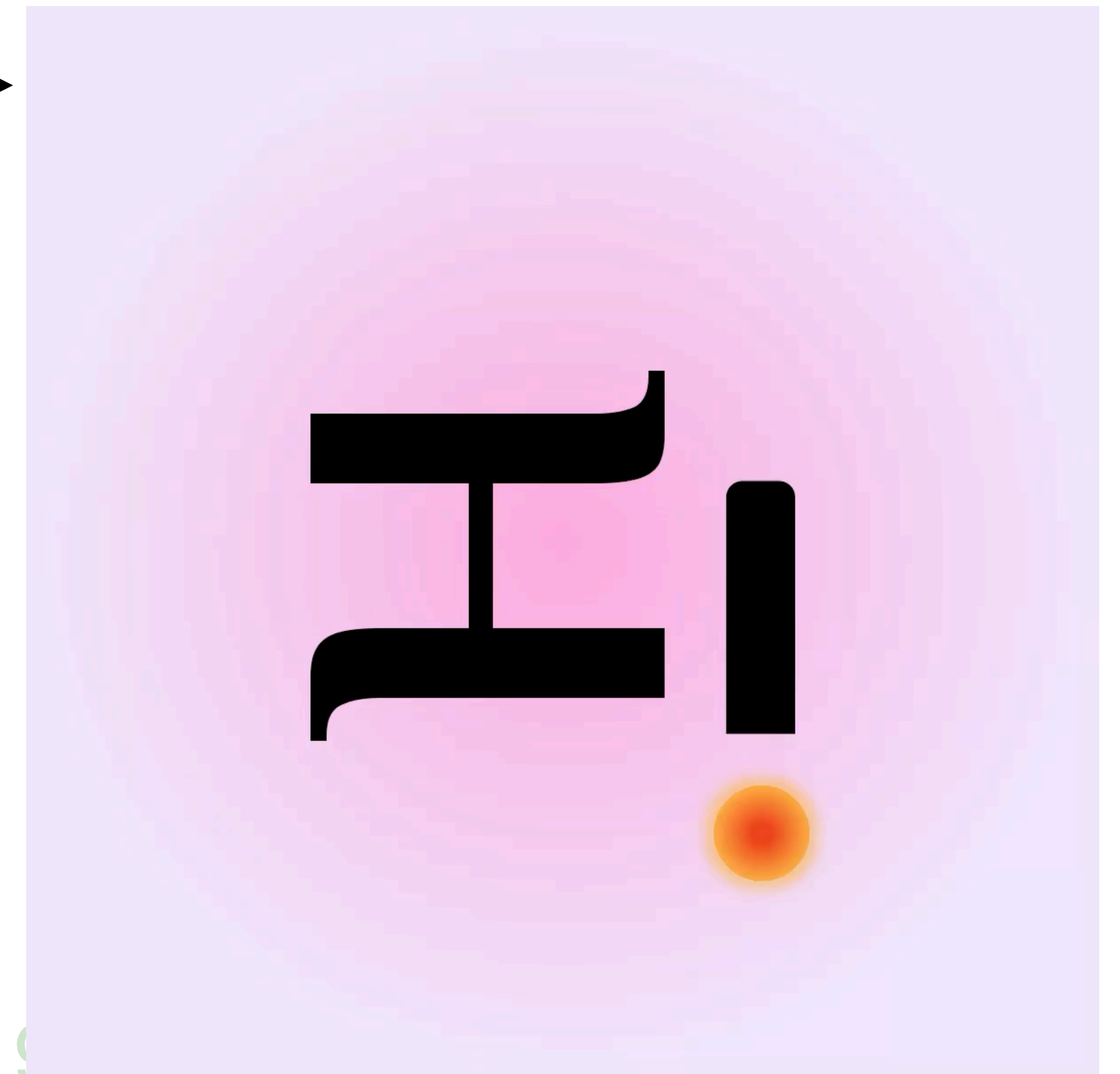
```
o2 = lo.shp(o, "rectangle")
```

```
o3 = lo.id(o, "H")
```

```
m1 = lm.type(m, "trn") ^ agt(m, o1) ^ post(m, top(o1, o2))
```

```
m2 = lm.type(m, "rot") ^ agt(m, o3) ^ dir(m, "cw") ^ mag(m, 90,
```

```
while(m1, m2)
```



Designing shared representation

1. The shared representation can directly be *in the same format* as the user input and the machine output
2. The shared representation can be an intermediate layer between the user input and the machine output when 1. is not feasible

How can we help humans
to *more efficiently interpret*
these intermediate layers?

Sketch-n-Sketch

[Hempel 2019]

The image shows a screenshot of the Sketch-n-Sketch application interface. On the left, there are three diagrams labeled e, d, and f, which illustrate the recursive construction of a fractal shape. Diagram e shows a triangle with three points. Diagram d shows a triangle with three points and three smaller triangles attached to its sides. Diagram f shows a more complex fractal shape with many points.

The main window displays the code for the fractal:

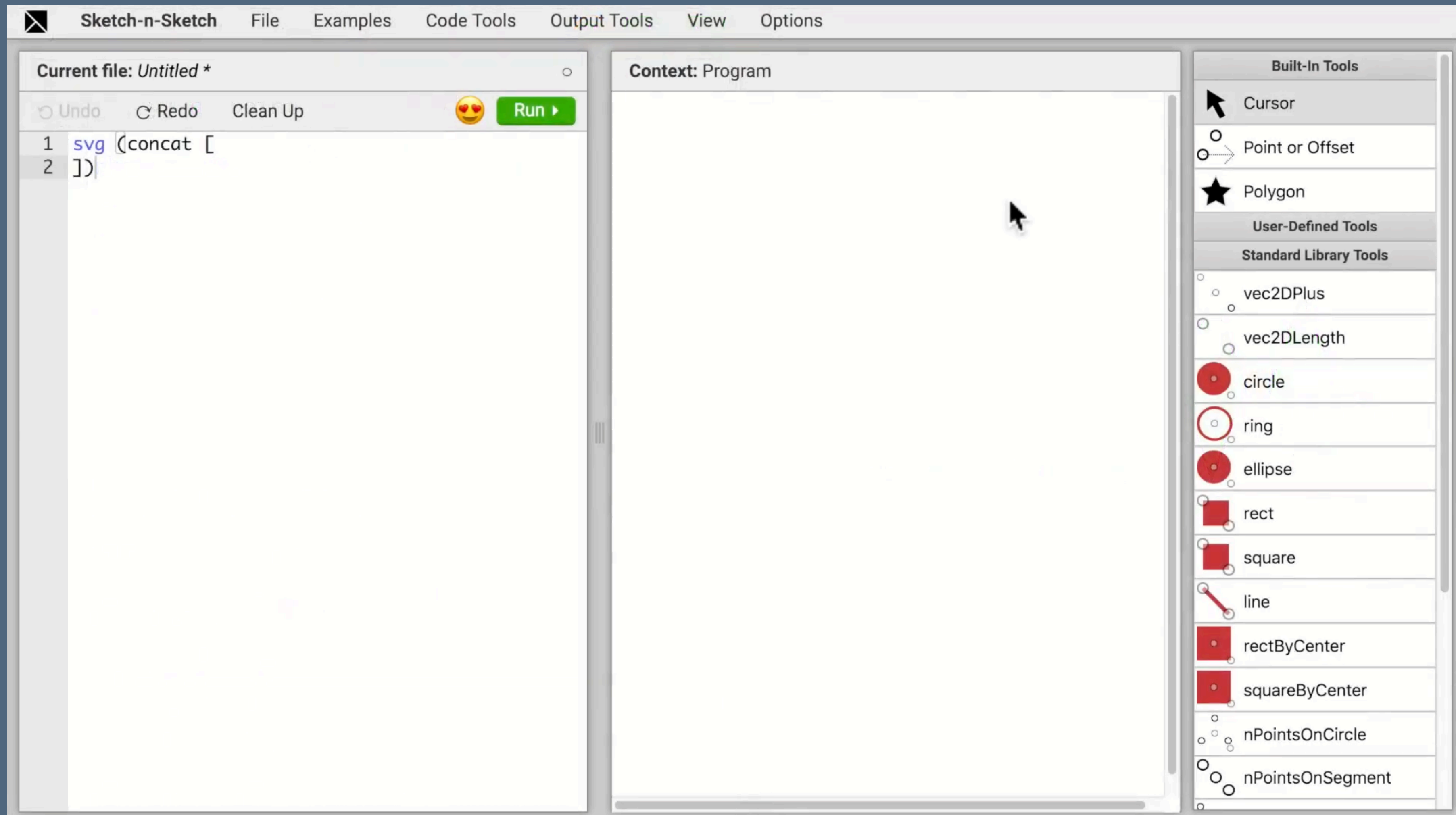
```
1 equiTriPt [x3, y3] [x2, y2] =  
2 [ (x2 + x3 + sqrt 3! * (y2 - y3))/ 2!, (y2 + y3 - sqrt 3! * (x2 - x3)) / 2!]  
3  
4 oneThirdPt [x3, y3] [x, y] =  
5 [ x / 1.5!+ x3 / 3!, y / 1.5! + y3 / 3!]  
6  
7 point = [39, 314]  
8  
9 point2 = [490, 301]  
10  
11 makeKochPts depth point point2 =  
12 let oneThirdPt2 = oneThirdPt point point2 in  
13 let oneThirdPt3 = oneThirdPt point2 point in  
14 let equiTriPt2 = equiTriPt oneThirdPt3 oneThirdPt2 in  
15 if depth < 2 then  
16 [point, oneThirdPt3, equiTriPt2, oneThirdPt2]  
17 else  
18 let makeKochPts2 = makeKochPts (depth - 1) point oneThirdPt3 in  
19 let makeKochPts3 = makeKochPts (depth - 1) oneThirdPt3 equiTriPt2 in  
20 let makeKochPts4 = makeKochPts (depth - 1) equiTriPt2 oneThirdPt2 in  
21 let makeKochPts5 = makeKochPts (depth - 1) oneThirdPt2 point2 in  
22 concat [makeKochPts2, makeKochPts3, makeKochPts4, makeKochPts5]  
23  
24 depth = 3{1-5}  
25  
26 topPts = makeKochPts depth point point2  
27  
28 botCorner = equiTriPt point2 point  
29  
30 rightPts = makeKochPts depth point2 botCorner  
31  
32 leftPts = makeKochPts depth botCorner point  
33  
34 snowflakePts = concat [topPts, rightPts, leftPts]  
35  
36 polygon1 =  
37 let pts = snowflakePts in  
38 let [color, strokeColor, strokeWidth] = [124, 360, 2] in  
39 polygon color strokeColor strokeWidth pts  
40  
41 svg (concat [  
42 [polygon1]  
43 ])
```

The right side of the window shows a tool palette with a list of tools. The 'User-Defined Tools' section is highlighted, showing the tools 'equiTriPt', 'oneThirdPt', and 'makeKochPts'. The 'Standard Library Tools' section includes tools like 'circle', 'ring', 'ellipse', 'rect', 'square', 'line', 'rectByCenter', 'squareByCenter', 'nPointsOnCircle', 'nPointsOnSegment', 'nPointsSepBy', 'nHorizontalPointsSepBy', 'nVerticalPointsSepBy', 'pointsBetweenSepBy', and 'midpoint'.

Labels a, b, and c are used to highlight specific parts of the interface: 'a' points to the tool palette, 'b' points to the 'User-Defined Tools' section, and 'c' points to the 'makeKochPts' tool.

Sketch-n-Sketch

[Hempel 2019]



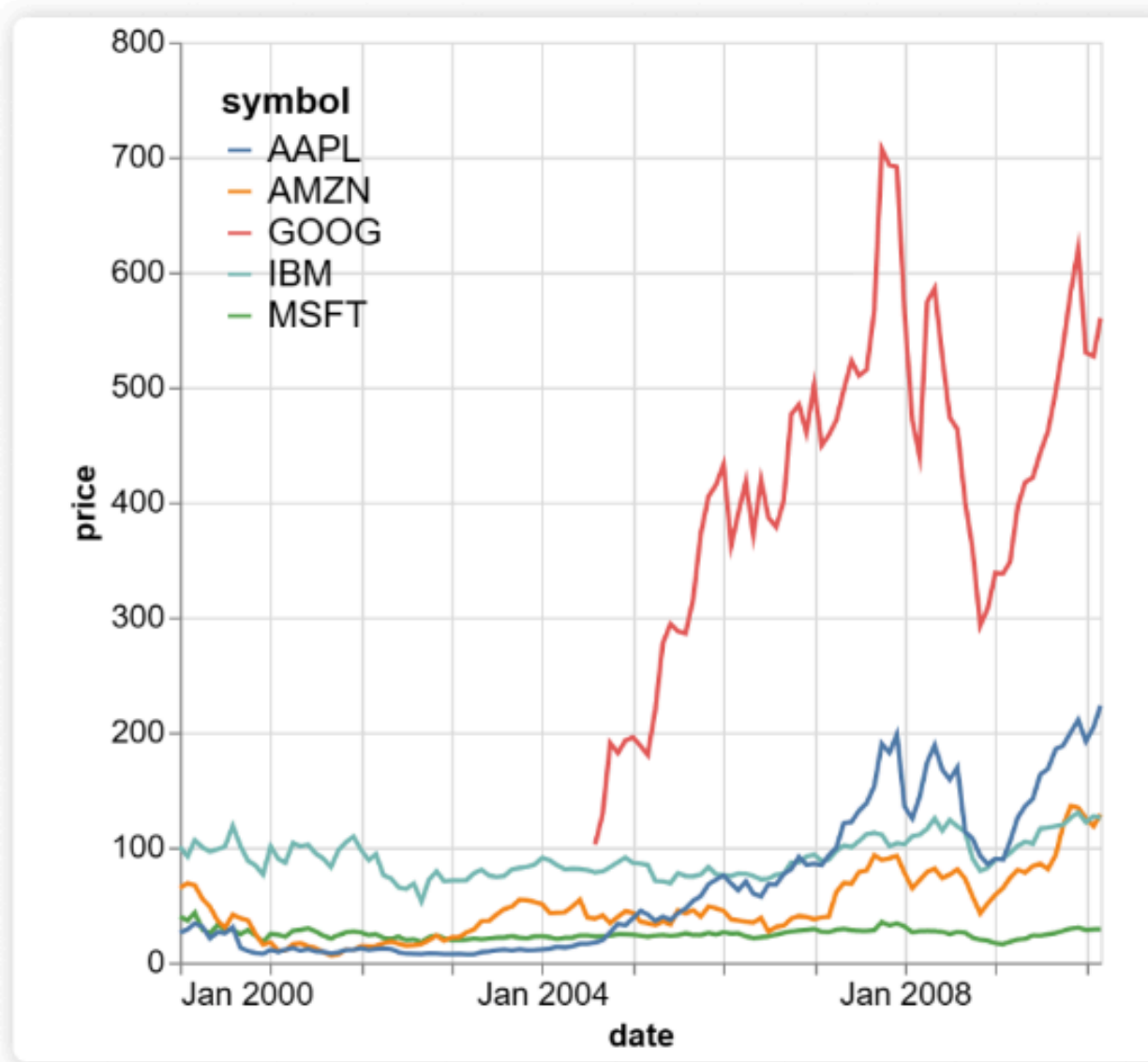
Designing shared representation

1. The shared representation can directly be *in the same format* as the user input and the machine output.
2. The shared representation can be an intermediate layer between the user input and the machine output when 1. is not feasible.
3. The shared representation can be designed to improve readability for the user while maintaining interpretability for the machine.

DynaVis

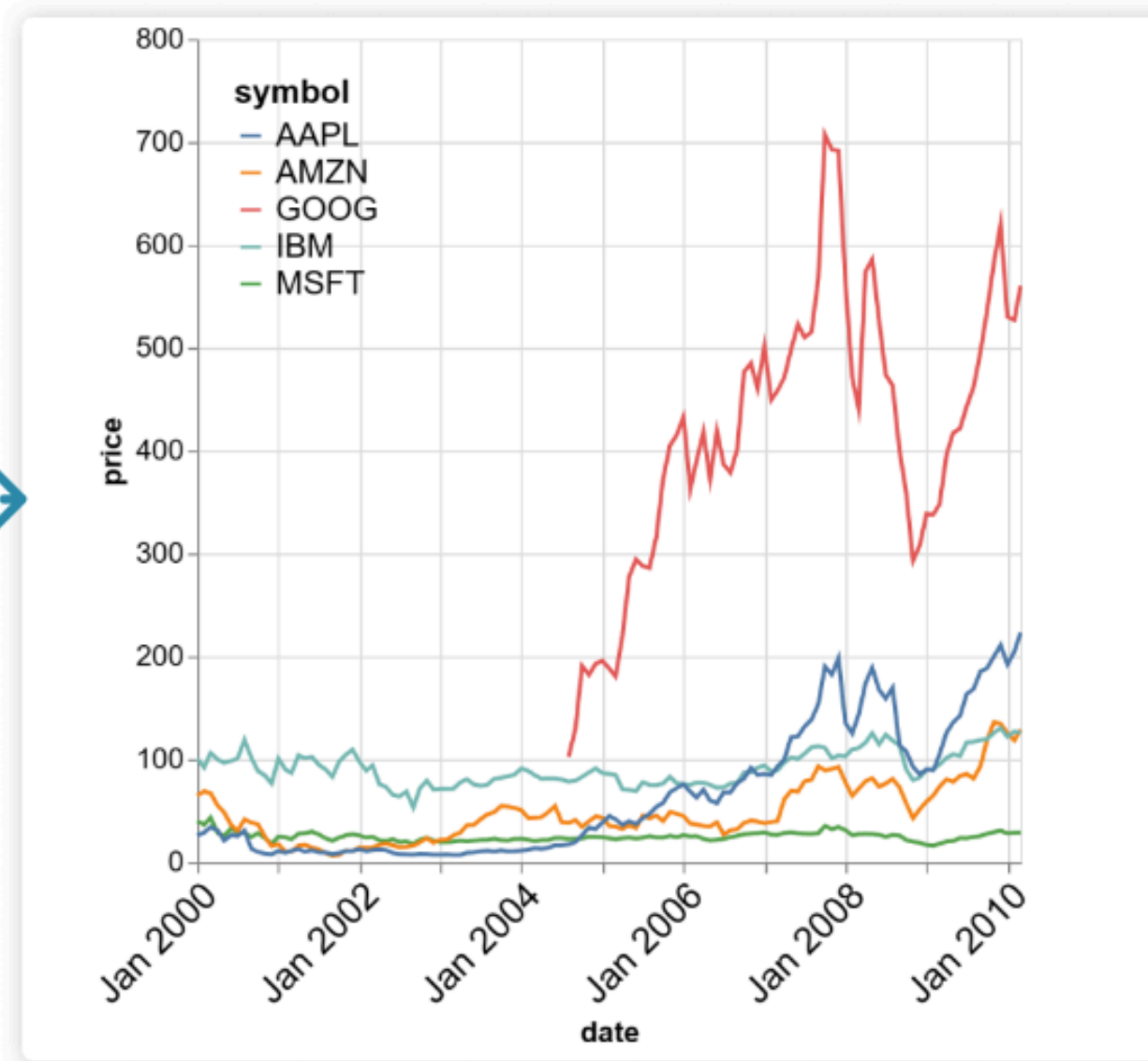
[Vaithilingam 2024]

Increase the x-axis text size to 20 and rotate the labels by 60 degree counter-clockwise



1 Alice gives a natural language command to edit the chart

How would you like to modify the chart?



Widgets

X-Axis Label Editor

X-Axis Font Size: 20

X-Axis Rotation Angle:

DynaVis automatically synthesizes a widget for the edits

DynaVis

[Vaithilingam 2024]

X-Axis Label Editor



X Axis Label Size:



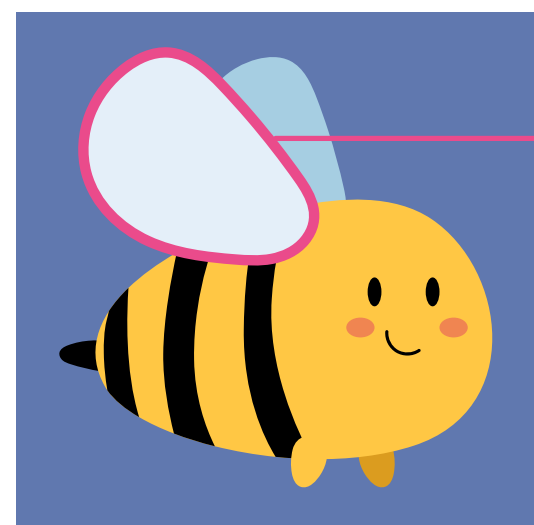
```
<div class="widget">
  <label for="xAxisSize">X Axis Label Size:</label>
  <input type="range" id="xAxisSize" name="xAxisSize"
min="2" max="80" step="1" value="14">
</div>
```

```
function callback(event, spec) {
  if(spec.encoding.x.axis) {
    spec.encoding.x.axis.labelFontSize = event.target.value;
  } else {
    spec.encoding.x.axis = {"labelFontSize": event.target.value};
  }
  return spec;
}
```

VibeAnimation

[Ma 2026]

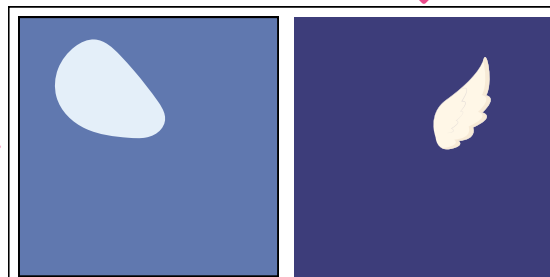
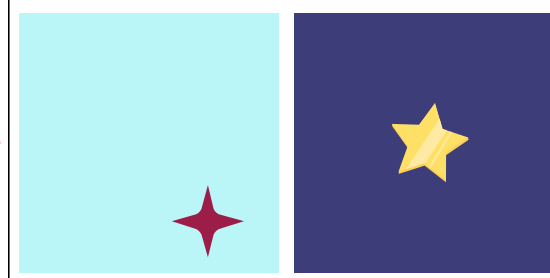
Source animations



Motion concept specs

Concept description
A "twinkle" concept where an element expands and dims

- Concept parameters**
- expansion (...)
 - dimness (...)
 - frequency (...)



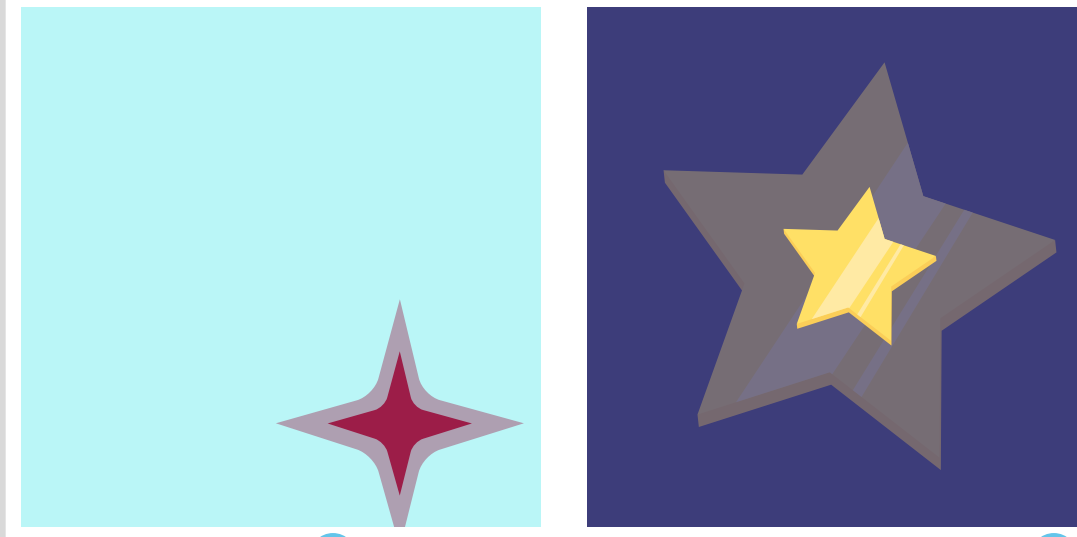
Concept description
A "wing flap" where an element rotates around a pivot point

- Concept parameters**
- flapRange (...)
 - pivot (...)
 - speed (...)

Motion concept function

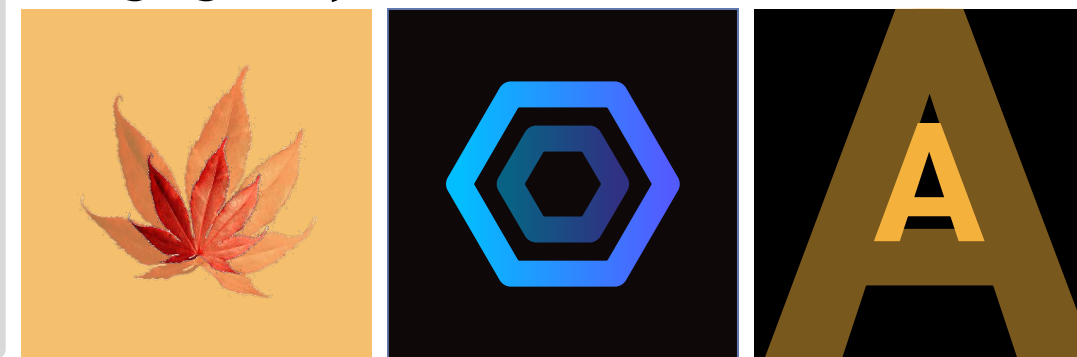
```
twinkle(elem, tl, startTime, //required  
        expansion, dimness, frequency)
```

Function control panel



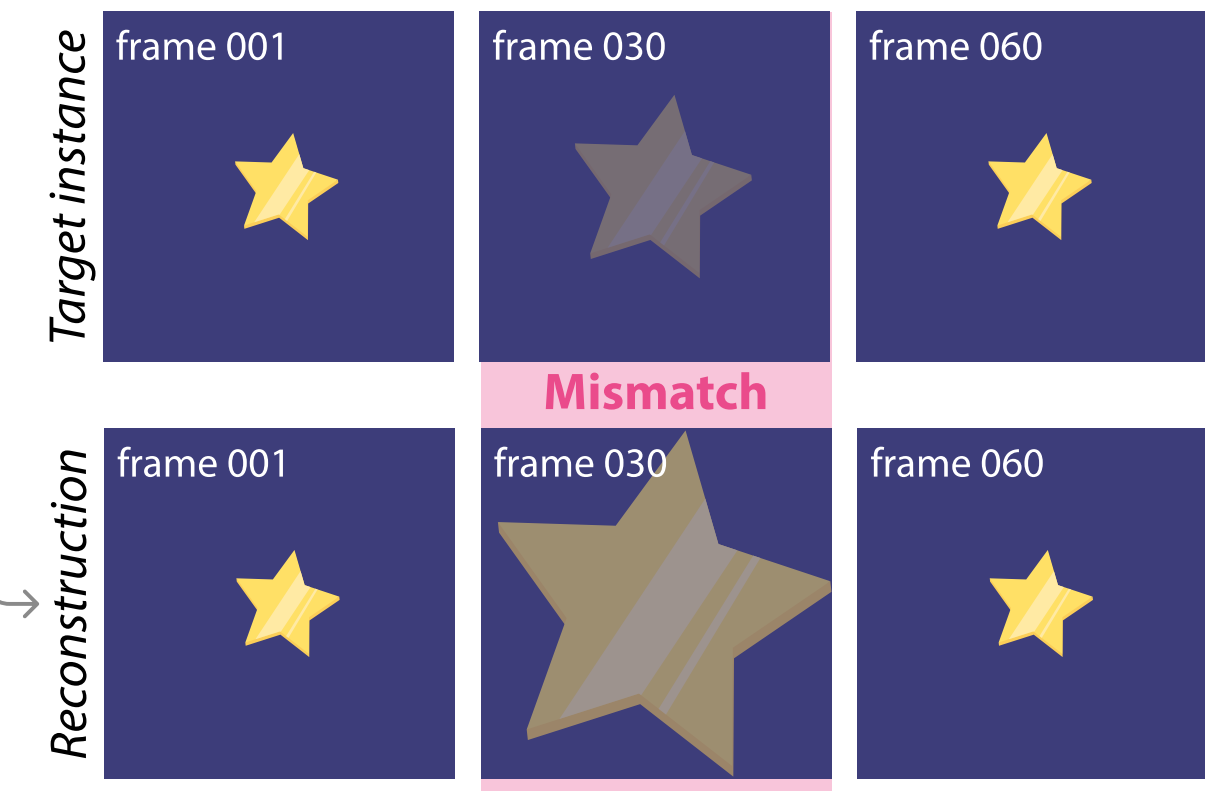
- expansion dimness frequency

Design gallery



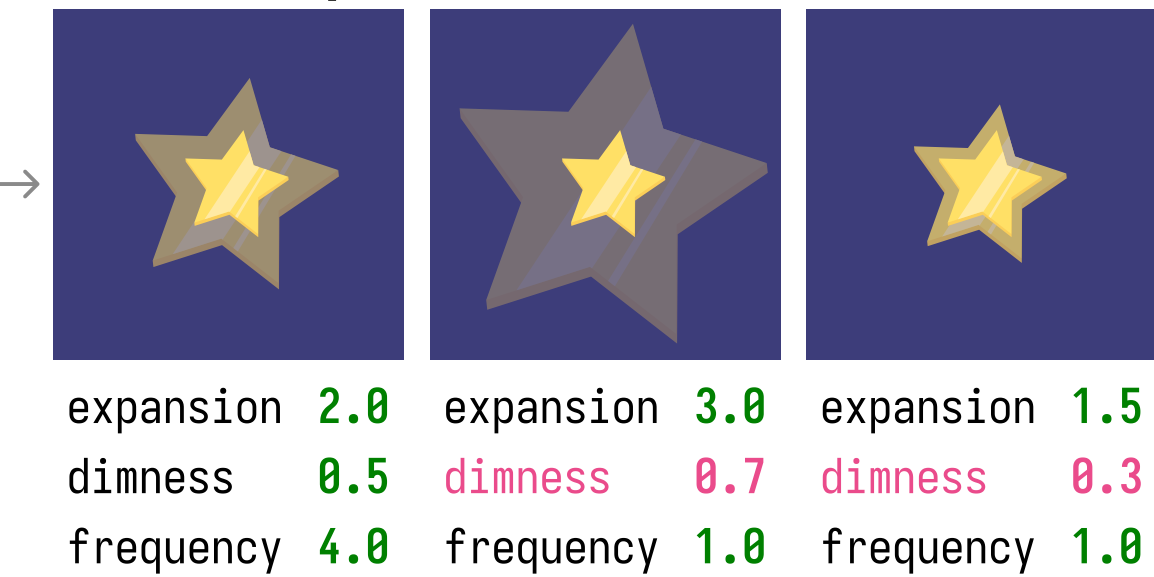
```
wingFlap(elem, tl, startTime, //required  
         flapRange, pivot, speed)
```

Automated instance reconstruction



```
twinkle(..., exp=4.0, dim=0.5, freq=1.0)
```

Automated parameter behavior verification



Motion concept specification

Motion concept function generation

Motion concept function verification

VibeAnimation

[Ma 2026]

Source animations

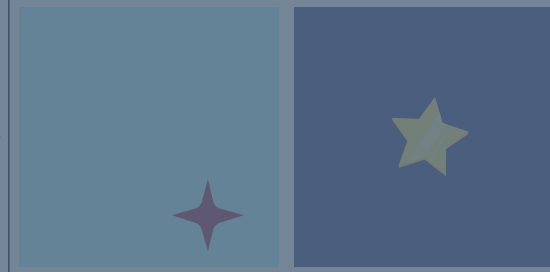


Motion concept specs

Concept description
A "twinkle" concept where an element expands and dims

Concept parameters

- expansion (...)
- dimness (...)
- frequency (...)



Concept description
A "wing flap" where an element rotates around a pivot point

Concept parameters

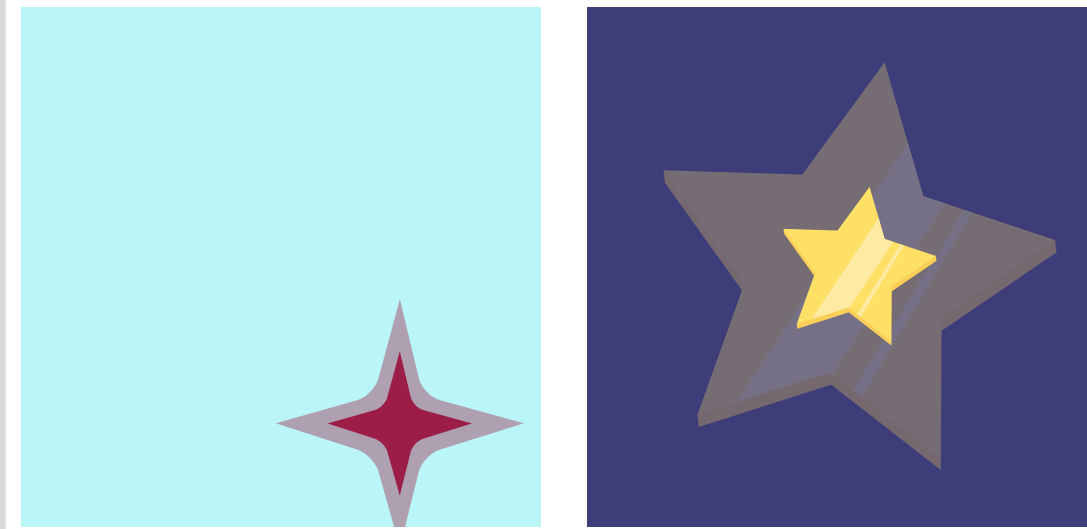
- flapRange (...)
- pivot (...)
- speed (...)

Motion concept specification

Motion concept function

```
twinkle(elem, tl, startTime, //required  
        expansion, dimness, frequency)
```

Function control panel



expansion expansion
dimness dimness
frequency frequency

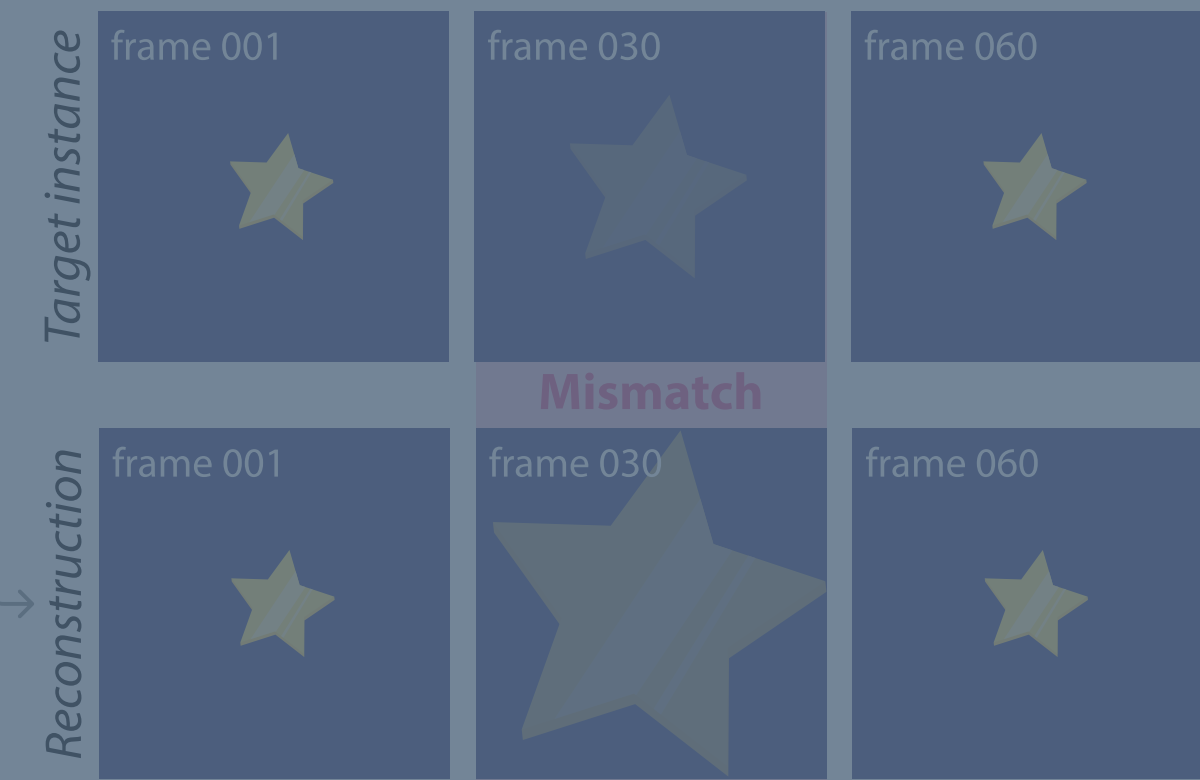
Design gallery



```
wingFlap(elem, tl, startTime, //required  
         flapRange, pivot, speed)
```

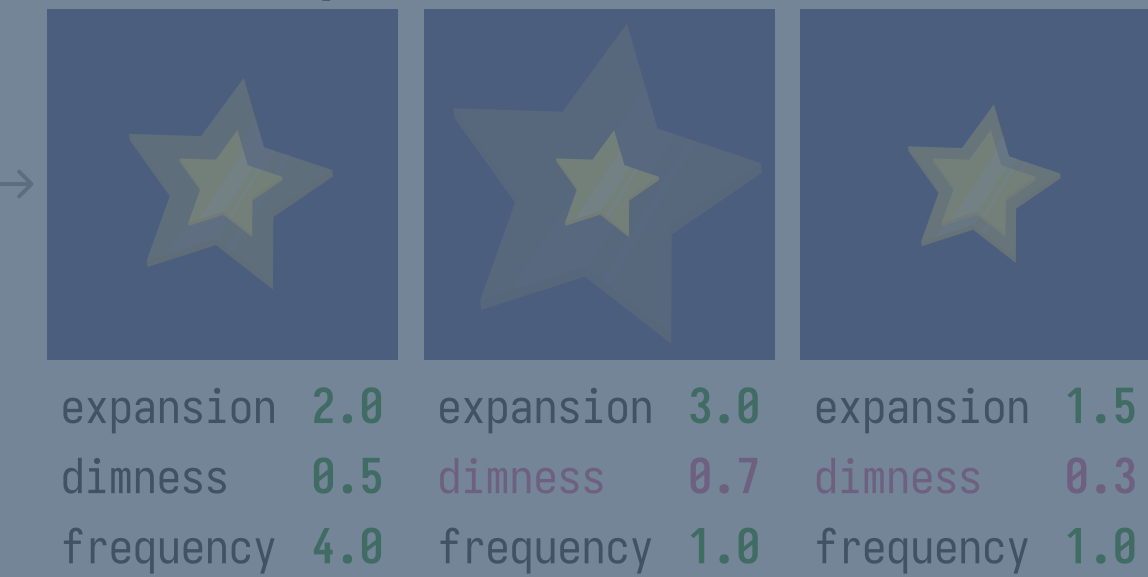
Motion concept function generation

Automated instance reconstruction



```
twinkle(..., exp=4.0, dim=0.5, freq=1.0)
```

Automated parameter behavior verification



Motion concept function verification

VibeAnimation

[Ma 2026]

The screenshot displays the VibeAnimation web application interface, which is divided into several sections:

- ANIMATION CREATION:** Located on the left side, it includes a "Static SVG scenes" section with a "Select a SVG file" dropdown and a "Library functions" section with a "twinkle_0418_abs_iter_0.js" dropdown. Below these is a preview window showing a yellow star on a dark blue background.
- Code Editor:** At the top right, it shows the filename "0418_fig_twinkle.json" and a "New chat name" input. The main area contains a code editor with the following content:

```
//@SVG_META: {"url": "server/output/0418_fig_twinkle/0418_fig_twinkle_example_1_abs_0_apply_0.html", "type": "new"}
```
- Playback Controls:** A green progress bar is visible, with a play/pause button and a "1x" speed indicator. The current time is shown as "0.67 / 0.70".
- Function parameters:** A detailed configuration panel for the animation, including:
 - SVG:** default (The static SVG file being animated)
 - startTime:** 0 (Absolute start time on the timeline.)
 - fillColor:** #FFF9B0 (Highlight fill color at the peak of the twinkle.)
 - opacity:** 1 (Peak opacity reached at mid-cycle before returning to the element's original opacity.)
 - rotation:** -244.8 (Relative rotation in degrees added at the twinkle peak, then reversed back to the starting orientation.)
 - scale:** 0.477 (Multiplicative peak scale factor applied to the element's current scale before returning to its starting size.)
 - twinkleDuration:** 0.7 (Total duration in seconds for the full out-and-back twinkle cycle.)
- Design gallery:** A section at the bottom with the text "> Design gallery [SVGs|Params]" and a text input field with the placeholder "Enter to send, Shift+Enter for new line".

Summary

1. The shared representation can directly be *in the same format* as the user input and the machine output.
2. The shared representation can be an intermediate layer between the user input and the machine output when 1. is not feasible.
 - Programs are often being used as the intermediate layer
3. The shared representation can be designed to improve readability for the user while maintaining interpretability for the machine.

References

Heer, Jeffrey. "Agency plus automation: Designing artificial intelligence into interactive systems." Proceedings of the National Academy of Sciences 116.6 (2019): 1844-1850.

Horvitz, Eric. "Principles of mixed-initiative user interfaces." Proceedings of the SIGCHI conference on Human Factors in Computing Systems.

Maes, Pattie. "Agents that reduce work and information overload." Readings in human-computer interaction. Morgan Kaufmann, 1995. 811-821.

Shneiderman, Ben, and Pattie Maes. "Direct manipulation vs. interface agents." interactions 4.6 (1997): 42-61.

Shneiderman, Ben. Human-centered AI. Oxford University Press, 2022.

Lee, Yong Jae, C. Lawrence Zitnick, and Michael F. Cohen. "Shadowdraw: real-time user guidance for freehand drawing." ACM Transactions on Graphics (ToG) 30.4 (2011): 1-10.

Hertzmann, Aaron, et al. "Image analogies." Seminal Graphics Papers: Pushing the Boundaries, Volume 2. 2023. 557-570.

Cho, Jaemin, Abhay Zala, and Mohit Bansal. "Visual programming for step-by-step text-to-image generation and evaluation." Advances in Neural Information Processing Systems 36 (2023): 6048-6069.

Surís, Dídac, Sachit Menon, and Carl Vondrick. "Vipergpt: Visual inference via python execution for reasoning." Proceedings of the IEEE/CVF international conference on computer vision. 2023.

References

Liu, Vivian, et al. "Logomotion: Visually-grounded code synthesis for creating and editing animation." Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems. 2025.

Artzi, Yoav, and Luke Zettlemoyer. "Weakly supervised learning of semantic parsers for mapping instructions to actions." Transactions of the association for computational linguistics 1 (2013): 49-62.

Ma, Jiaju, and Maneesh Agrawala. "Mover: Motion verification for motion graphics animations." ACM Transactions on Graphics (TOG) 44.4 (2025): 1-17.

Hempel, Brian, Justin Lubin, and Ravi Chugh. "Sketch-n-sketch: Output-directed programming for svg." Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology. 2019.

Vaithilingam, Priyan, et al. "Dynavis: Dynamically synthesized ui widgets for visualization editing." Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems. 2024.

<https://medium.com/free-code-camp/things-you-need-to-know-about-working-with-svg-in-vs-code-63be593444dd>

<https://codepen.io/chrisgannon/pen/GJpyrb>