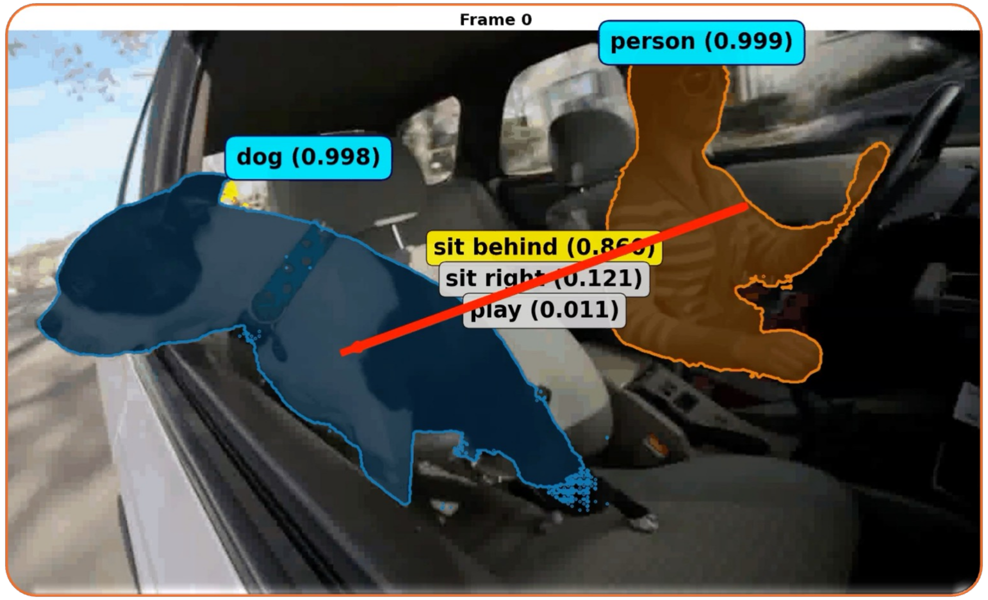
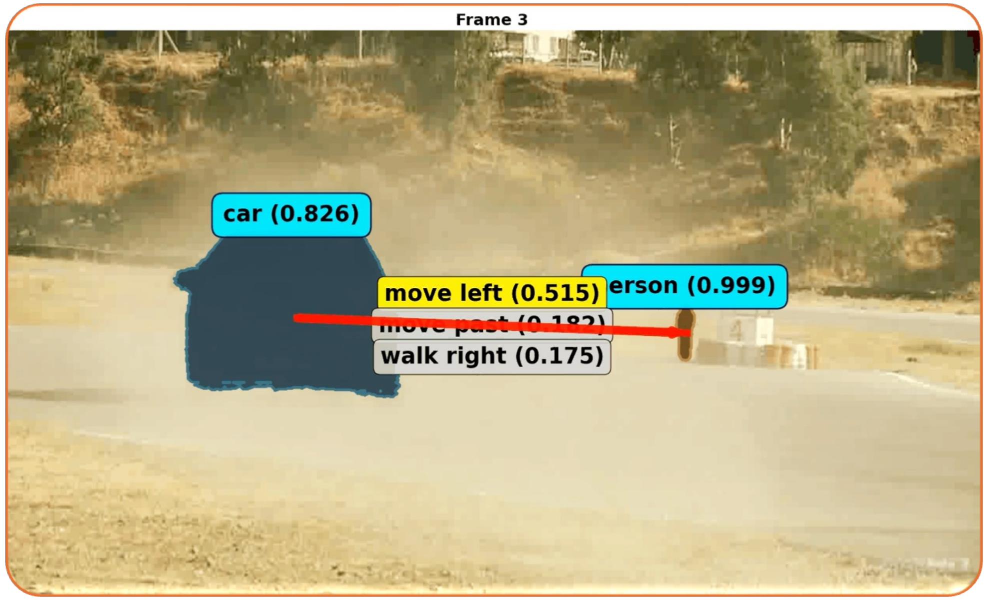
# Machine Programming

Lecture 19 – Special Topic:

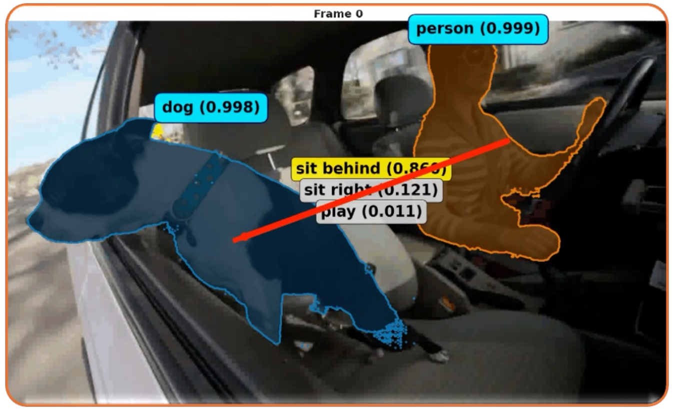
Neurosymbolic Programming & Application to Computer Vision

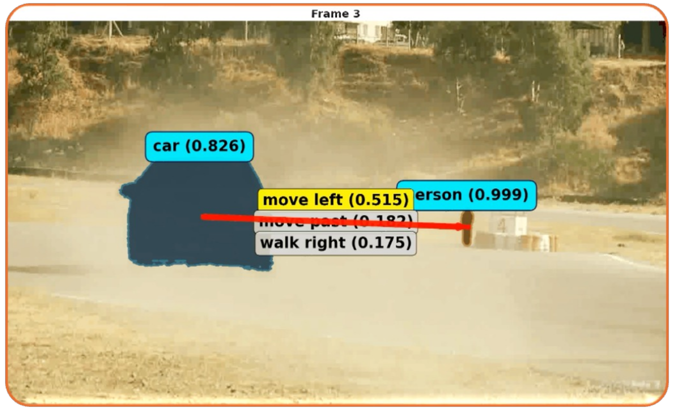
# Spatio-Temporal Scene Graphs (STSGs)



# Scene Graphs



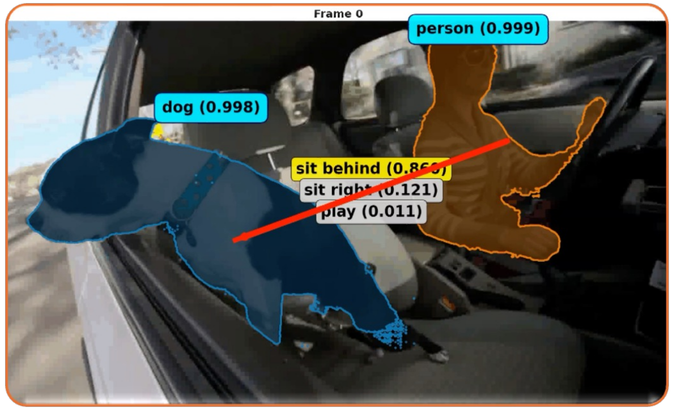



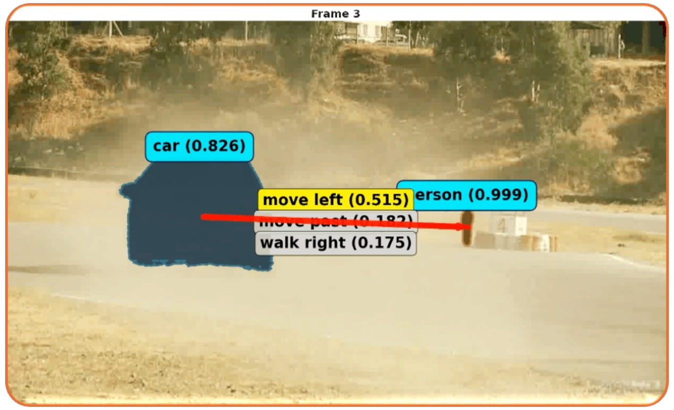

- Node: Entity
  - grounded by bounding-boxes or masks
- Self-loop: Attribute
  - entity class (e.g., person, dog, car)
  - physical properties (e.g., red, bending)
  - intransitive actions (e.g., moving, running)
- Edge: Relations
  - spatial relations (e.g., behind of, above)
  - interactions (e.g., moving towards)

# Spatio-Temporal Scene Graphs

**Evolving over time**

Evolving probabilistic score is assigned to each element (node, edge, etc.) in the graph.




- Node: Evolving Entity
  - grounded by bounding-boxes or masks
- Self-loop: Evolving Attribute
  - entity class (e.g., person, dog, car)
  - physical properties (e.g., red, bending)
  - intransitive actions (e.g., moving, running)
- Edge: Evolving Relations
  - spatial relations (e.g., behind of, above)
  - interactions (e.g., moving towards)

# Spatio-Temporal Scene Graphs: Applications

Identifying that the green cloth is “traversable” meaning that the model needs to understand physical properties of scene entities.

ConceptGraphs: Open-Vocabulary 3D Scene Graphs for Perception and Planning, Gu et. al., 2024

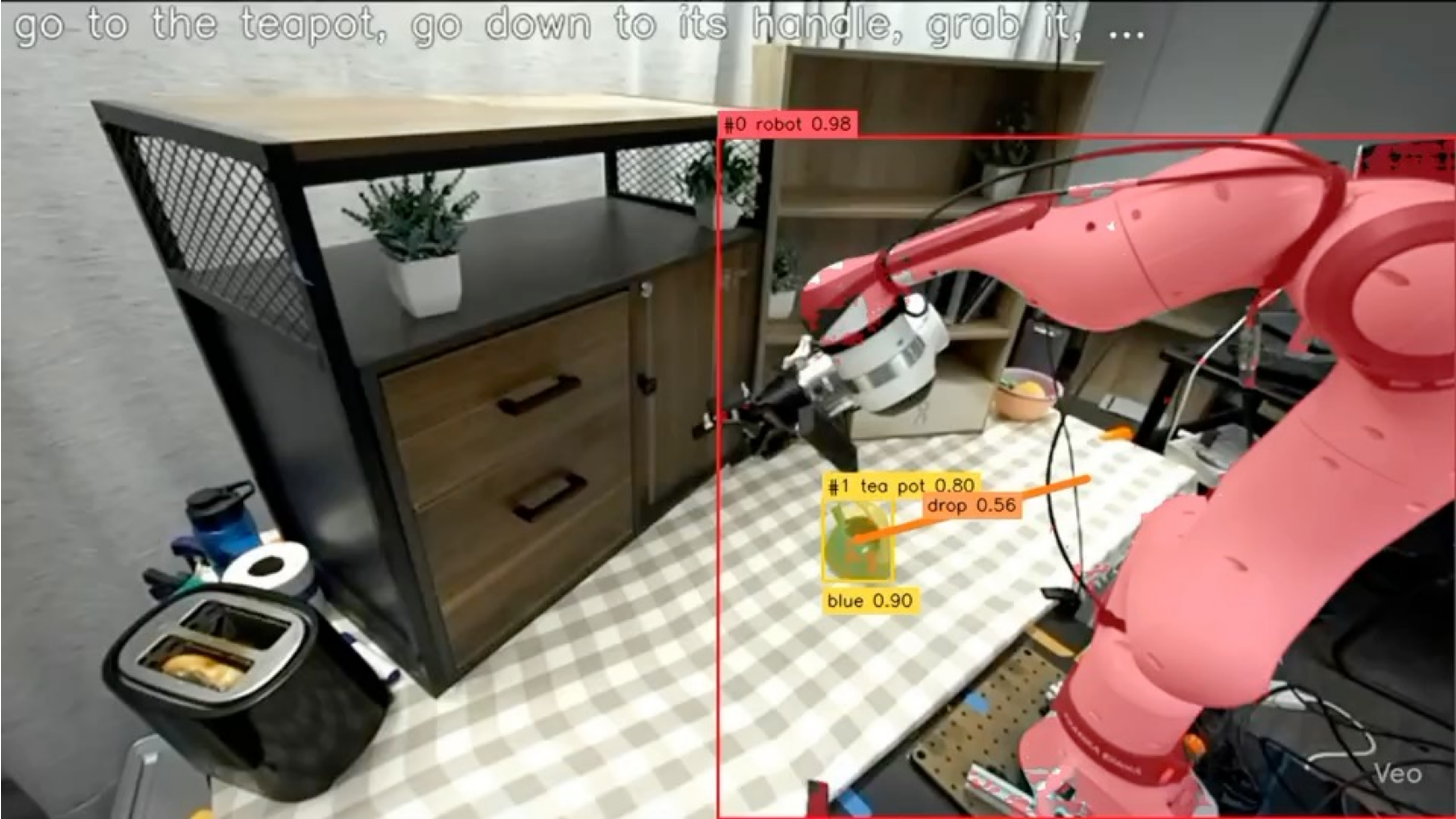
# Spatio-Temporal Scene Graphs: Applications

```
behavior achieve_at_3(g: Pose):
  goal: at(g)
  body:
    bind x: Pose
    bind t: Trajectory where:
      connect(x, g, t)
    promotable:
      achieve at(x)
      forall o: Object:
        achieve not blocking(o, t)
    do exec_traj(t)
```

```
(:action pick-up
  :parameters (?p-paper)
  :precondition (and
    (at ?l)
    (isHomeBase ?l)
    (unpacked ?p))
  :effect (and
    (not (unpacked ?p))
    (carrying ?p)))
```

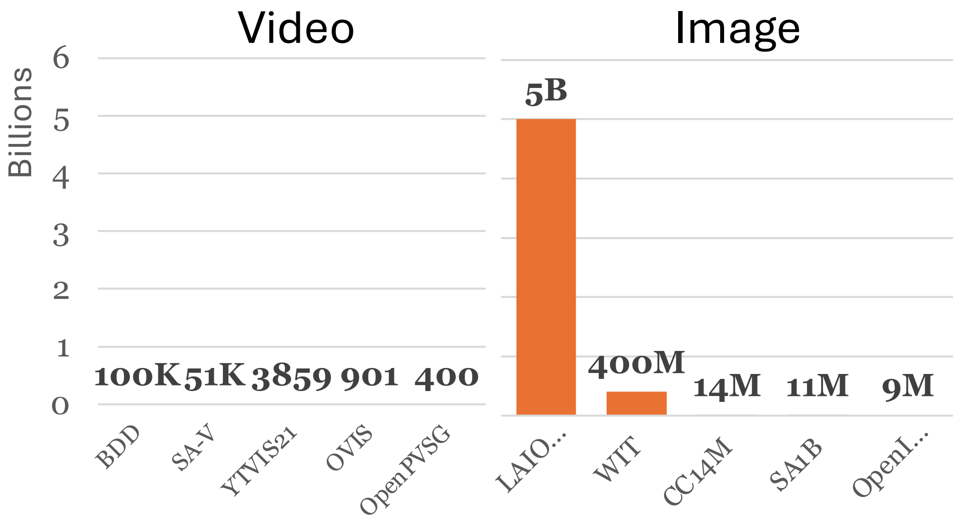
Being able to programmatically plan
requires structured scene understanding
such as "`blocking(o, t)`".

Hybrid Declarative-Imperative Representations for Hybrid Discrete-Continuous Decision-Making, Mao et. al., 2024

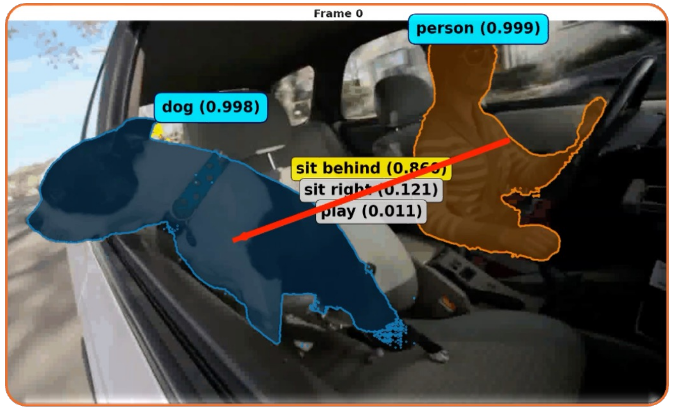

go to the teapot, go down to its handle, grab it, ...
#0 robot 0.98
#1 tea pot 0.80
drop 0.56
blue 0.90
Veo

# Spatio-Temporal Scene Graphs: Challenges

Unknown Granularity




Scarcity of Dataset




Open-Vocabulary

# Neurosymbolic to the rescue!

- Scene graph is a symbolic data-structure
  - Can be regularized by specifications
  - Symbolic algorithm can align scene graphs and specifications
- Specifications can be extracted from natural scene description
  - Vision-language models can generate natural language captions
  - Captions can be semantically parsed into
- Neurosymbolic framework can facilitate training
  - Scallop programming language is differentiable
  - Allowing to program alignment checker for neurosymbolic learning

# Agenda

## Scallop, A Language for Neurosymbolic Programming

Ziyang Li, Jiani Huang, Mayur Naik
[PLDI 2023]

## LASER: A Neurosymbolic Framework For Learning Spatio-temporal Scene Graphs with Weak Supervision

Jiani Huang, Ziyang Li, Ser-Nam Lim, Mayur Naik
[ICLR 2025]

## ESCA: Contextualizing Embodied Agents via Scene-Graph Generation

Jiani Huang, Ziyang Li, Matthew Kuo, Amish Sethi, Neelay Velingker, Mayank Keoliya, Ser-Nam Lim, Mayur Naik
[NeurIPS 2025]

# Agenda

Scallop, A Language for Neurosymbolic Programming

Ziyang Li, Jiani Huang, Mayur Naik
[PLDI 2023]

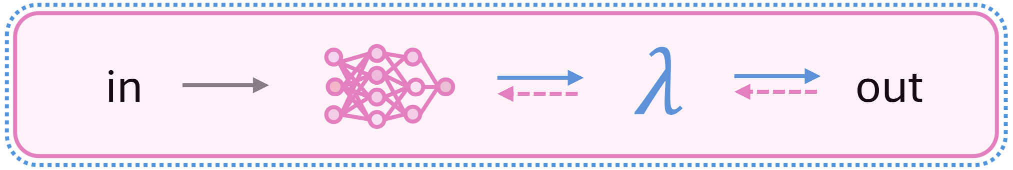
LASER: A Neurosymbolic Framework For Learning Spatio-temporal Scene Graphs with Weak Supervision

Jiani Huang, Ziyang Li, Ser-Nam Lim, Mayur Naik
[ICLR 2025]

ESCA: Contextualizing Embodied Agents via Scene-Graph Generation

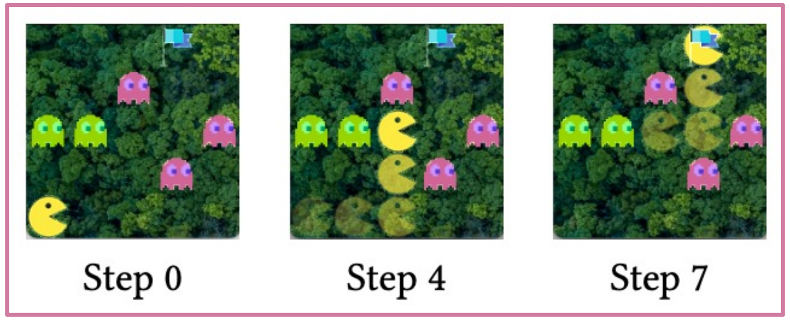
Jiani Huang, Ziyang Li, Matthew Kuo, Amish Sethi, Neelay Velingker, Mayank Keoliya, Ser-Nam Lim, Mayur Naik
[NeurIPS 2025]

# Scallop: A Language for Neurosymbolic Programming



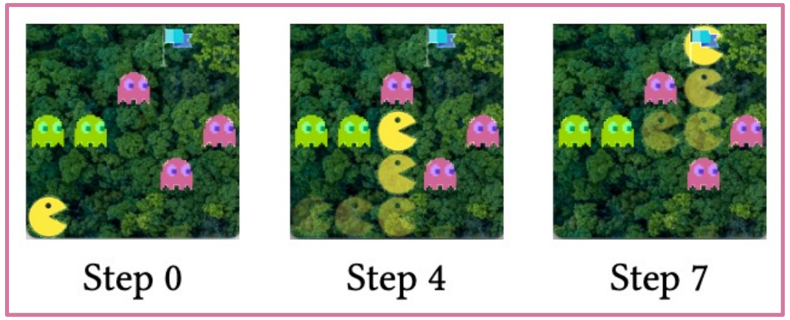

[Algorithmic Supervision]

# Illustrative Example: PacMan-Maze

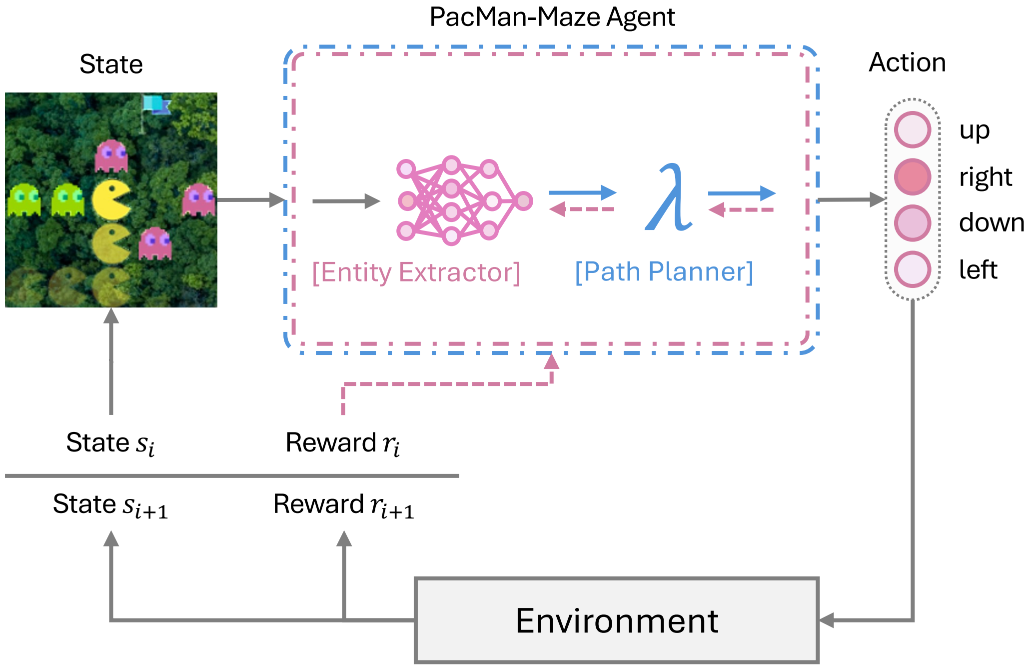



**Environment**: a 5x5 grid of PacMan Arena
**State**: 200x200 colored image
**Action**: Up/Down/Left/Right
**Goal**: Reach the flag without touching enemy
(Environments are randomly generated)

# Illustrative Example: PacMan-Maze




**Environment**: a 5x5 grid of PacMan Arena
**State**: 200x200 colored image
**Action**: Up/Down/Left/Right
**Goal**: Reach the flag without touching enemy
(Environments are randomly generated)



# Illustrative

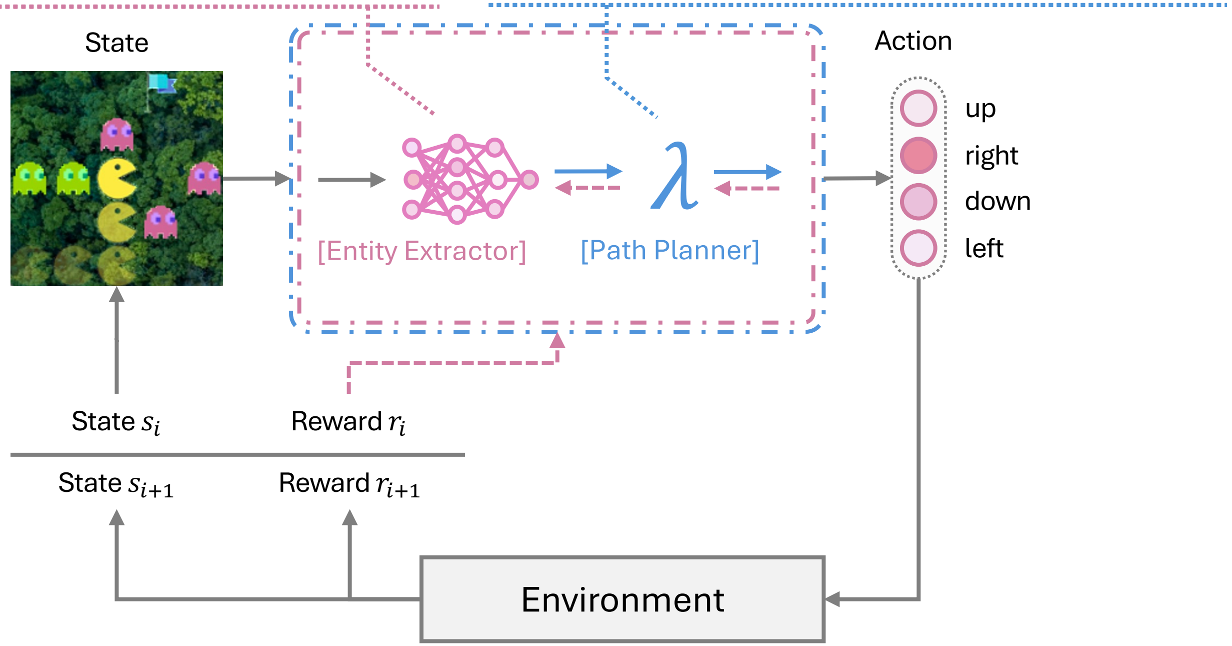
## Entity Extractor

Simple Convolutional Neural Network (CNN) based model. Initialized randomly and needs to be trained to recognize entities.

## Path Planner

Given the entities, plan the best path to reach the goal. Planner also needs to propagate learning signal to train the entity extractor.




**Environment:** a 5x5 grid of PacMan Arena
**State:** 200x200 colored image
**Action:** Up/Down/Left/Right
**Goal:** Reach the flag without touching enemy
(Environments are randomly generated)

# PacMan-Maze: Results

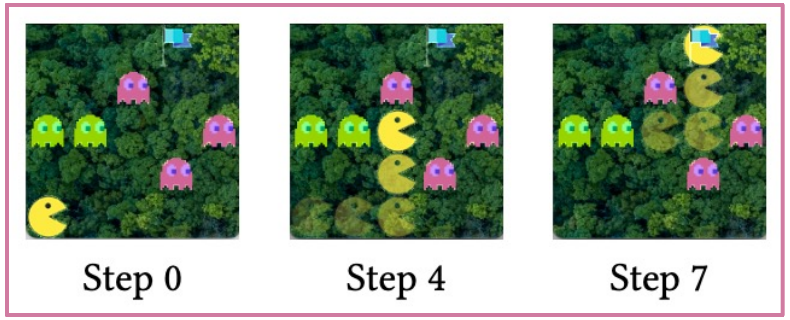
Step 0 Step 4 Step 7

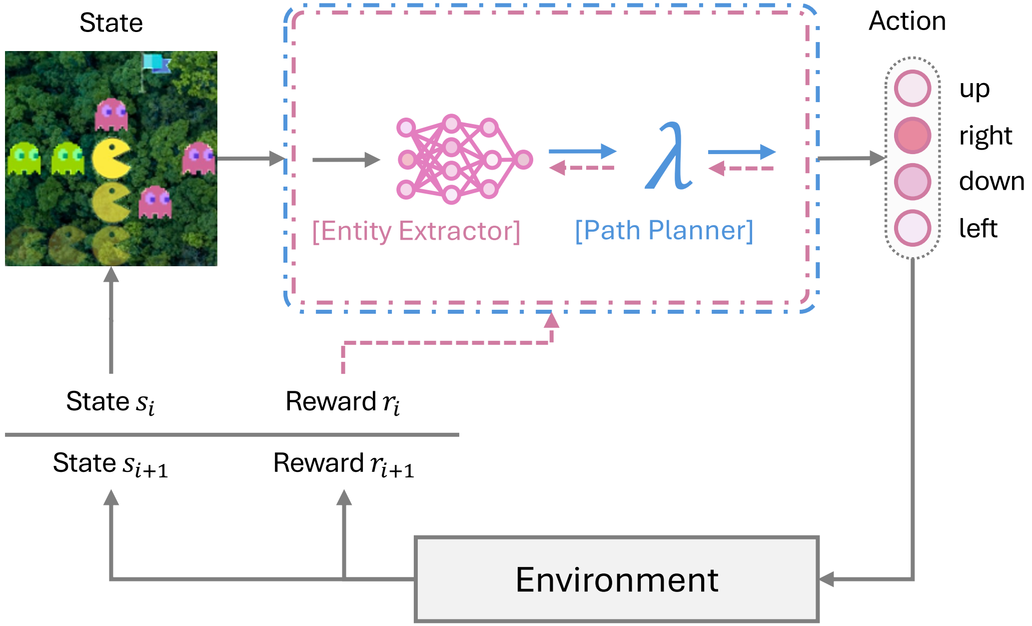
**Environment**: a 5x5 grid of PacMan Arena
**State**: 200x200 colored image
**Action**: Up/Down/Left/Right
**Goal**: Reach the flag without touching enemy
(Environments are randomly generated)

| Method | Success Rate / #Training Episodes |
| --- | --- |
| in → → out [Deep-Q-Network] | 84.9% / 50,000 |
| in → λ out [Scallop+CNN Q-Learning] | **99.4% / 50** |

Success rate: whether the flag is reached within 50 steps.
#Training Episodes: # of training episodes needed to achieve the success rate.

# Illustrative Example: PacMan-Maze

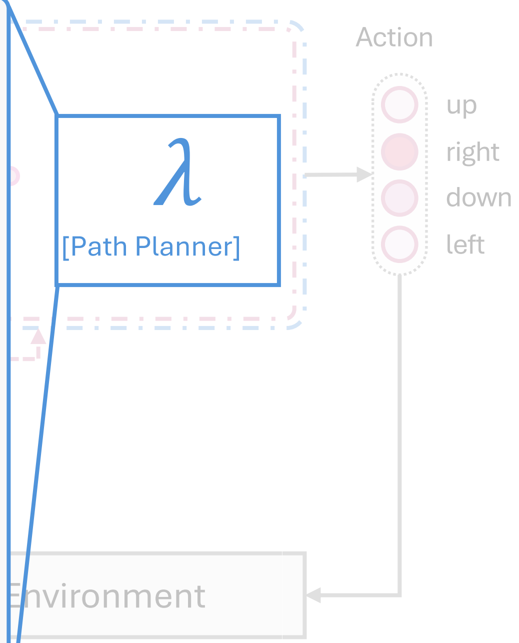



**Environment**: a 5x5 grid of PacMan Arena
**State**: 200x200 colored image
**Action**: Up/Down/Left/Right
**Goal**: Reach the flag without touching enemy
(Environments are randomly generated)

# Illustrative Example: PacMan-Maze

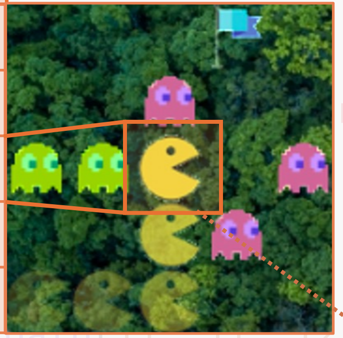
```
// Inputs from the Entity Extractor (Neural Network)
type pacman(i: i32, j: i32), flag(i: i32, j: i32), ghost(i: i32, j: i32)

// Find safe cells and safe edges
rel safe_cell(i, j) = grid_cell(i, j) and not ghost(i, j)
rel edge(i, j, i, j + 1,UP) = safe_cell(i, j) and safe_cell(i, j + 1) //...

// Find safe paths recursively
rel path(i, j, i, j) = safe_cell(i, j)
rel path(i1, j1, i3, j3) = path(i1, j1, i2, j2) and edge(i2, j2, i3, j3, _)

// Compute the action that can lead to the goal
rel take_action(a) = pacman(i, j) and edge(i, j, m, n, a) and
                     path(m, n, g, h) and flag(g, h)
```



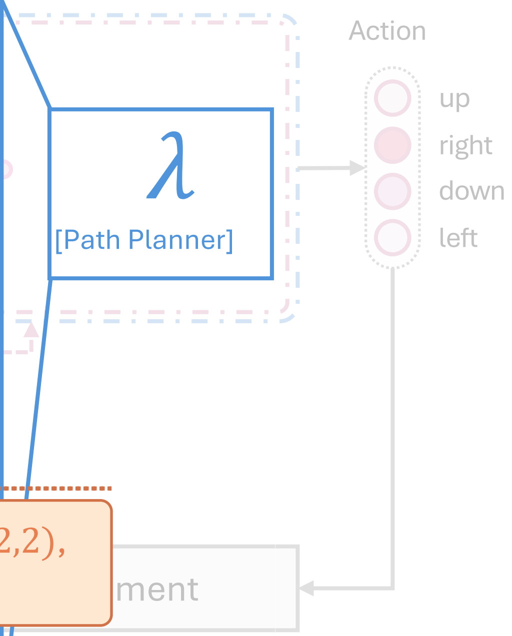
# Illustrative Example: PacMan-Maze

```
// Inputs from the Entity Extractor (Neural Network)
type pacman(i: i32, j: i32), flag(i: i32, j: i32), ghost(i: i32, j: i32)

// F
rel s                           not ghost(i, j)
rel                           j) and safe_cell(i, j + 1) //...

// F
rel p
rel path(i1, j1, i3, j3) = path(i1, j1, i2, j2) and edge(i2, j2, i3, j3, _)

// Compute the action that can lea
rel take_action(a) = pacman(i, j)
                     path(m, n, g, h) and flag(g, h)
```

| $\mathbf{Pr}(\text{pacman}_{ij} \mid \boldsymbol{\theta})$ | i | j |
|---|---|---|
| 0.00 | 2 | 1 |
| **0.92** | **2** | **2** |
| 0.02 | 3 | 4 |
| … | … | … |

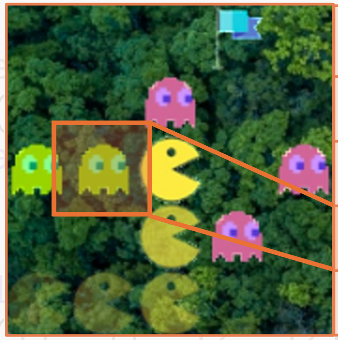


Estimates the probability of PacMan being at the location (2,2), given the neural network parameters $\theta$.

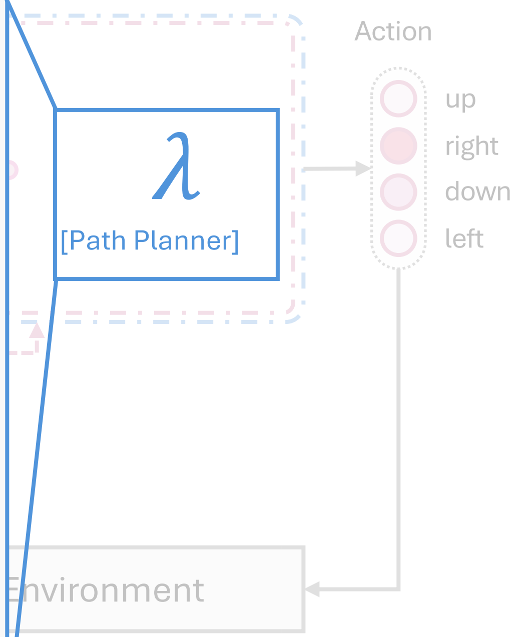


# Illustrative Example: PacMan-Maze

```
// Inputs from the Entity Extractor (Neural Network)
type pacman(i: i32, j: i32), flag(i: i32, j: i32), ghost(i: i32, j: i32)

// Find safe cells and safe edge
rel safe_cell(i, j) = grid_cell(
rel edge(i, j, i, j + 1,UP) = sa

// Find safe paths recursively
rel path(i, j, i, j) = safe_cell
rel path(i1, j1, i3, j3) = path(i1, j1, i2, j2) and edge(i2, j2, i3, j3, _)

// Compute the action that can lead to the goal
rel take_action(a) = pacman(i, j) and edge(i, j, m, n, a) and
                     path(m, n, g, h) and flag(g, h)
```



| $\Pr(\text{ghost}_{ij} \mid \theta)$ | i | j |
|---|---|---|
| 0.00 | 0 | 0 |
| 0.94 | 1 | 2 |
| **0.85** | **2** | **1** |
| ... | ... | ... |

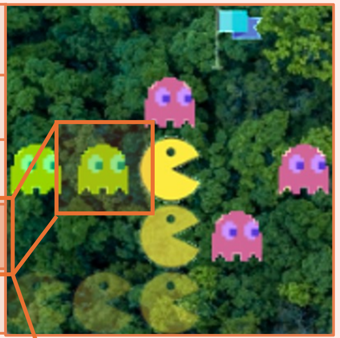


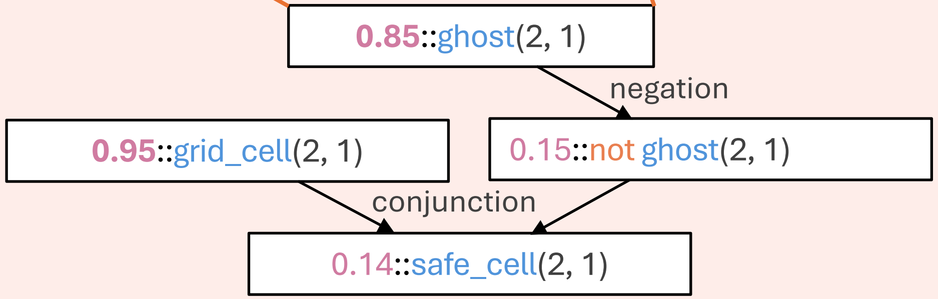
# Illustrative Example: PacMan-Maze

```
// Inputs from the Entity Extractor (Neural Network)
type pacman(i: i32, j: i32), flag(i: i32, j: i32), ghost(i: i3

// Find safe cells and safe edges
rel safe_cell(i, j) = grid_cell(i, j) and not ghost(i, j)
rel edge(i, j, i, j + 1,UP) = safe_cell(i, j) and safe_cell(i,

// Find safe paths recursively
rel path(i, j, i, j) = safe_cell(i, j)
rel path(i1, j1, i3, j3) = path(i1, j1, i2, j2) and edge(i2, j

// Compute the action that can lead to the goal
rel take_action(a) = pacman(i, j) and edge(i, j, m, n, a) and
                     path(m, n, g, h) and flag(g, h)
```

## Illustration of Probabilistic Reasoning

| $\mathbf{Pr}(\text{ghost}_{ij} \| \boldsymbol{\theta})$ | i | j |
|---|---|---|
| 0.00 | 0 | 0 |
| 0.94 | 1 | 2 |
| **0.85** | **2** | **1** |
| … | … | … |





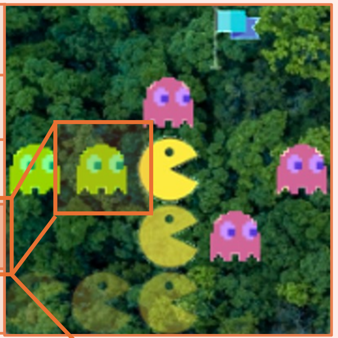
# Illustrative Example: PacMan-Maze

```
// Inputs from the Entity Extractor (Neural Network)
type pacman(i: i32, j: i32), flag(i: i32, j: i32), ghost(i: i3

// Find safe cells and safe edges
rel safe_cell(i, j) = grid_cell(i, j) and not ghost(i, j)
rel edge(i, j, i, j + 1,UP) = safe_cell(i, j) and safe_cell(i,

// Find safe paths
rel path(i, j, i,
rel path(i1, j1, i

// Compute the action that can lead to the goal
rel take_action(a) = pacman(i, j) and edge(i, j, m, n, a) and
                     path(m, n, g, h) and flag(g, h)
```

$\frac{\partial g_{21}}{\partial}$ The gradient carries **partial derivative** with respect to neural-network parameters!

## Illustration of **Differentiable** Reasoning

| $\mathbf{Pr}(\text{ghost}_{ij} \mid \boldsymbol{\theta})$ | i | j |
|---|---|---|
| $\langle 0.00, \nabla g_{00} \rangle$ | 0 | 0 |
| $\langle 0.94, \nabla g_{12} \rangle$ | 1 | 2 |
| $\langle 0.85, \nabla g_{21} \rangle$ | **2** | **1** |
| … | … | … |



$\langle 0.85, \nabla g_{21} \rangle$ ::ghost(2, 1)

negation

$\langle 0.15, -\nabla g_{21} \rangle$ ::not ghost(2, 1)

$\langle 0.95, \vec{0} \rangle$ ::grid_cell(2, 1)

conjunction

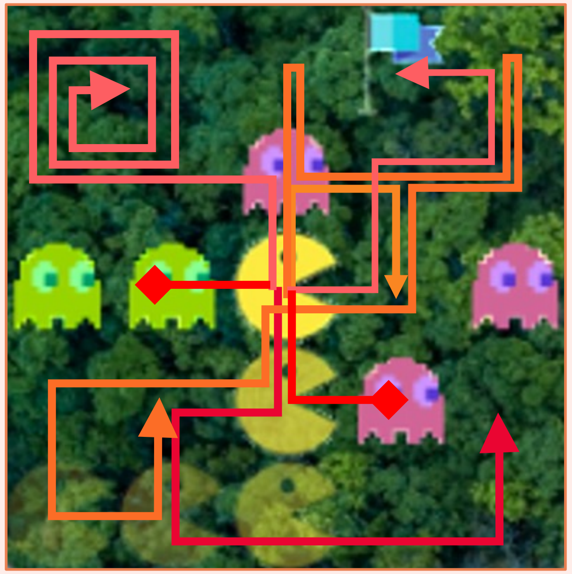
$\langle 0.14, -0.95 \cdot \nabla g_{21} \rangle$ ::safe_cell(2, 1)

# Illustrative Example: PacMan-Maz

**Recursively** building safe paths by appending safe edges to existing safe paths.

```
// Inputs from the Entity Extractor (Neural Network)
type pacman(i: i32, j: i32)

// Find safe cells and safe edge
rel safe_cell(i, j) = grid_cell(i, j) and not ghost(i, j)
rel edge(i, j, i, j + 1,UP) = safe_cell(i, j) and safe_cell(i, j + 1) //...

// Find safe paths recursively
rel path(i, j, i, j) = safe_cell(i, j)
rel path(i1, j1, i3, j3) = path(i1, j1, i2, j2) and edge(i2, j2, i3, j3, _)

// Compute the action that can lead to the goal
rel take_action(a) = pacman(i, j) and edge(i, j, m, n, a) and
                     path(m, n, g, h) and flag(g, h)
```

What would happen when reasoning becomes more **complicated**?



Non-Termination

Complexity of Exact Probability Estimation

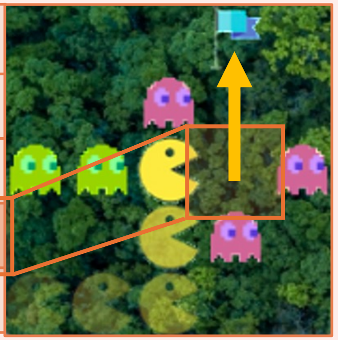
# Illustrative Exam

| | | | |
|---|---|---|---|
| (Tag Space) | $t$ | $\in$ | $T$ |
| (False) | $\mathbf{0}$ | $\in$ | $T$ |
| (True) | $\mathbf{1}$ | $\in$ | $T$ |
| (Disjunction) | $\oplus$ | : | $T \times T \rightarrow T$ |
| (Conjunction) | $\otimes$ | : | $T \times T \rightarrow T$ |
| (Negation) | $\ominus$ | : | $T \rightharpoonup T$ |
| (Saturation) | $\ominus$ | : | $T \times T \rightarrow \text{Bool}$ |

```
// Inputs from the En
type pacman(i: i32, j

// Find safe cells an
rel safe_cell(i, j) = grid_cell(i, j) and not gho
rel edge(i, j, i, j + 1,UP) = safe_cell(i, j) and

// Find safe paths recursively
rel path(i, j, i, j) = safe_cell(i, j)
rel path(i1, j1, i3, j3) = path(i1, j1, i2, j2) an

// Compute the action that can lead to the goal
rel take_action(a) = pacman(i, j) and edge(i, j,
                     path(m, n, g, h) and flag(g,
```

## Illustration of Abstracted Tagged-Semantics

| **Tag:** $T$ | **i** | **j** |
|---|---|---|
| $\text{safe}_{00}$ | 0 | 0 |
| $\text{safe}_{13}$ | 1 | 3 |
| $\textbf{safe}_{23}$ | **2** | **3** |
| ... | ... | ... |



...

$\ominus$

$\text{safe}_{23}$::safe_cell(2, 3)

$\otimes$

$\text{safe}_{23}$::path(2, 3, 2, 3)

...

$\text{safe}_{23} \otimes \text{safe}_{13}$::edge(2, 3, 1, 3)

...

$\oplus \otimes \ominus$

$\text{safe}_{23} \otimes \text{safe}_{13} \otimes \text{safe}_{03}$::path(2, 3, 0, 3)

# Built-in Library of Provenance Structures

| Kind | Provenance | $T$ | $\mathbf{0}$ | $\mathbf{1}$ | $\oplus$ | $\otimes$ | $\ominus$ | $\circleddash$ | $\tau$ | $\rho$ |
|---|---|---|---|---|---|---|---|---|---|---|
| Discrete | `unit` | $\{()\}$ | $()$ | $()$ | $\lambda t_1, t_2.()$ | $\lambda t_1, t_2.()$ | $\lambda a.\text{FAIL}$ | $==$ | $\lambda i.()$ | $\lambda t.()$ |
| | `bool` | $\{\top, \bot\}$ | $\bot$ | $\top$ | $\vee$ | $\wedge$ | $\neg$ | $==$ | $id$ | $id$ |
| | `natural` | $\mathbb{N}$ | $0$ | $1$ | $+$ | $\times$ | | | | |
| Probabilistic | `max-min-prob` | $[0,1]$ | $0$ | $1$ | max | min | | | | |
| | `add-mult-prob` | $[0,1]$ | $0$ | $1$ | $\lambda t_1, t_2.\text{clamp}(t_1 + t_2)$ | $\lambda t_1, t_2.(t_1 \cdot$ | | | | |
| | `nand-min-prob` | $[0,1]$ | $0$ | $1$ | $\lambda t_1, t_2. -(1 - t_1)(1 - t_2)$ | min | | | | |
| | `nand-mult-prob` | $[0,1]$ | $0$ | $1$ | $\lambda t_1, t_2. -(1 - t_1)(1 - t_2)$ | $\lambda t_1, t_2.t_1 \cdot$ | | | | |
| | `top-k-proofs` | $\Phi$ | $\emptyset$ | $\{\emptyset\}$ | $\vee_{\text{top-}k}$ | $\wedge_{\text{top-}k}$ | | | | ) |
| | `sample-k-proofs` | $\Phi$ | $\emptyset$ | $\{\emptyset\}$ | $\vee_{\text{sample-}k}$ | $\wedge_{\text{sample-}k}$ | | | | ) |
| Differentiable | `diff-max-min-prob` | $\mathbb{D}$ | $\hat{0}$ | $\hat{1}$ | max | min | | | | |
| | `diff-add-mult-prob` | $\mathbb{D}$ | $\hat{0}$ | $\hat{1}$ | $\lambda \hat{t}_1, \hat{t}_2.\text{clamp}(\hat{t}_1 + \hat{t}_2)$ | $\lambda \hat{t}_1, \hat{t}_2.\hat{t}_1 \cdot$ | | | | |
| | `diff-nand-min-prob` | $[\hat{0}, \hat{1}]$ | $\hat{0}$ | $\hat{1}$ | $\lambda \hat{t}_1, \hat{t}_2. -(\hat{1} - \hat{t}_1)(\hat{1} - \hat{t}_2)$ | min | | | | |
| | `diffnand-mult-prob` | $[\hat{0}, \hat{1}]$ | $\hat{0}$ | $\hat{1}$ | $\lambda \hat{t}_1, \hat{t}_2. -(\hat{1} - \hat{t}_1)(\hat{1} - \hat{t}_2)$ | $\lambda \hat{t}_1, \hat{t}_2.\hat{t}_1 \cdot \hat{t}_2$ | $\lambda t.1 - t$ | $\lambda t.1$ | $id$ | $id$ |
| | `diff-top-k-proofs` | $\Phi$ | $\emptyset$ | $\{\emptyset\}$ | $\vee_{\text{top-}k}$ | $\wedge_{\text{top-}k}$ | $\neg_{\text{top-}k}$ | $==$ | $\lambda \hat{p}_i.\{\{\text{pos}(i)\}\}$ | $\lambda \varphi.\text{WMC}(\varphi, \hat{\Gamma})$ |
| | `diff-sample-k-proofs` | $\Phi$ | $\emptyset$ | $\{\emptyset\}$ | $\vee_{\text{sample-}k}$ | $\wedge_{\text{sample-}k}$ | $\neg_{\text{sample-}k}$ | $==$ | $\lambda \hat{p}_i.\{\{\text{pos}(i)\}\}$ | $\lambda \varphi.\text{WMC}(\varphi, \hat{\Gamma})$ |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |

$$
\begin{array}{rccl}
\text{(Tag Space)} & t & \in & T \\
\text{(False)} & \mathbf{0} & \in & T \\
\text{(True)} & \mathbf{1} & \in & T \\
\text{(Disjunction)} & \oplus & : & T \times T \to T \\
\text{(Conjunction)} & \otimes & : & T \times T \to T \\
\text{(Negation)} & \ominus & : & T \rightharpoonup T \\
\text{(Saturation)} & \circleddash & : & T \times T \to \text{Bool}
\end{array}
$$

Inspired by *Provenance Semiring* (Green et. al. PODS 2007)

# Approximated Probabilistic Provenance

## Top-k Proofs

$T_{tkp}$: Boolean Formula in DNF

$\otimes_{tkp}$:
$\oplus_{tkp}$:
$\ominus_{tkp}$:

Perform the logical "conjunction", "disjunction", and "negation", while preserving only the $k$ most significant conjunctive clauses.

## Top-Bottom-k Clauses

$T_{tbkc}$: Boolean Formula in DNF or CNF

$\otimes_{tbkc}$:
$\oplus_{tbkc}$:
$\ominus_{tbkc}$:

Perform the logical "conjunction", "disjunction", and "negation". For DNF, keep the top-$k$ conjunctive clauses, while for CNF, keep the bottom-$k$ disjunctive clauses.

## Optimal-k Proofs

$T_{okp}$: Boolean Formula in DNF

$\otimes_{okp}$:
$\oplus_{okp}$:
$\ominus_{okp}$:

Employs a greedy algorithm to preserve the $k$ conjunctive clauses that has a joint probability the closest to the exact probability.

Inspired by [Renkens et. al. 2012]

Performs Weighted Model Counting (WMC) to recover probabilities from Boolean formula $\phi_y$.

$$\Pr(y \mid \theta) = \mathrm{WMC}(\phi_y, \{x \mapsto \Pr(x \mid \theta)\})$$

| | | | |
|---|---|---|---|
| (Tag Space) | $t$ | $\in$ | $T$ |
| (False) | $\mathbf{0}$ | $\in$ | $T$ |
| (True) | $\mathbf{1}$ | $\in$ | $T$ |
| (Disjunction) | $\oplus$ | : | $T \times T \to T$ |
| (Conjunction) | $\otimes$ | : | $T \times T \to T$ |
| (Negation) | $\ominus$ | : | $T \rightharpoonup T$ |
| (Saturation) | $\circleddash$ | : | $T \times T \to \mathrm{Bool}$ |

# Approximated Differentiable Provenance

## Diff. Top-k Proofs

$T_{tkp}$: Boolean Formula in DNF

$\otimes_{tkp}$:, $\oplus_{tkp}$:, $\ominus_{tkp}$: Perform the logical "conjunction", "disjunction", and "negation", while preserving only the $k$ most significant conjunctive clauses.

## Diff. Top-Bottom-k Clauses

$T_{tbkc}$: Boolean Formula in DNF or CNF

$\otimes_{tbkc}$:, $\oplus_{tbkc}$:, $\ominus_{tbkc}$: Perform the logical "conjunction", "disjunction", and "negation". For DNF, keep the top-$k$ conjunctive clauses, while for CNF, keep the bottom-$k$ disjunctive clauses.

## Diff. Optimal-k Proofs

$T_{okp}$: Boolean Formula in DNF

$\otimes_{okp}$:, $\oplus_{okp}$:, $\ominus_{okp}$: Employs a greedy algorithm to preserve the $k$ conjunctive clauses that has a joint probability the closest to the exact probability.

Inspired by [Renkens et. al. 2012]

Performs Weighted Model Counting (WMC) to recover probabilities with gradients from Boolean formula $\phi_y$.

$$\langle \Pr(y \mid \theta), \nabla \Pr(y \mid \theta) \rangle = \mathrm{WMC}(\phi_y, \{x \mapsto \langle \Pr(x \mid \theta), \nabla \Pr(x \mid \theta) \rangle\})$$

$$\begin{array}{rccl}
\text{(Tag Space)} & t & \in & T \\
\text{(False)} & \mathbf{0} & \in & T \\
\text{(True)} & \mathbf{1} & \in & T \\
\text{(Disjunction)} & \oplus & : & T \times T \to T \\
\text{(Conjunction)} & \otimes & : & T \times T \to T \\
\text{(Negation)} & \ominus & : & T \rightharpoonup T \\
\text{(Saturation)} & \ominus & : & T \times T \to \text{Bool}
\end{array}$$

# Built-in Library of Provenance Structures

| Kind | Provenance | $T$ | $\mathbf{0}$ | $\mathbf{1}$ | $\oplus$ | $\otimes$ | $\ominus$ | $\circleddash$ | $\tau$ | $\rho$ |
|---|---|---|---|---|---|---|---|---|---|---|
| Discrete | `unit` | $\{()\}$ | $()$ | $()$ | $\lambda t_1, t_2.()$ | $\lambda t_1, t_2.()$ | $\lambda a.\text{FAIL}$ | $==$ | $\lambda i.()$ | $\lambda t.()$ |
| | `bool` | $\{\top, \bot\}$ | $\bot$ | $\top$ | $\vee$ | $\wedge$ | $\neg$ | $==$ | $id$ | $id$ |
| | `natural` | $\mathbb{N}$ | $0$ | $1$ | $+$ | $\times$ | | | | |
| Probabilistic | `max-min-prob` | $[0,1]$ | $0$ | $1$ | max | min | | | | |
| | `add-mult-prob` | $[0,1]$ | $0$ | $1$ | $\lambda t_1, t_2.\text{clamp}(t_1 + t_2)$ | $\lambda t_1, t_2.(t_1 \cdot$ | | | | |
| | `nand-min-prob` | $[0,1]$ | $0$ | $1$ | $\lambda t_1, t_2. -(1 - t_1)(1 - t_2)$ | min | | | | |
| | `nand-mult-prob` | $[0,1]$ | $0$ | $1$ | $\lambda t_1, t_2. -(1 - t_1)(1 - t_2)$ | $\lambda t_1, t_2.t_1 \cdot$ | | | | |
| | `top-k-proofs` | $\Phi$ | $\emptyset$ | $\{\emptyset\}$ | $\vee_{\text{top-}k}$ | $\wedge_{\text{top-}k}$ | | | | |
| | `sample-k-proofs` | $\Phi$ | $\emptyset$ | $\{\emptyset\}$ | $\vee_{\text{sample-}k}$ | $\wedge_{\text{sample-}k}$ | | | | |
| Differentiable | `diff-max-min-prob` | $\mathbb{D}$ | $\hat{0}$ | $\hat{1}$ | max | min | | | | |
| | `diff-add-mult-prob` | $\mathbb{D}$ | $\hat{0}$ | $\hat{1}$ | $\lambda \hat{t}_1, \hat{t}_2.\text{clamp}(\hat{t}_1 + \hat{t}_2)$ | $\lambda \hat{t}_1, \hat{t}_2.\hat{t}_1 \cdot$ | | | | |
| | `diff-nand-min-prob` | $[\hat{0}, \hat{1}]$ | $\hat{0}$ | $\hat{1}$ | $\lambda \hat{t}_1, \hat{t}_2. -(\hat{1} - \hat{t}_1)(\hat{1} - \hat{t}_2)$ | min | | | | |
| | `diffnand-mult-prob` | $[\hat{0}, \hat{1}]$ | $\hat{0}$ | $\hat{1}$ | $\lambda \hat{t}_1, \hat{t}_2. -(\hat{1} - \hat{t}_1)(\hat{1} - \hat{t}_2)$ | $\lambda \hat{t}_1, \hat{t}_2.\hat{t}_1 \cdot \hat{t}_2$ | | | | |
| | `diff-top-k-proofs` | $\Phi$ | $\emptyset$ | $\{\emptyset\}$ | $\vee_{\text{top-}k}$ | $\wedge_{\text{top-}k}$ | $\neg_{\text{top-}k}$ | $==$ | $\lambda \hat{p}_i.\{\{\text{pos}(i)\}\}$ | $\lambda \varphi.\text{WMC}(\varphi, \hat{\Gamma})$ |
| | `diff-sample-k-proofs` | $\Phi$ | $\emptyset$ | $\{\emptyset\}$ | $\vee_{\text{sample-}k}$ | $\wedge_{\text{sample-}k}$ | $\neg_{\text{sample-}k}$ | $==$ | $\lambda \hat{p}_i.\{\{\text{pos}(i)\}\}$ | $\lambda \varphi.\text{WMC}(\varphi, \hat{\Gamma})$ |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |

$$
\begin{array}{rccl}
\text{(Tag Space)} & t & \in & T \\
\text{(False)} & \mathbf{0} & \in & T \\
\text{(True)} & \mathbf{1} & \in & T \\
\text{(Disjunction)} & \oplus & : & T \times T \to T \\
\text{(Conjunction)} & \otimes & : & T \times T \to T \\
\text{(Negation)} & \ominus & : & T \rightharpoonup T \\
\text{(Saturation)} & \circleddash & : & T \times T \to \text{Bool}
\end{array}
$$

Inspired by *Provenance Semiring* (Green et. al. PODS 2007)

# Built-in Library of Provenance Structures

| Kind | Provenance | $T$ | **0** | **1** | $\oplus$ | $\otimes$ | $\ominus$ | $\oplus$ | $\tau$ | $\rho$ |
|---|---|---|---|---|---|---|---|---|---|---|
| | unit | $\{()\}$ | $()$ | $()$ | $\lambda t_1, t_2.()$ | $\lambda t_1, t_2.()$ | $\lambda a.\text{FAIL}$ | $==$ | $\lambda i.()$ | $\lambda t.()$ |
| Discrete | bool | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | $\neg$ | $==$ | $id$ | $id$ |
| | natura[illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | $\lambda n.\mathbb{1}[n > 0]$ | $==$ | $id$ | $id$ |
| | max-min-p[illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | $\lambda t.1 - t$ | $==$ | $id$ | $id$ |
| | add-mult-p[illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | $\lambda t.1 - t$ | $\lambda t.\top$ | $id$ | $id$ |
| Probabilistic | nand-min-prob | $[0, 1]$ | $0$ | [illegible] | $\lambda t_1, t_2. -(1 - t_1)(1 - t_2)$ | min | $\lambda t.1 - t$ | $\lambda t.\top$ | $id$ | $id$ |
| | nand-mult-prob | $[0, 1]$ | $0$ | $1$ | $\lambda t_1, t_2. -(1 - t_1)(1 - t_2)$ | $\lambda t_1, t_2.t_1 \cdot t_2$ | $\lambda t.1 - t$ | $\lambda t.\top$ | $id$ | $id$ |
| | top-k-proofs | $\Phi$ | $\emptyset$ | $\{\emptyset\}$ | $\vee_{\text{top-}k}$ | $\wedge_{\text{top-}k}$ | $\neg_{\text{top-}k}$ | $==$ | $\lambda p_i.\{\{\text{pos}(i)\}\}$ | $\lambda \varphi.\text{WMC}(\varphi, \Gamma)$ |
| | sample-k-proofs | $\Phi$ | $\emptyset$ | $\{\emptyset\}$ | $\vee_{\text{sample-}k}$ | $\wedge_{\text{sample-}k}$ | $\neg_{\text{sample-}k}$ | $==$ | $\lambda p_i.\{\{\text{pos}(i)\}\}$ | $\lambda \varphi.\text{WMC}(\varphi, \Gamma)$ |
| | diff-max-min-prob | $\mathbb{D}$ | $\hat{0}$ | $\hat{1}$ | max | min | $\lambda \hat{t}.\hat{1} - \hat{t}$ | $==$ | $id$ | $id$ |
| | diff-add-mult-prob | $\mathbb{D}$ | $\hat{0}$ | $\hat{1}$ | $\lambda \hat{t}_1, \hat{t}_2.\text{clamp}(\hat{t}_1 + \hat{t}_2)$ | $\lambda \hat{t}_1, \hat{t}_2.\hat{t}_1 \cdot \hat{t}_2$ | $\lambda \hat{t}.\hat{1} - \hat{t}$ | $\lambda \hat{t}.\top$ | $id$ | $id$ |
| Differentiable | diff-nand-min-prob | $[\hat{0}, \hat{1}]$ | $\hat{0}$ | $\hat{1}$ | $\lambda \hat{t}_1, \hat{t}_2. -(\hat{1} - \hat{t}_1)(\hat{1} - \hat{t}_2)$ | min | $\lambda \hat{t}.\hat{1} - \hat{t}$ | $\lambda \hat{t}.\top$ | $id$ | $id$ |
| | diffnand-mult-prob | $[\hat{0}, \hat{1}]$ | $\hat{0}$ | $\hat{1}$ | $\lambda \hat{t}_1, \hat{t}_2. -(\hat{1} - \hat{t}_1)(\hat{1} - \hat{t}_2)$ | $\lambda \hat{t}_1, \hat{t}_2.\hat{t}_1 \cdot \hat{t}_2$ | $\lambda \hat{t}.\hat{1} - \hat{t}$ | $\lambda \hat{t}.\top$ | $id$ | $id$ |
| | diff-top-k-proofs | $\Phi$ | $\emptyset$ | $\{\emptyset\}$ | $\vee_{\text{top-}k}$ | $\wedge_{\text{top-}k}$ | $\neg_{\text{top-}k}$ | $==$ | $\lambda \hat{p}_i.\{\{\text{pos}(i)\}\}$ | $\lambda \varphi.\text{WMC}(\varphi, \hat{\Gamma})$ |
| | diff-sample-k-proofs | $\Phi$ | $\emptyset$ | $\{\emptyset\}$ | $\vee_{\text{sample-}k}$ | $\wedge_{\text{sample-}k}$ | $\neg_{\text{sample-}k}$ | $==$ | $\lambda \hat{p}_i.\{\{\text{pos}(i)\}\}$ | $\lambda \varphi.\text{WMC}(\varphi, \Gamma)$ |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |

Employing **approximation** algorithms helps in scaling the probabilistic and differentiable reasoning modules.

Allowing extensions such as uncertainty intervals and embeddings

# Built-in Library of Provenance Structures

| Kind | Provenance | $T$ | $\mathbf{0}$ | $\mathbf{1}$ | $\oplus$ | $\otimes$ | $\ominus$ | $\doteq$ | $\tau$ | $\rho$ |
|---|---|---|---|---|---|---|---|---|---|---|
| Discrete | unit | $\{()\}$ | $()$ | $()$ | $\lambda t_1, t_2.()$ | $\lambda t_1, t_2.()$ | $\lambda a.\mathrm{FAIL}$ | == | $\lambda i.()$ | $\lambda t.()$ |
| | bool | $\{\top, \bot\}$ | $\bot$ | $\top$ | $\vee$ | $\wedge$ | $\neg$ | == | *id* | *id* |
| | natural | $\mathbb{N}$ | 0 | 1 | $+$ | $\times$ | $\lambda n.\mathbb{1}[n > 0]$ | == | *id* | *id* |
| Probabilistic | max-min-prob | $[0, 1]$ | 0 | 1 | max | min | $\lambda t.1 - t$ | == | *id* | *id* |
| | | | | | | | | | | *id* |
| | | | | | | | | | | *id* |
| | | | | | | | | | | *id* |
| | | | | | | | | | | $\lambda\varphi.\mathrm{WMC}(\varphi, \Gamma)$ |
| | | | | | | | | | | $\lambda\varphi.\mathrm{WMC}(\varphi, \Gamma)$ |
| Differentiable | | | | | | | | | | *id* |
| | diff-add-mult-prob | $\mathbb{D}$ | 0 | 1 | $\lambda t_1, t_2.\mathrm{clamp}(t_1 + t_2)$ | $\lambda t_1, t_2.t_1 \cdot t_2$ | $\lambda t.1 - t$ | $\lambda t.\top$ | *id* | *id* |
| | diff-nand-min-prob | $[\hat{0}, \hat{1}]$ | $\hat{0}$ | $\hat{1}$ | $\lambda \hat{t}_1, \hat{t}_2. -(\hat{1} - \hat{t}_1)(\hat{1} - \hat{t}_2)$ | min | $\lambda \hat{t}.\hat{1} - \hat{t}$ | $\lambda \hat{t}.\top$ | *id* | *id* |
| | diffnand-mult-prob | $[\hat{0}, \hat{1}]$ | $\hat{0}$ | $\hat{1}$ | $\lambda \hat{t}_1, \hat{t}_2. -(\hat{1} - \hat{t}_1)(\hat{1} - \hat{t}_2)$ | $\lambda \hat{t}_1, \hat{t}_2.\hat{t}_1 \cdot \hat{t}_2$ | $\lambda \hat{t}.\hat{1} - \hat{t}$ | $\lambda \hat{t}.\top$ | *id* | *id* |
| | diff-top-k-proofs | $\Phi$ | $\emptyset$ | $\{\emptyset\}$ | $\vee_{\text{top-}k}$ | $\wedge_{\text{top-}k}$ | $\neg_{\text{top-}k}$ | == | $\lambda \hat{p}_i.\{\{\mathrm{pos}(i)\}\}$ | $\lambda\varphi.\mathrm{WMC}(\varphi, \hat{\Gamma})$ |
| | diff-sample-k-proofs | $\Phi$ | $\emptyset$ | $\{\emptyset\}$ | $\vee_{\text{sample-}k}$ | $\wedge_{\text{sample-}k}$ | $\neg_{\text{sample-}k}$ | == | $\lambda \hat{p}_i.\{\{\mathrm{pos}(i)\}\}$ | $\lambda\varphi.\mathrm{WMC}(\varphi, \hat{\Gamma})$ |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |

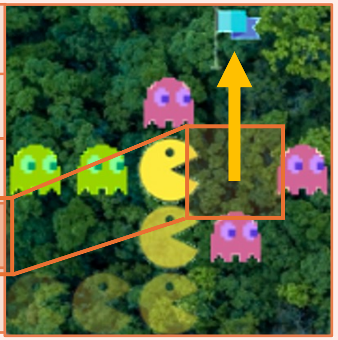
While the syntax of Scallop program remains familiar,
employing different tags allows **discrete**, **probabilistic**, and **differentiable** modes of reasoning.

# Illustrative Exam

```
// Inputs from the Entity Extractor (Neural Netwo
type pacman(i: i32, j: i32), flag(i: i32, j: i32)

// Find safe cells and safe edges
rel safe_cell(i, j) = grid_cell(i, j) and not gho
rel edge(i, j, i, j + 1,UP) = safe_cell(i, j) and

// Find safe paths recursively
rel path(i, j, i, j) = safe_cell(i, j)
rel path(i1, j1, i3, j3) = path(i1, j1, i2, j2) an

// Compute the action that can lead to the goal
rel take_action(a) = pacman(i, j) and edge(i, j,
                     path(m, n, g, h) and flag(g,
```

## Illustration of Abstracted Tagged-Semantics

| **Tag:** $T$ | i | j |
|---|---|---|
| $\text{safe}_{00}$ | 0 | 0 |
| $\text{safe}_{13}$ | 1 | 3 |
| **$\text{safe}_{23}$** | **2** | **3** |
| ... | ... | ... |



...

$\ominus$

$\text{safe}_{23}$::safe_cell(2, 3)

$\otimes$

$\text{safe}_{23}$::path(2, 3, 2, 3)

...

$\text{safe}_{23} \otimes \text{safe}_{13}$::edge(2, 3, 1, 3)

...

$\oplus\otimes\ominus$

$\text{safe}_{23} \otimes \text{safe}_{13} \otimes \text{safe}_{03}$::path(2, 3, 0, 3)

# Illustrative Example: PacM

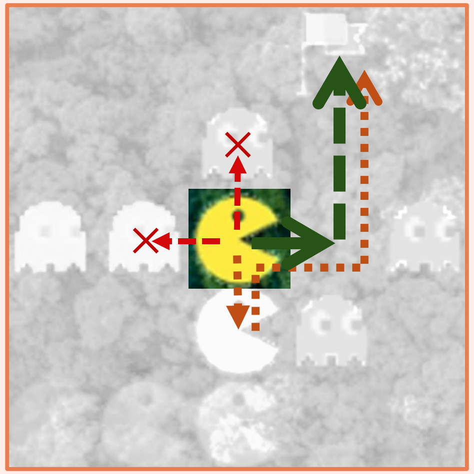
Estimates the **probability of success** after taking the action $a$, conditioned on neural network parameters $\theta$.

```
take_action(RIGHT) <- pacman(2,2) and
                      not ghost(2,3) and
                      not ghost(1,3) and
                      flag(0,3) or ...
```

$$
\begin{aligned}
\Pr(\text{success} \mid \text{action} = \text{RIGHT}) = \Pr(\text{pacman}_{22}) \times \\
(1 - \Pr(\text{ghost}_{23})) \times \\
(1 - \Pr(\text{ghost}_{13})) \times \\
\Pr(\text{flag}_{03}) + \cdots
\end{aligned}
$$

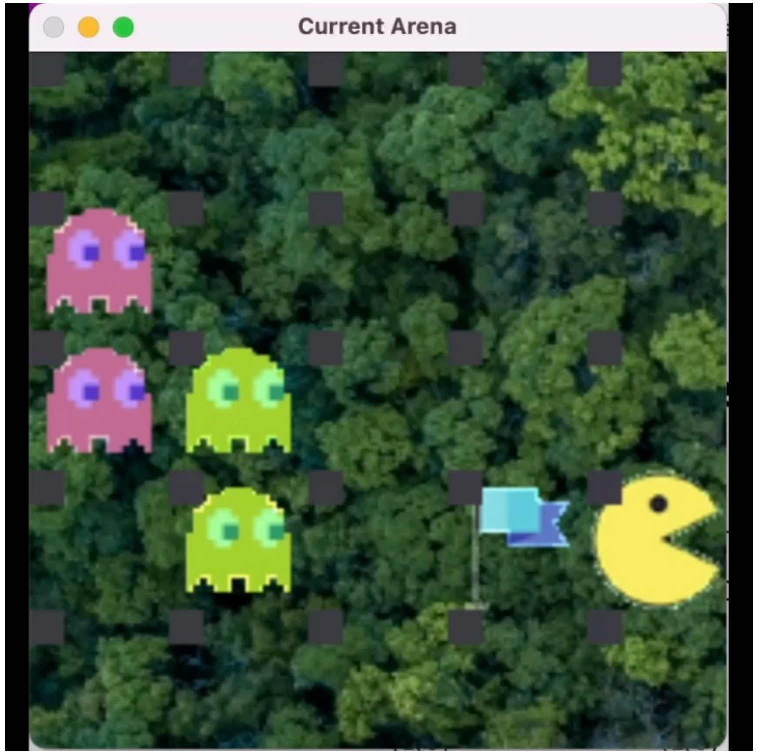
```
// Inputs from the Entit
type pacman(i: i32, j: i                                        : i32,

// Find safe cells and s
rel safe_cell(i, j) = gr
rel edge(i, j, i, j + 1,                                        l(i, j

// Find safe paths recur
rel path(i, j, i, j) = s
rel path(i1, j1, i3, j3)                                        2, j2,

// Compute the action that can lead to the goal
rel take_action(a) = pacman(i, j) and edge(i, j, m, n, a) and
                     path(m, n, g, h) and flag(g, h)
```

## Applying Top-k-Proofs Provenance



| $\widehat{\Pr}(\text{success} \mid \text{action} = \boldsymbol{a}, \boldsymbol{\theta})$ | a |
|---|---|
| $\langle 0.01, \nabla\text{UP} \rangle$ | UP |
| $\langle \mathbf{0.92}, \nabla\mathbf{RIGHT} \rangle$ | **RIGHT** |
| $\langle 0.79, \nabla\text{DOWN} \rangle$ | DOWN |
| $\langle 0.02, \nabla\text{LEFT} \rangle$ | LEFT |

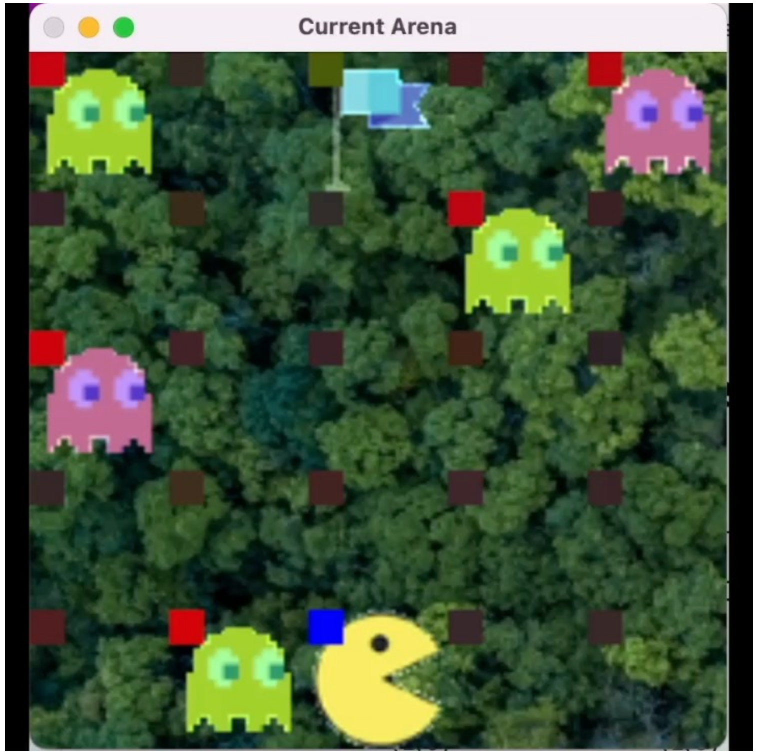
# Demo: Training PacMan-Maze Agent




Top-left marker indicates neural network prediction on what the cell contains.

- Ghost (Red)
- PacMan (Blue)
- Flag (Green)

# Demo: Testing PacMan-Maze Agent




Top-left marker indicates neural network prediction on what the cell contains.

- Ghost (Red)
- PacMan (Blue)
- Flag (Green)

# Agenda

## Scallop, A Language for Neurosymbolic Programming

Ziyang Li, Jiani Huang, Mayur Naik
[PLDI 2023]

## LASER: A Neurosymbolic Framework For Learning Spatio-temporal Scene Graphs with Weak Supervision

Jiani Huang, Ziyang Li, Ser-Nam Lim, Mayur Naik
[ICLR 2025]

## ESCA: Contextualizing Embodied Agents via Scene-Graph Generation

Jiani Huang, Ziyang Li, Matthew Kuo, Amish Sethi, Neelay Velingker, Mayank Keoliya, Ser-Nam Lim, Mayur Naik
[NeurIPS 2025]

# Agenda

Scallop, A Language for Neurosymbolic Programming

Ziyang Li, Jiani Huang, Mayur Naik
[PLDI 2023]

LASER: A Neurosymbolic Framework For Learning Spatio-temporal Scene Graphs with Weak Supervision

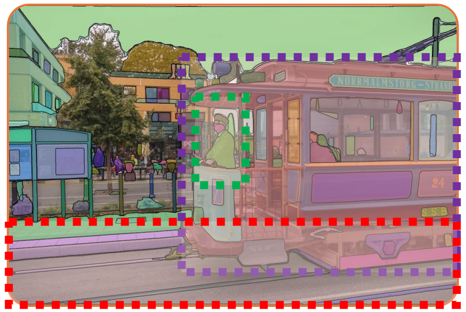
Jiani Huang, Ziyang Li, Ser-Nam Lim, Mayur Naik
[ICLR 2025]

ESCA: Contextualizing Embodied Agents via Scene-Graph Generation

Jiani Huang, Ziyang Li, Matthew Kuo, Amish Sethi, Neelay Velingker, Mayank Keoliya, Ser-Nam Lim, Mayur Naik
[NeurIPS 2025]

# Neurosymbolic Multi-modal Alignment

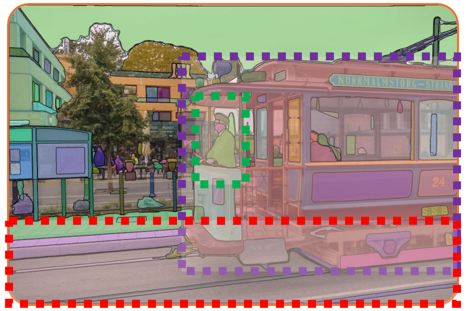
Unknown Granularity



"A <u>driver</u> is driving the <u>trolley</u> <u>towards the left</u>"

Known Granularity

# Neurosymbolic Multi-modal Alignment



Video

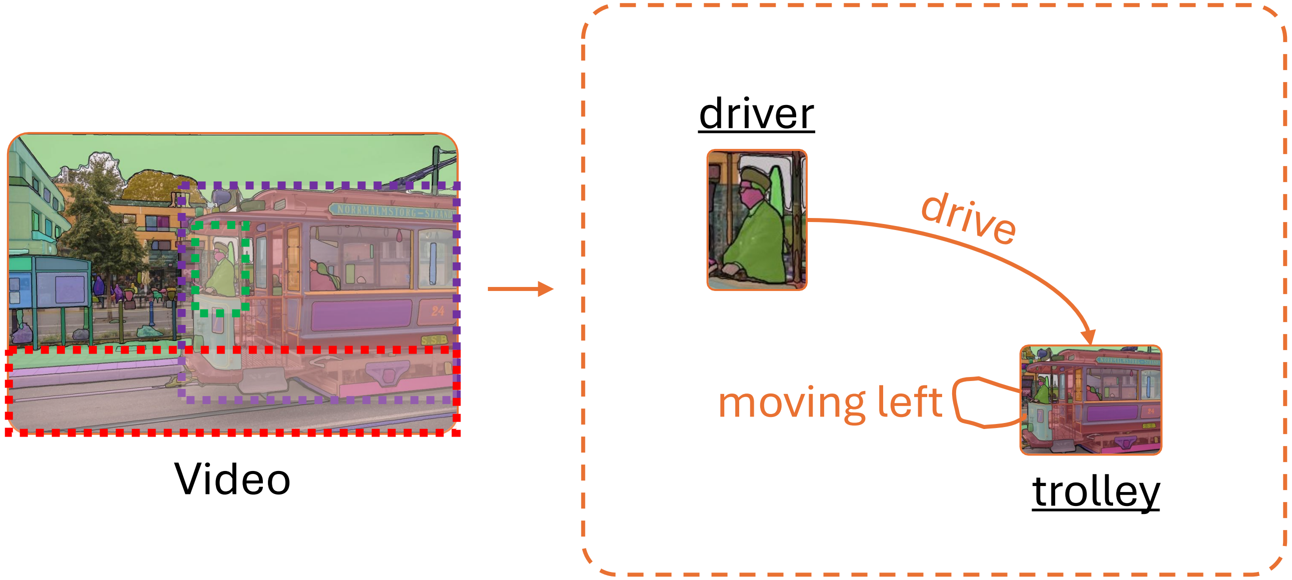
Spatio-Temporal
Scene Graph

**Alignment**

"A driver is driving the trolley towards the left"

Natural Language

# Neurosymbolic Multi-modal Alignment



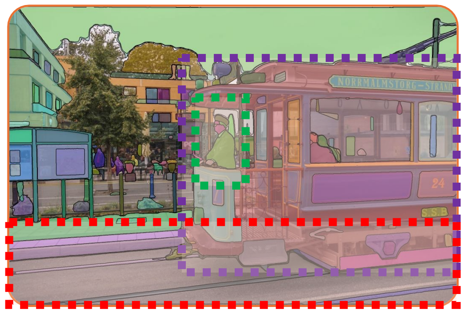

Video

**Scene Graph**

"A driver is driving the trolley towards the left"

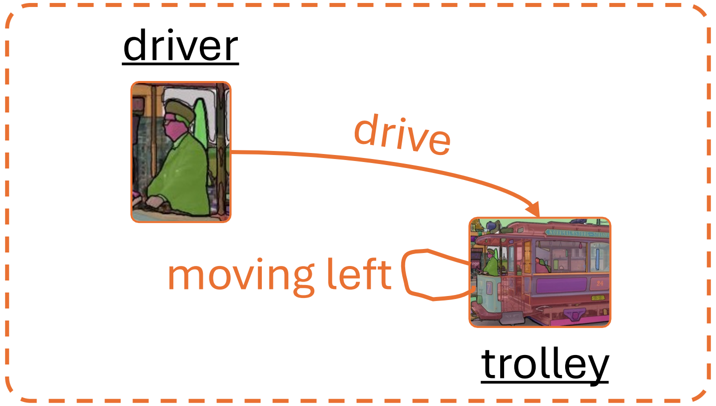
Natural Language

# Aligning Symbolic Representations



Video

Scene
Graph
Generator




Scene Graph

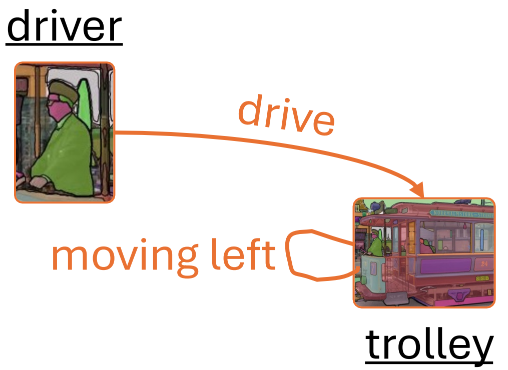
“A driver is driving the trolley towards the left”

Natural Language

# Aligning Symbolic Representations

Video

Scene Graph Generator →




Scene Graph

"A driver is driving the trolley towards the left"

Natural Language

Language Model →

$\exists x, y$: Globally(driver($x$) $\wedge$ trolley($y$) $\wedge$ driving($x, y$) $\wedge$ moving_left($y$))

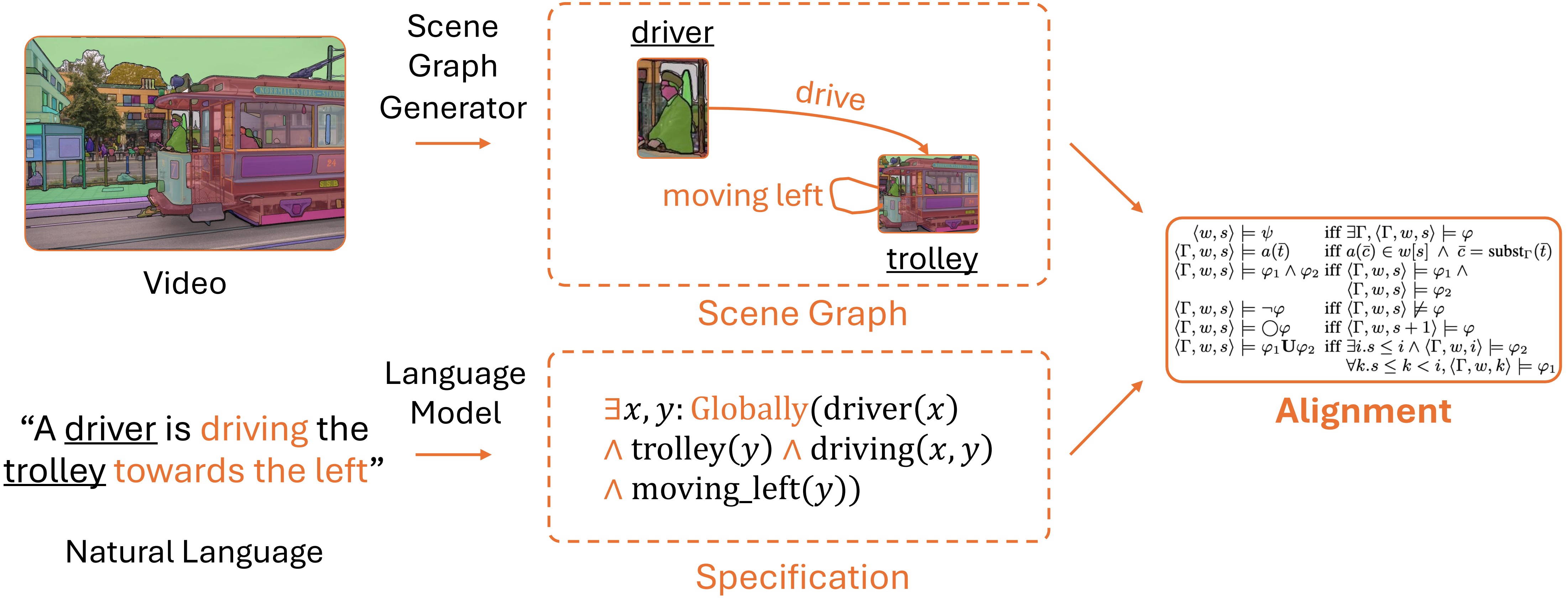
**Specification**

$$\begin{aligned} \text{(Formula)}\ \varphi &::= a(\bar{t}) \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \neg\varphi \\ &\mid \bigcirc\varphi \mid \varphi_1 \,\mathbf{U}\, \varphi_2 \mid \Box\varphi \mid \Diamond\varphi \\ \text{(Specification)}\ \psi &::= \exists v_1, \ldots, v_k, \text{s.t. } \varphi \end{aligned}$$

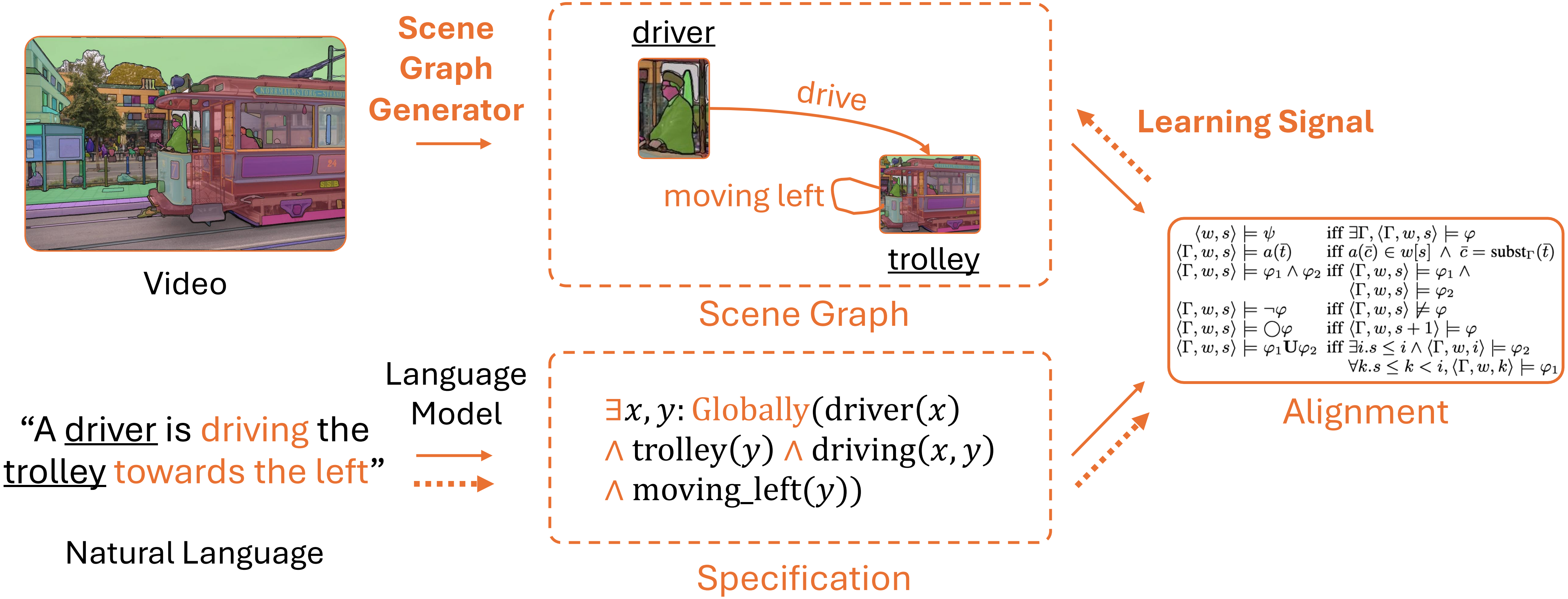
**Specification Language**

Based on Linear Temporal Logic

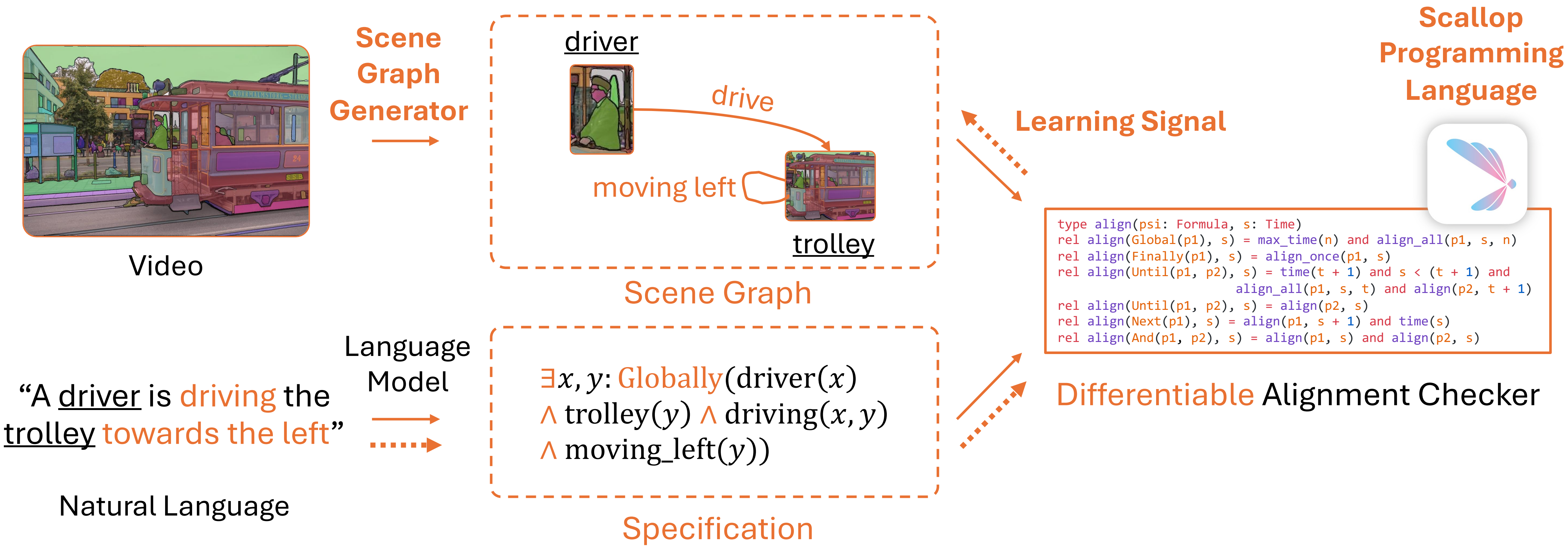
# Aligning Symbolic Representations



# Training by Aligning Symbolic Representations



# Training by Aligning Symbolic Representations




LASER: A Neurosymbolic Framework For Learning Spatio-temporal Scene Graphs with Weak Supervision, Huang et. al., 2025

# Specification Language and Alignment

$$
\begin{aligned}
\text{(Term)}\quad & t ::= c \mid v \\
\text{(Formula)}\quad & \varphi ::= a(\bar{t}) \mid \neg a(\bar{t}) \\
& \quad \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \\
& \quad \mid \bigcirc \varphi \mid \varphi_1 \,\mathbf{U}\, \varphi_2 \mid \Box \varphi \mid \Diamond \varphi \\
\text{(Specification)}\quad & \psi ::= \exists v_1, \ldots, v_k, \text{s.t. } \varphi
\end{aligned}
$$

```
type Term = Constant(String) | Var(VarId)
type Atom = UnaryAtom(String, Term) | BinaryAtom(String, Term, Term)
type Spec = Atom(Atom) | NotAtom(Atom)
          | And(Spec, Spec) | Or(Spec, Spec)
          | Global(Spec) | Until(Spec, Spec) | Next(Spec) | Finally(Spec)
```

# Specification Language and Alignment

$$\langle \Gamma, w, [i, j] \rangle \models \varphi$$

```
type align(phi: Spec, start: Time, end: Time)
```

# Specification Language and Alignment

**Evaluation Context**

Context contains variable assignments that is stored in Scallop's intentional database (IDB)

**Spatio-Temporal Scene Graph**

NN predicted scene graph is encoded as probabilistic facts within Scallop's extensional database (EDB)

$$\langle \Gamma, w, [i, j] \rangle \models \varphi$$

```
type align(phi: Spec, start: Time, end: Time)
```

# Specification Language and Alignment

$$\langle \Gamma, w, [i,j] \rangle \models a(\bar{t}) \qquad \text{iff } \forall f \in [i,j], a(\bar{c}) \in w[f] \ \wedge \ \bar{c} = \text{substitute}_{\Gamma}(\bar{t})$$

```
rel align(UnaryAtom(a, t), i, i)         = sg_unary(i, a, c) and subst_term(t, c)
rel align(BinaryAtom(a, t1, t2), i, i) = sg_binary(i, a, c1, c2) and
                                           subst_term(t1, c1) and subst_term(t2, c2)
rel align(phi, i, k)                     = align(phi, i, j) and align(phi, j + 1, k)
```

$$\langle \Gamma, w, [i,j] \rangle \models \varphi_1 \mathbf{U} \varphi_2 \ \text{ iff } \exists k, \langle \Gamma, w, [i,k] \rangle \models \varphi_1 \wedge \langle \Gamma, w, [k,j] \rangle \models \varphi_2$$

```
rel align(Until(phi_1, phi_2), i, j) = align(phi_1, i, k) and align(phi_2, k + 1, j)
```

# Probabilistic and Differentiable Alignment

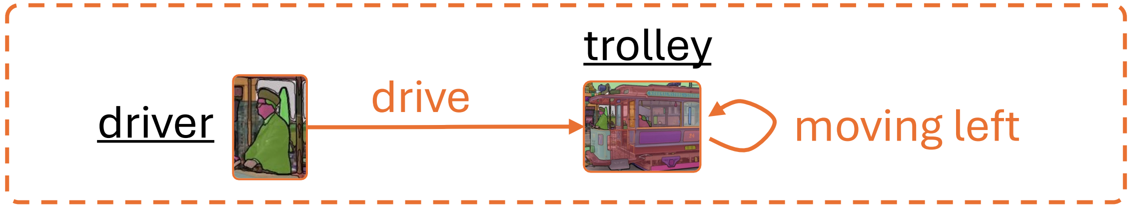
$\langle w, [i, j] \rangle \models \psi$ $\longrightarrow$ $\Pr(\langle w, [i, j] \rangle \models \psi)$ $\longrightarrow$ $\langle \Pr(\langle w, [i, j] \rangle \models \psi) \, , \nabla \Pr(\langle w, [i, j] \rangle \models \psi) \rangle$

Discrete Alignment

Probabilistic Alignment

Differentiable Alignment

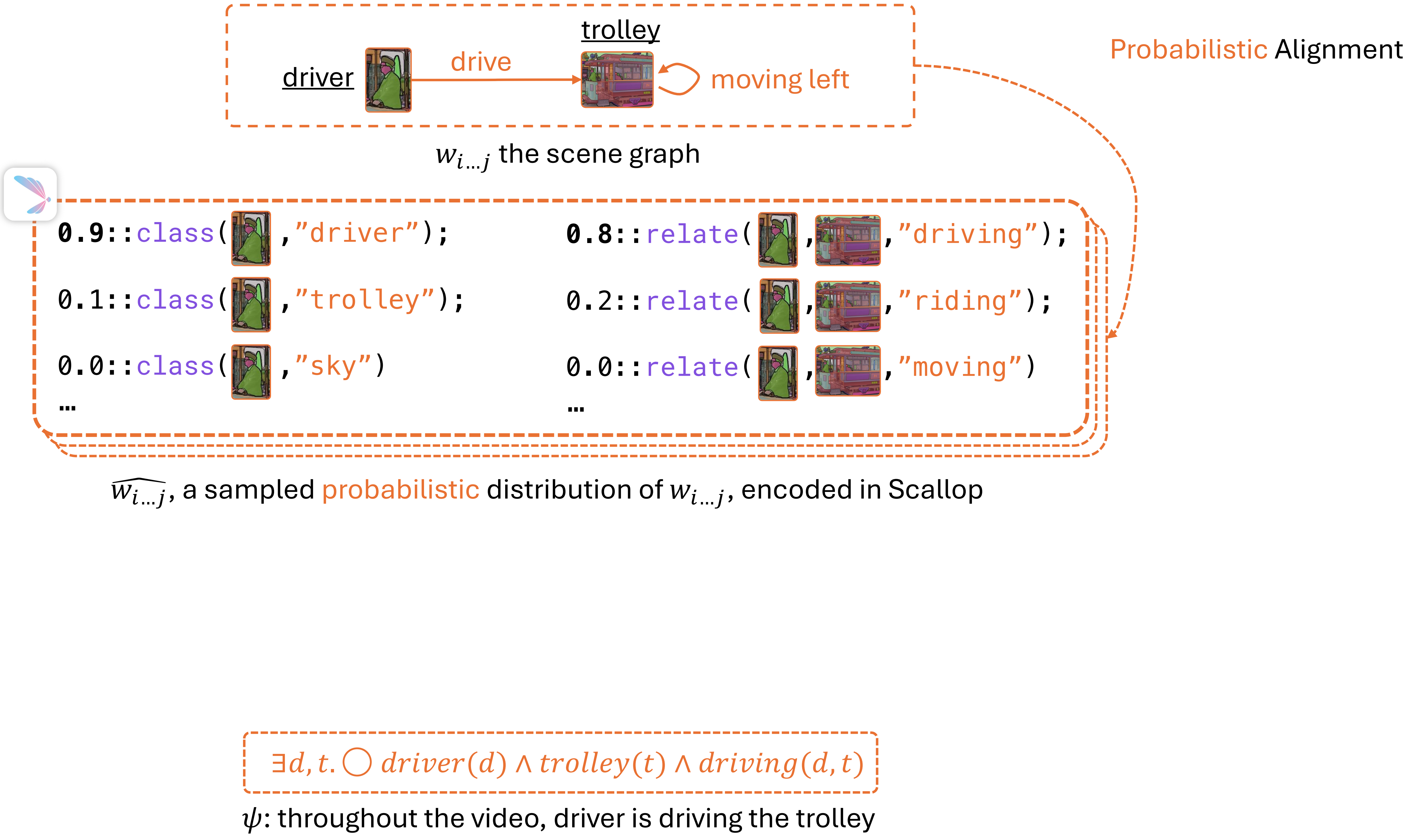
Probabilistic Alignment



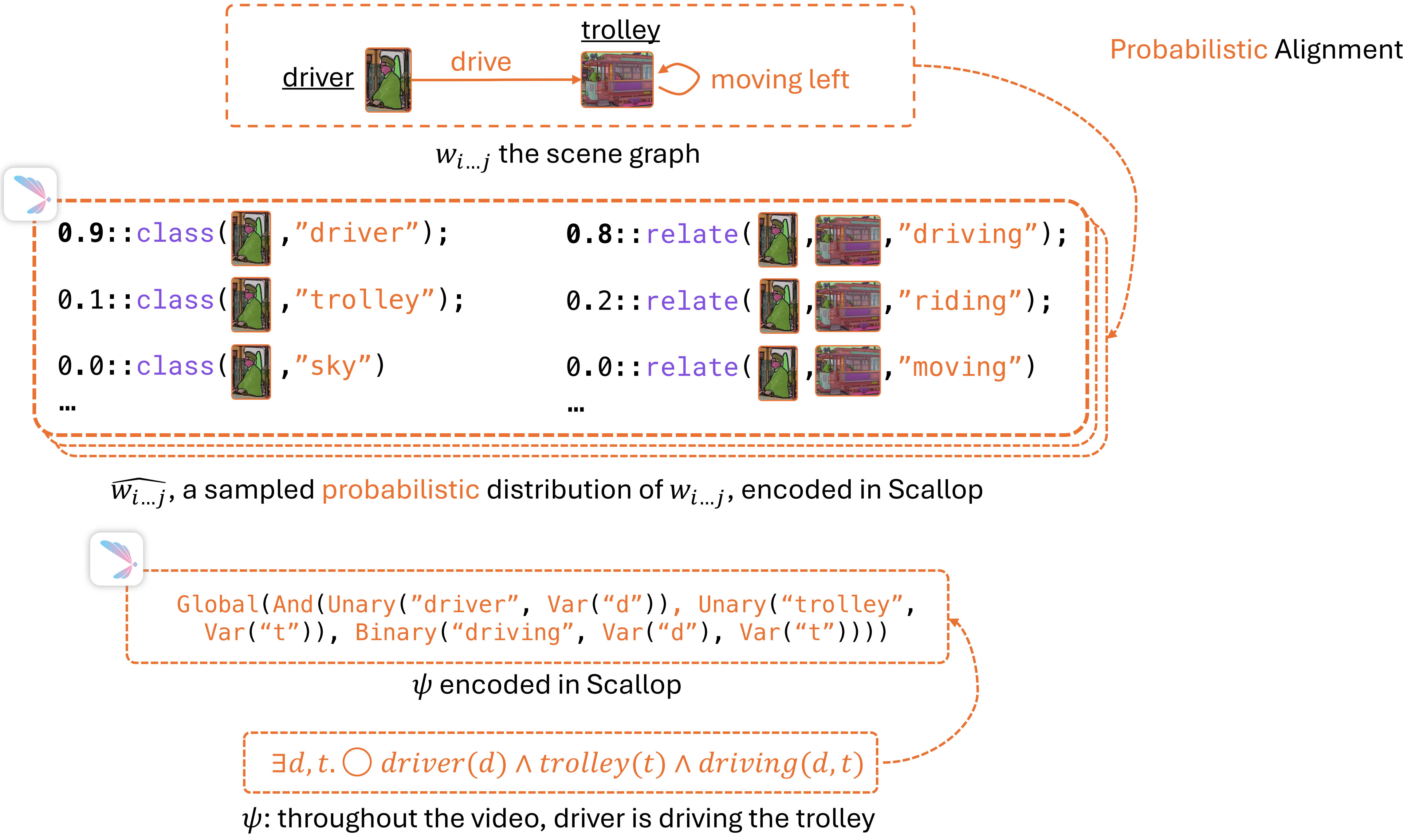

$w_{i\ldots j}$ the scene graph

$$\exists d, t. \bigcirc\ driver(d) \land trolley(t) \land driving(d, t)$$

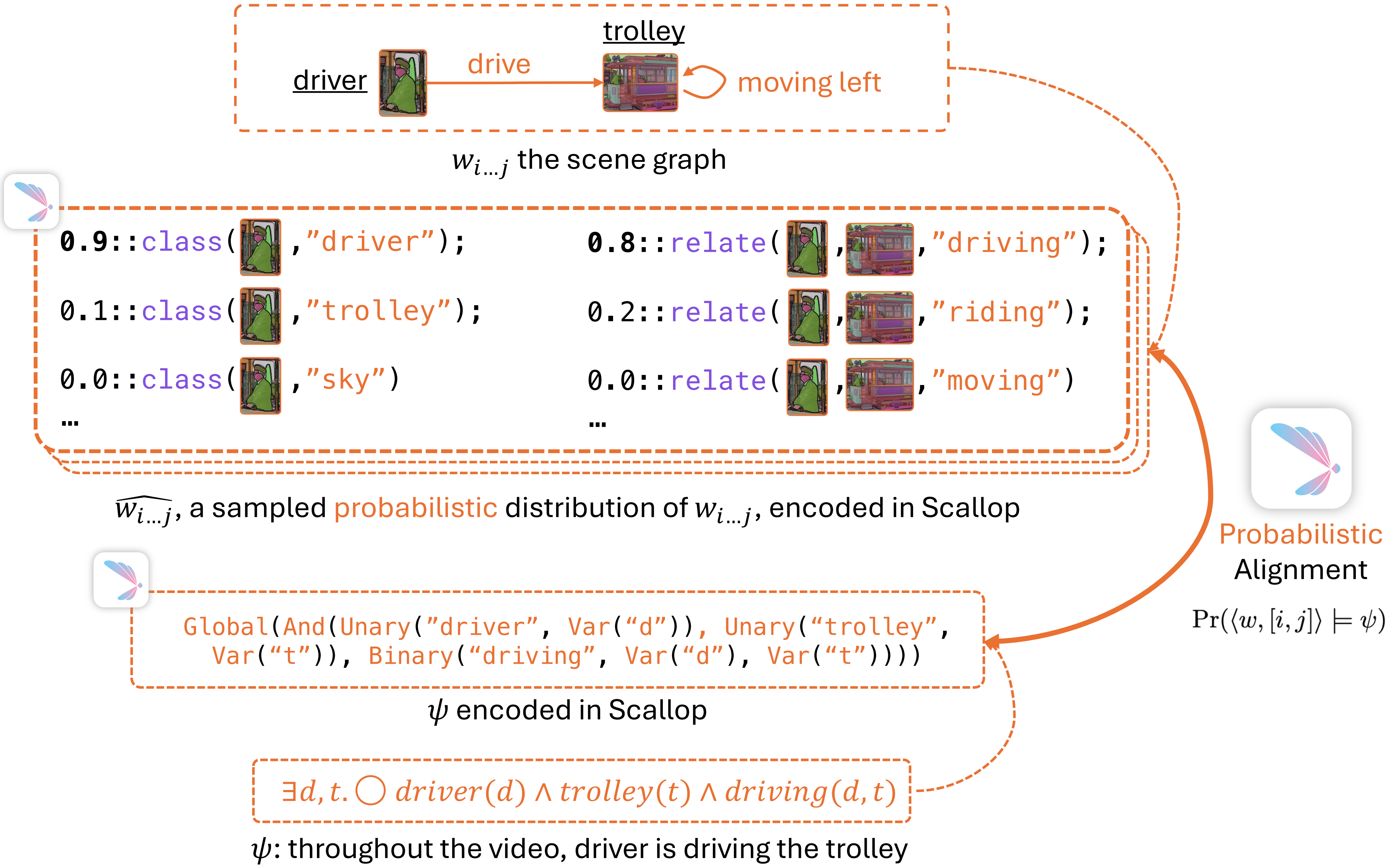
$\psi$: throughout the video, driver is driving the trolley

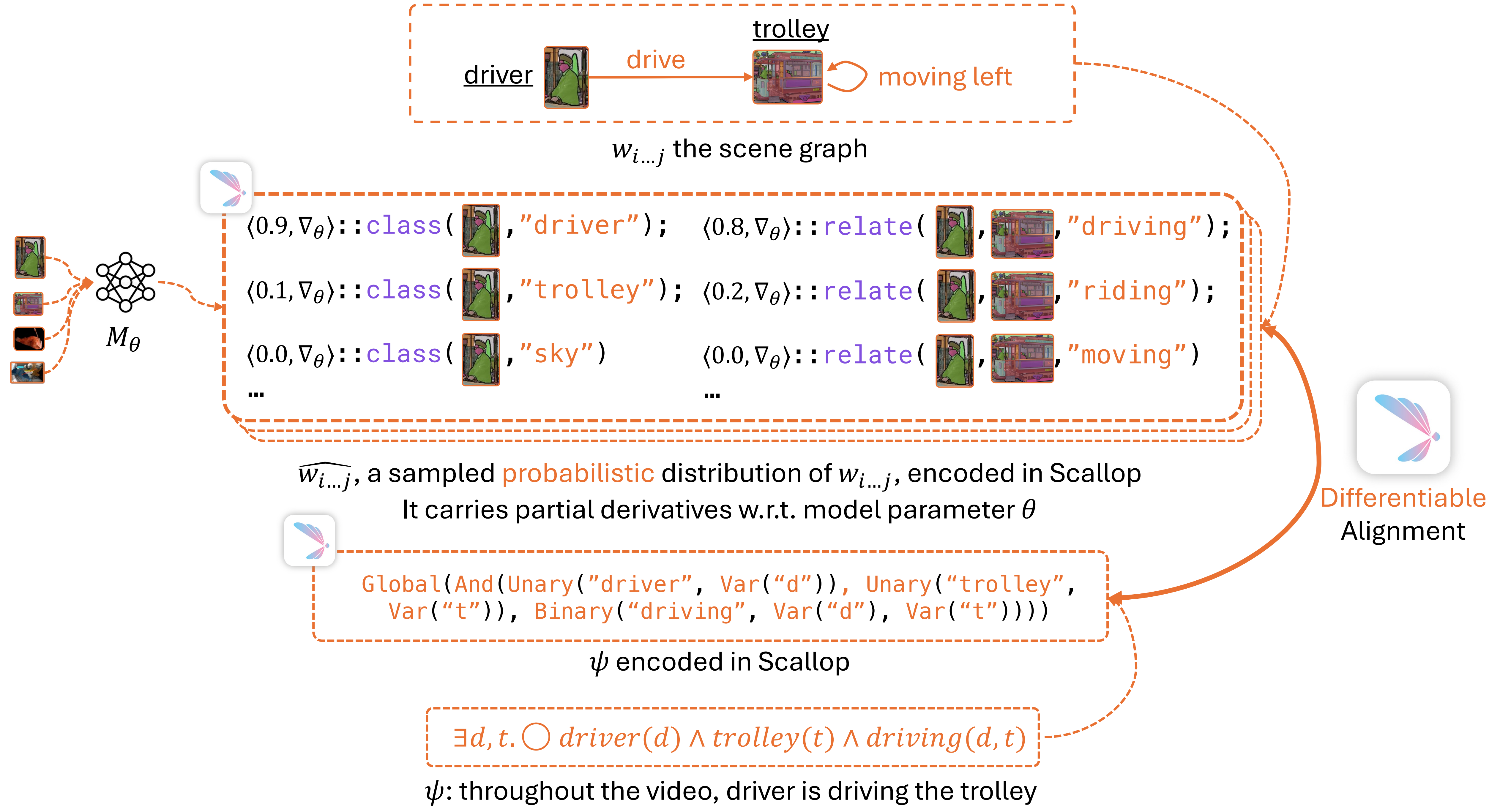


Probabilistic Alignment
trolley
driver
drive
moving left
$w_{i \ldots j}$ the scene graph
0.9::class( ,"driver");
0.1::class( ,"trolley");
0.0::class( ,"sky")
…
0.8::relate( , ,"driving");
0.2::relate( , ,"riding");
0.0::relate( , ,"moving")
…
$\widehat{w_{i \ldots j}}$, a sampled probabilistic distribution of $w_{i \ldots j}$, encoded in Scallop
$\exists d, t. \bigcirc\, driver(d) \wedge trolley(t) \wedge driving(d, t)$
$\psi$: throughout the video, driver is driving the trolley



Probabilistic Alignment
trolley
driver
drive
moving left
wi...j the scene graph
0.9::class( ,"driver");
0.1::class( ,"trolley");
0.0::class( ,"sky")
…
0.8::relate( , ,"driving");
0.2::relate( , ,"riding");
0.0::relate( , ,"moving")
…
ŵi...j, a sampled probabilistic distribution of wi...j, encoded in Scallop
Global(And(Unary("driver", Var("d")), Unary("trolley", Var("t")), Binary("driving", Var("d"), Var("t"))))
ψ encoded in Scallop
∃d, t. ◯ driver(d) ∧ trolley(t) ∧ driving(d, t)
ψ: throughout the video, driver is driving the trolley


trolley
driver
drive
moving left
w_{i…j} the scene graph
0.9::class( ,"driver");
0.1::class( ,"trolley");
0.0::class( ,"sky")
…
0.8::relate( , ,"driving");
0.2::relate( , ,"riding");
0.0::relate( , ,"moving")
…
ŵ_{i…j}, a sampled probabilistic distribution of w_{i…j}, encoded in Scallop
Probabilistic
Alignment
Pr(⟨w, [i, j]⟩ ⊨ ψ)
Global(And(Unary("driver", Var("d")), Unary("trolley", Var("t")), Binary("driving", Var("d"), Var("t"))))
ψ encoded in Scallop
∃d, t. ◯ driver(d) ∧ trolley(t) ∧ driving(d, t)
ψ: throughout the video, driver is driving the trolley


trolley
driver
drive
moving left
$w_{i...j}$ the scene graph
$\langle 0.9, \nabla_\theta \rangle$::class( ,"driver");
$\langle 0.8, \nabla_\theta \rangle$::relate( , ,"driving");
$\langle 0.1, \nabla_\theta \rangle$::class( ,"trolley");
$\langle 0.2, \nabla_\theta \rangle$::relate( , ,"riding");
$\langle 0.0, \nabla_\theta \rangle$::class( ,"sky")
$\langle 0.0, \nabla_\theta \rangle$::relate( , ,"moving")
...
...
$M_\theta$
$\widehat{w_{i...j}}$, a sampled probabilistic distribution of $w_{i...j}$, encoded in Scallop
It carries partial derivatives w.r.t. model parameter $\theta$
Differentiable
Alignment
Global(And(Unary("driver", Var("d")), Unary("trolley", Var("t")), Binary("driving", Var("d"), Var("t"))))
$\psi$ encoded in Scallop
$\exists d, t. \bigcirc\, driver(d) \wedge trolley(t) \wedge driving(d, t)$
$\psi$: throughout the video, driver is driving the trolley

# Probabilistic and Differentiable Alignment

$\langle w, [i, j] \rangle \models \psi$ $\longrightarrow$ $\Pr(\langle w, [i, j] \rangle \models \psi)$ $\longrightarrow$ $\langle \Pr(\langle w, [i, j] \rangle \models \psi) \,, \nabla \Pr(\langle w, [i, j] \rangle \models \psi) \rangle$

Discrete Alignment

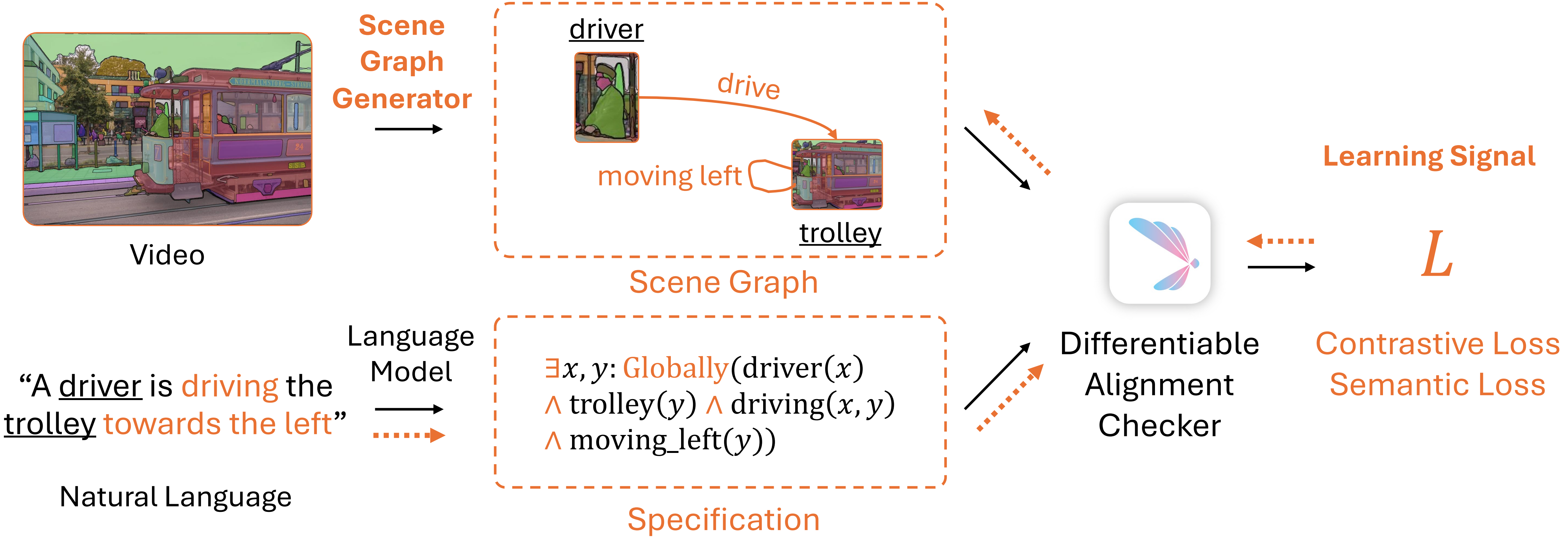
Probabilistic Alignment

Differentiable Alignment

| | | | |
|---|---|---|---|
| (Tag Space) | $t$ | $\in$ | $T$ |
| (False) | $\mathbf{0}$ | $\in$ | $T$ |
| (True) | $\mathbf{1}$ | $\in$ | $T$ |
| (Disjunction) | $\oplus$ | : | $T \times T \to T$ |
| (Conjunction) | $\otimes$ | : | $T \times T \to T$ |
| (Negation) | $\ominus$ | : | $T \rightharpoonup T$ |
| (Saturation) | $\circledequal$ | : | $T \times T \to \text{Bool}$ |

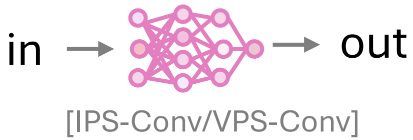
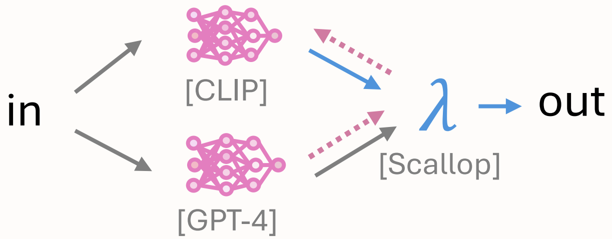
Provenance Framework

# Training by Aligning Symbolic Representations




LASER: A Neurosymbolic Framework For Learning Spatio-temporal Scene Graphs with Weak Supervision, Huang et. al., 2025

# Results

| Method | Unary Acc. / Binary R@10 |
|---|---|
|   | 15.1% / 23.4% |
|   | **27.8% / 54.0%** |

Trained and Evaluated on Dataset *OpenPVSG*.

# Results

| **Method** (with LASER) | | Unary | | | Binary | | |
|---|---|---|---|---|---|---|---|
| | | **R@1** | **R@5** | **R@10** | **R@1** | **R@5** | **R@10** |
| VIOLET | Base | 0.0660 | 0.1855 | 0.2983 | 0.0460 | 0.1307 | 0.2636 |
| | Fine-tuned | 0.0878 | 0.2574 | 0.3463 | 0.0501 | 0.2028 | 0.3451 |
| | Incr. | ↑ 0.0218 | ↑ 0.0719 | ↑ 0.0480 | ↑ 0.0041 | ↑ 0.0721 | ↑ 0.0815 |
| SigLIP | Base | 0.0000 | 0.0179 | 0.0483 | 0.0000 | 0.0362 | 0.1667 |
| | Fine-tuned | 0.1467 | 0.2627 | 0.3152 | 0.0347 | 0.1624 | 0.3012 |
| | Incr. | ↑ 0.1467 | ↑ 0.2448 | ↑ 0.2669 | ↑ 0.0347 | ↑ 0.1262 | ↑ 0.1345 |
| CLIP | Base | 0.1633 | 0.3381 | 0.4404 | 0.0197 | 0.0673 | 0.0988 |
| | Fine-tuned | 0.2778 | 0.5231 | 0.6402 | 0.1482 | 0.4214 | 0.5398 |
| | Incr. | ↑ 0.1145 | ↑ 0.1850 | ↑ 0.1998 | ↑ 0.1284 | ↑ 0.3540 | ↑ 0.4410 |

Table 1: We show the performance improvements of base backbone models and their fine-tuned version, on the R@k metrics of unary and binary predicate prediction. As shown by the increments, LASER's weak supervisory learning framework significantly enhances all three models' performance on the STSG extraction tasks.

# Agenda

Scallop, A Language for Neurosymbolic Programming

Ziyang Li, Jiani Huang, Mayur Naik
[PLDI 2023]

LASER: A Neurosymbolic Framework For Learning Spatio-temporal Scene Graphs with Weak Supervision

Jiani Huang, Ziyang Li, Ser-Nam Lim, Mayur Naik
[ICLR 2025]

ESCA: Contextualizing Embodied Agents via Scene-Graph Generation

Jiani Huang, Ziyang Li, Matthew Kuo, Amish Sethi, Neelay Velingker, Mayank Keoliya, Ser-Nam Lim, Mayur Naik
[NeurIPS 2025]

# Agenda

Scallop, A Language for Neurosymbolic Programming

Ziyang Li, Jiani Huang, Mayur Naik
[PLDI 2023]

LASER: A Neurosymbolic Framework For Learning Spatio-temporal Scene Graphs with Weak Supervision

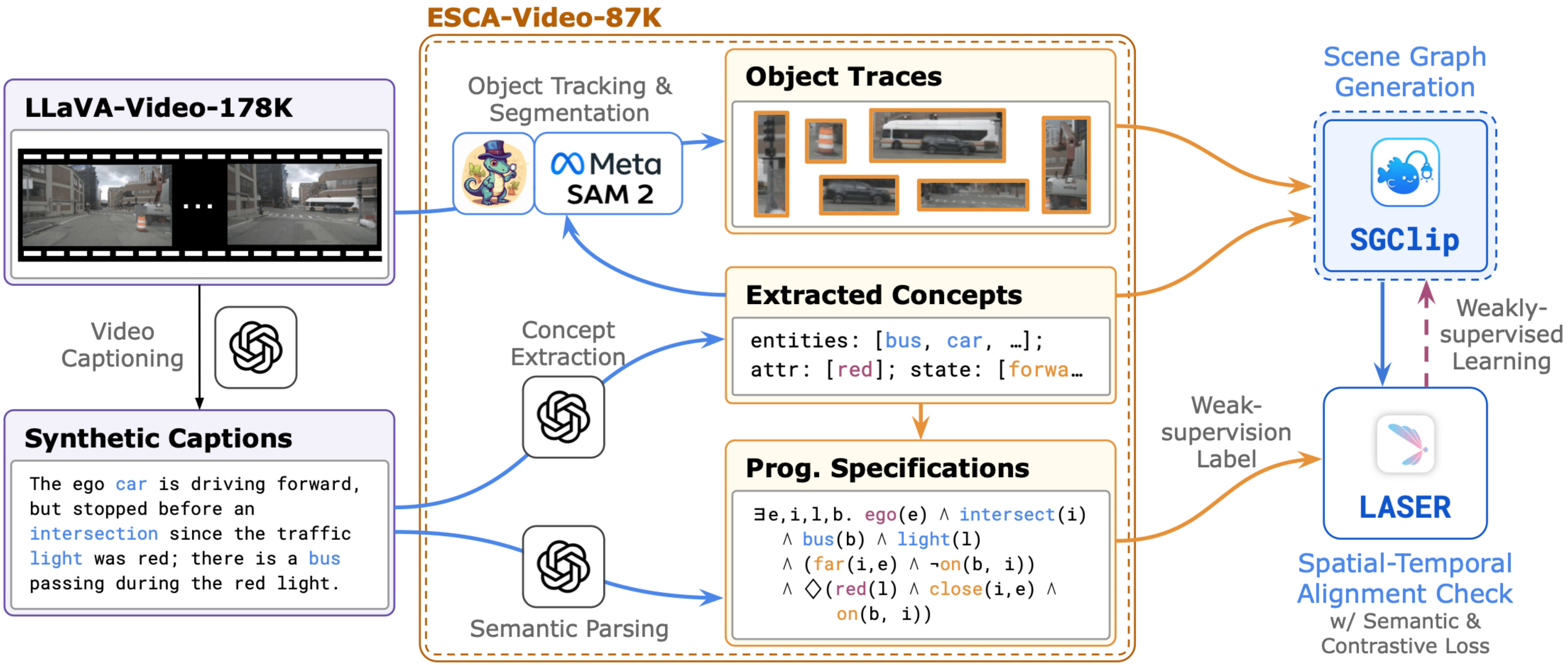
Jiani Huang, Ziyang Li, Ser-Nam Lim, Mayur Naik
[ICLR 2025]

ESCA: Contextualizing Embodied Agents via Scene-Graph Generation

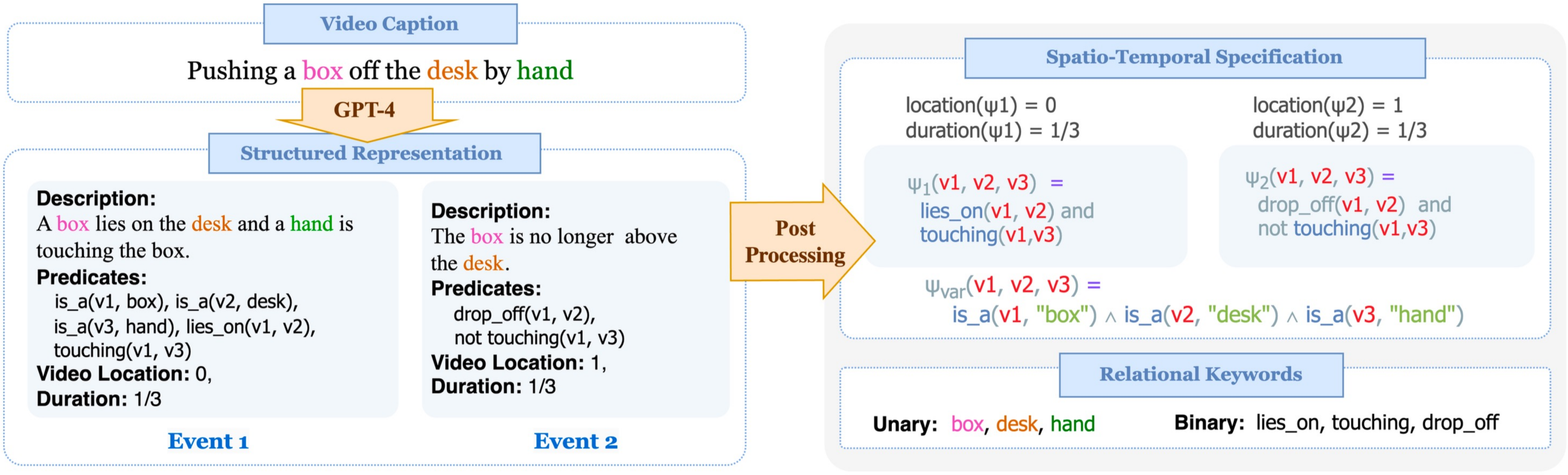
Jiani Huang, Ziyang Li, Matthew Kuo, Amish Sethi, Neelay Velingker, Mayank Keoliya, Ser-Nam Lim, Mayur Naik
[NeurIPS 2025]

# ESCA: Contextualizing Embodied Agents via Scene-Graph Generation

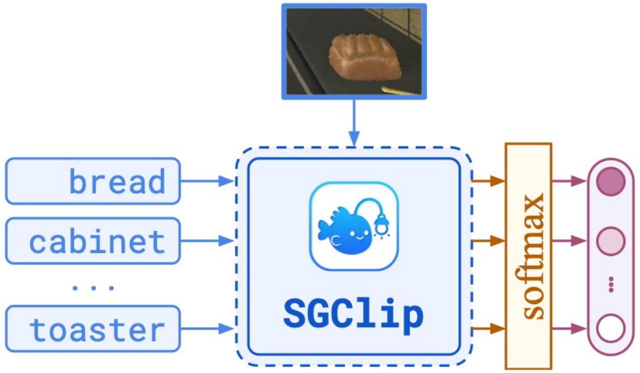
# Scaling Up Alignment Training Pipeline



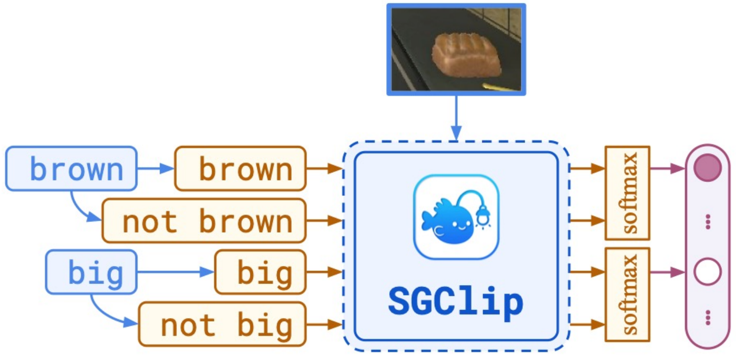
# Scaling Up Alignment Training Pipeline



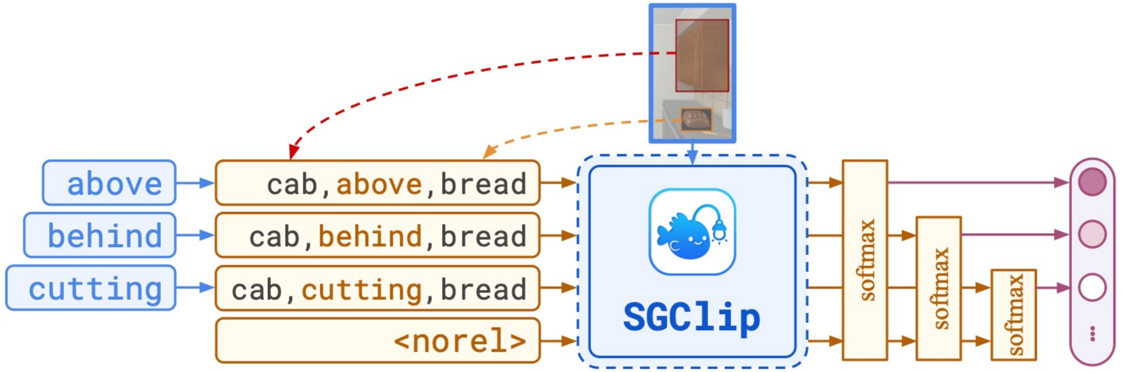
# Unified Open-Vocabulary Interface




(a) Entity classes




(b) Attributes

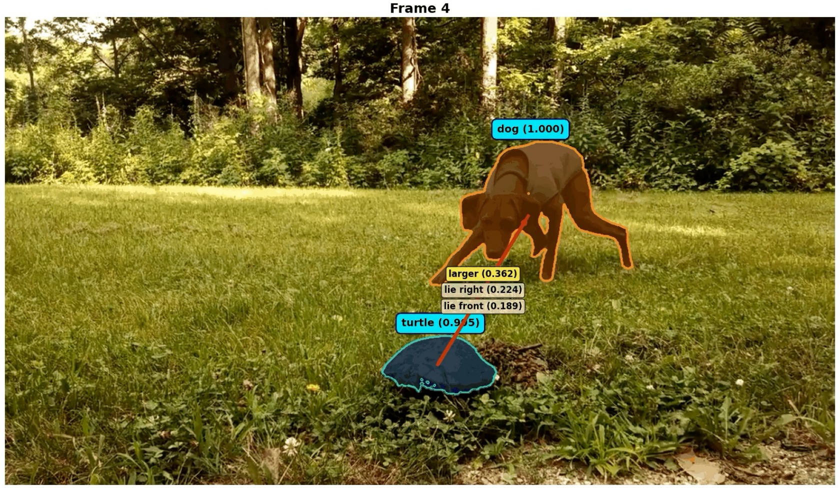



(c) Binary relations

# Promptable STSG Generation Pipeline

- Model-driven self-supervised training
  - Eliminating the need for fine-grained STSG label
  - Scaling up training for SGClip
- Unified open-vocabulary interface
  - Can simultaneously handle entity classes, attributes, actions, spatial relations, interactions
  - Zero-shot generalizable to diverse scenarios such as action localization
- Controllable scene graph generation granularity
  - Granularity is controlled via prompting the caption generation and concept extraction

A couple walking their dog

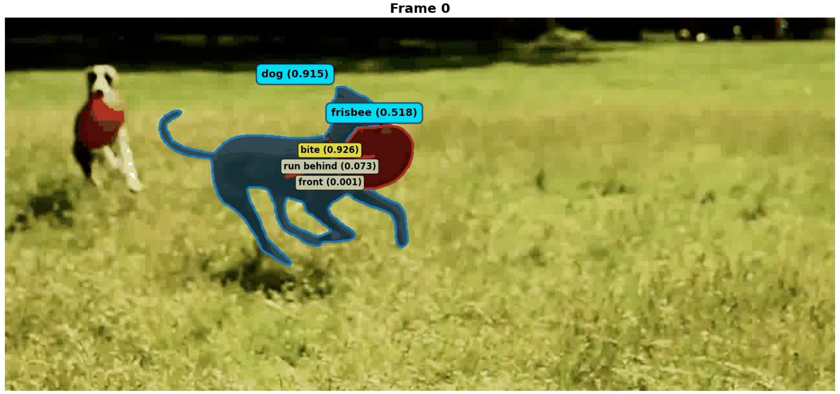

A dog runs around a turtle

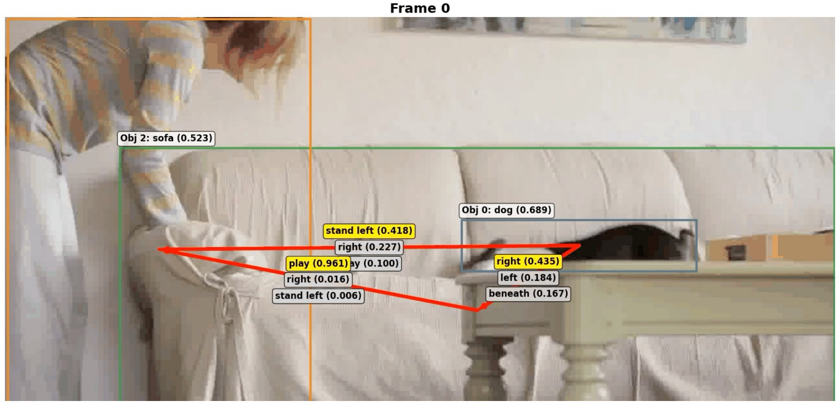
A car quickly drives pass a person



A person riding on a horse

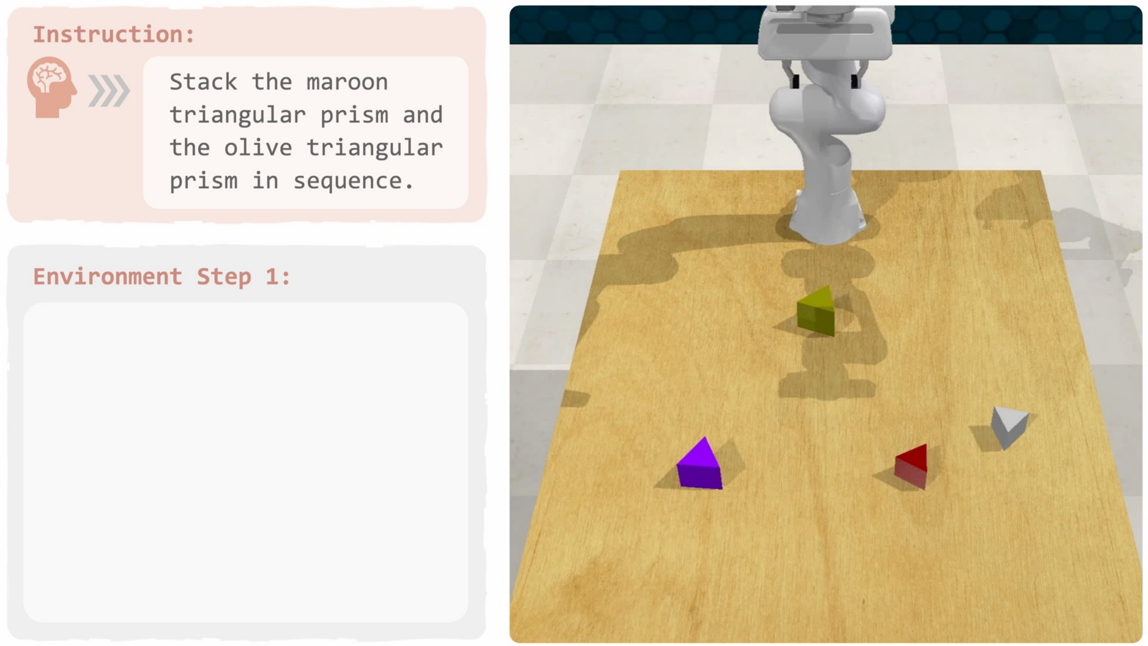
A dog playing with a frisbee




A dog jumps left on the sofa to play with a woman

# Application to VLM-based Embodied Agents

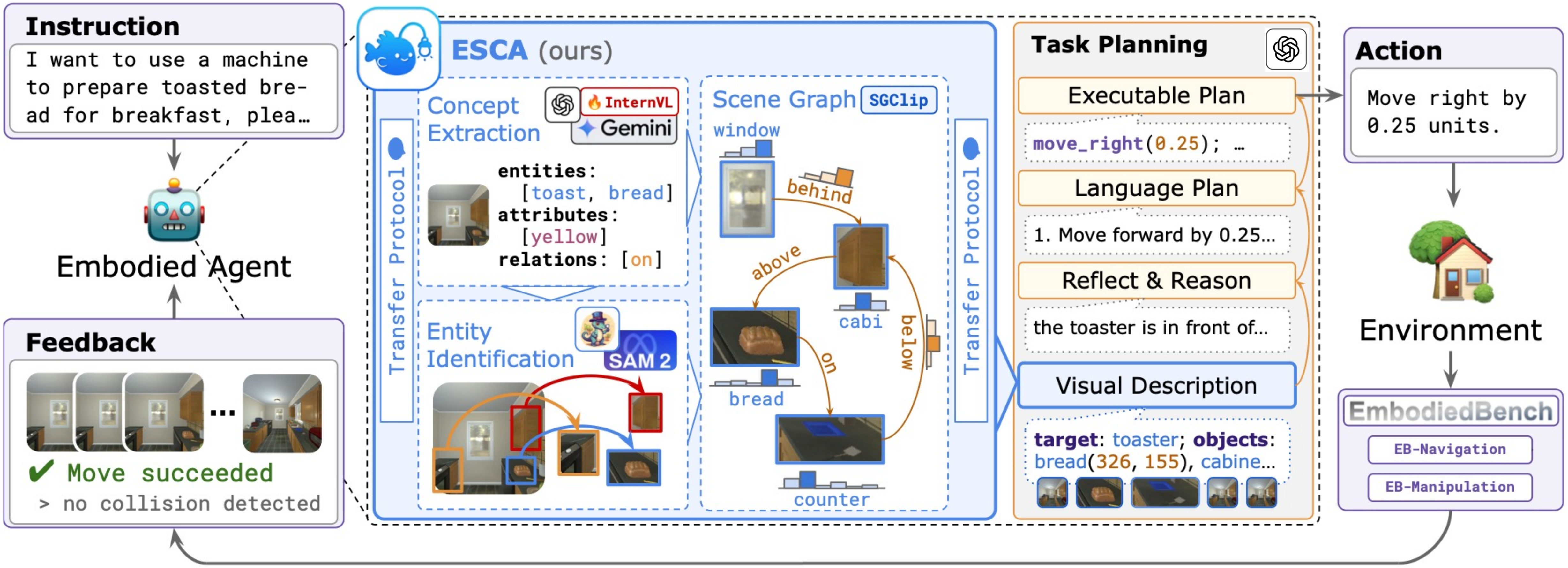



EB-Manipulation


EB-Navigation

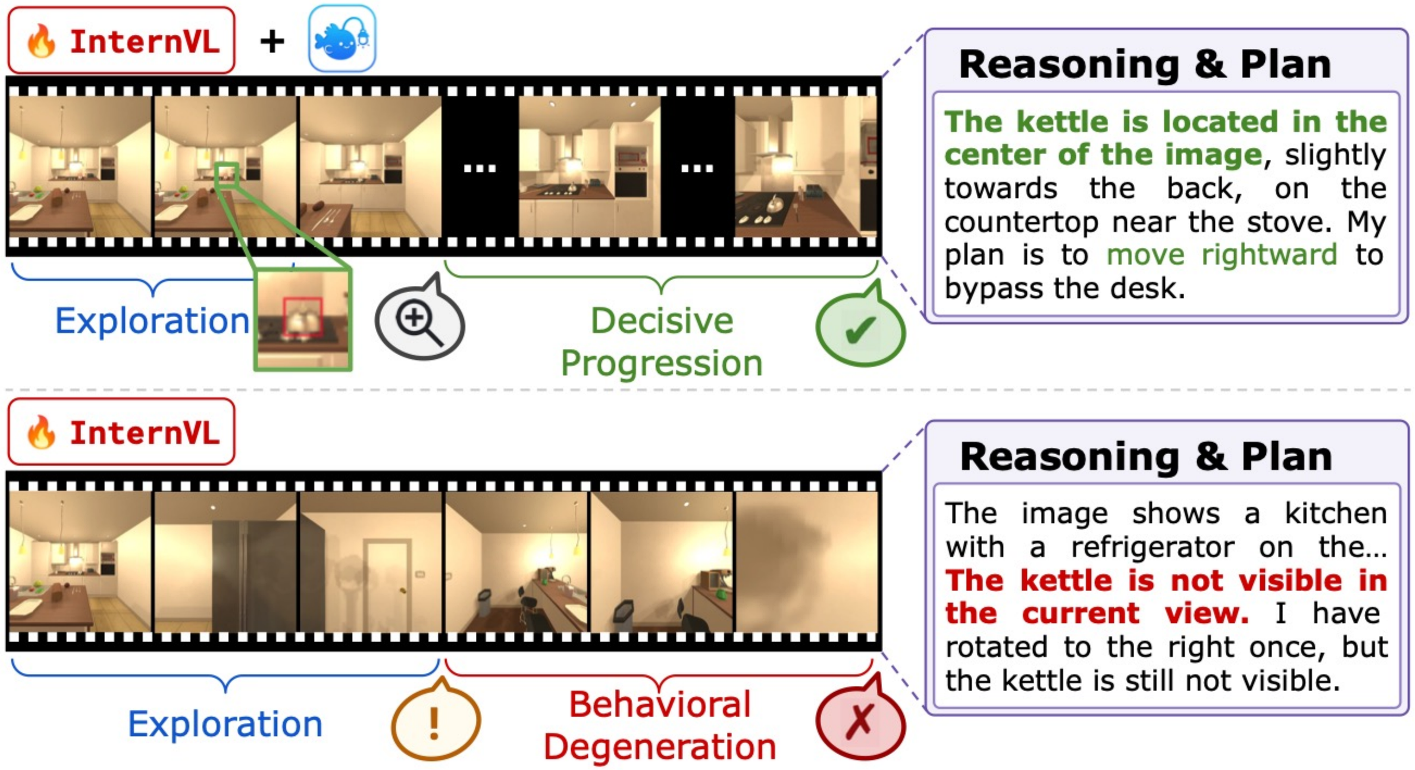
Comprehensive Benchmarking Multi-modal Large Language Models for Vision-Driven Embodied Agents, Yang et. al., 2025

# Application to Embodied Agents

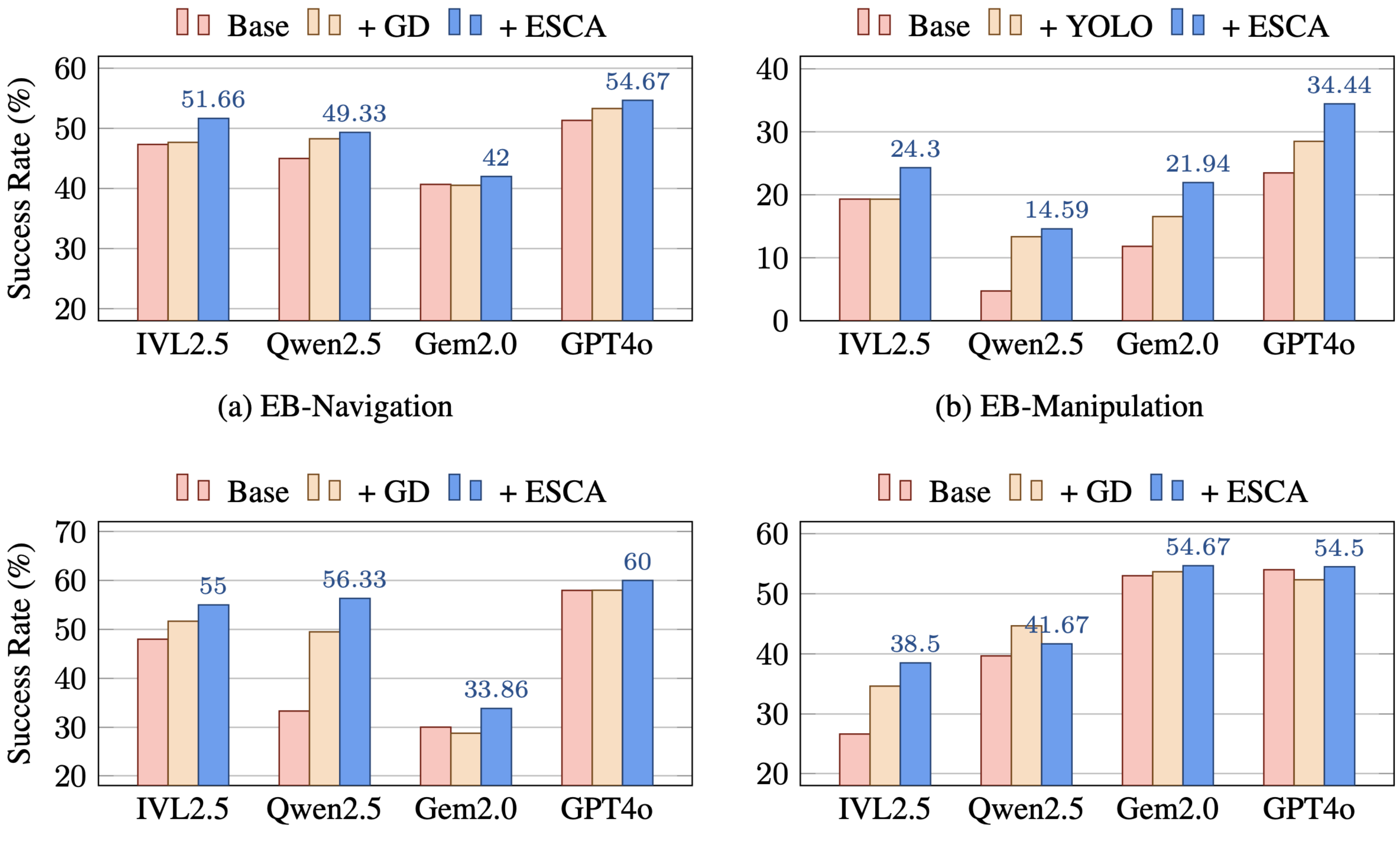


# Application to Embodied Agents

**Instruction**
"navigate to the kettle on the stove"



# Results

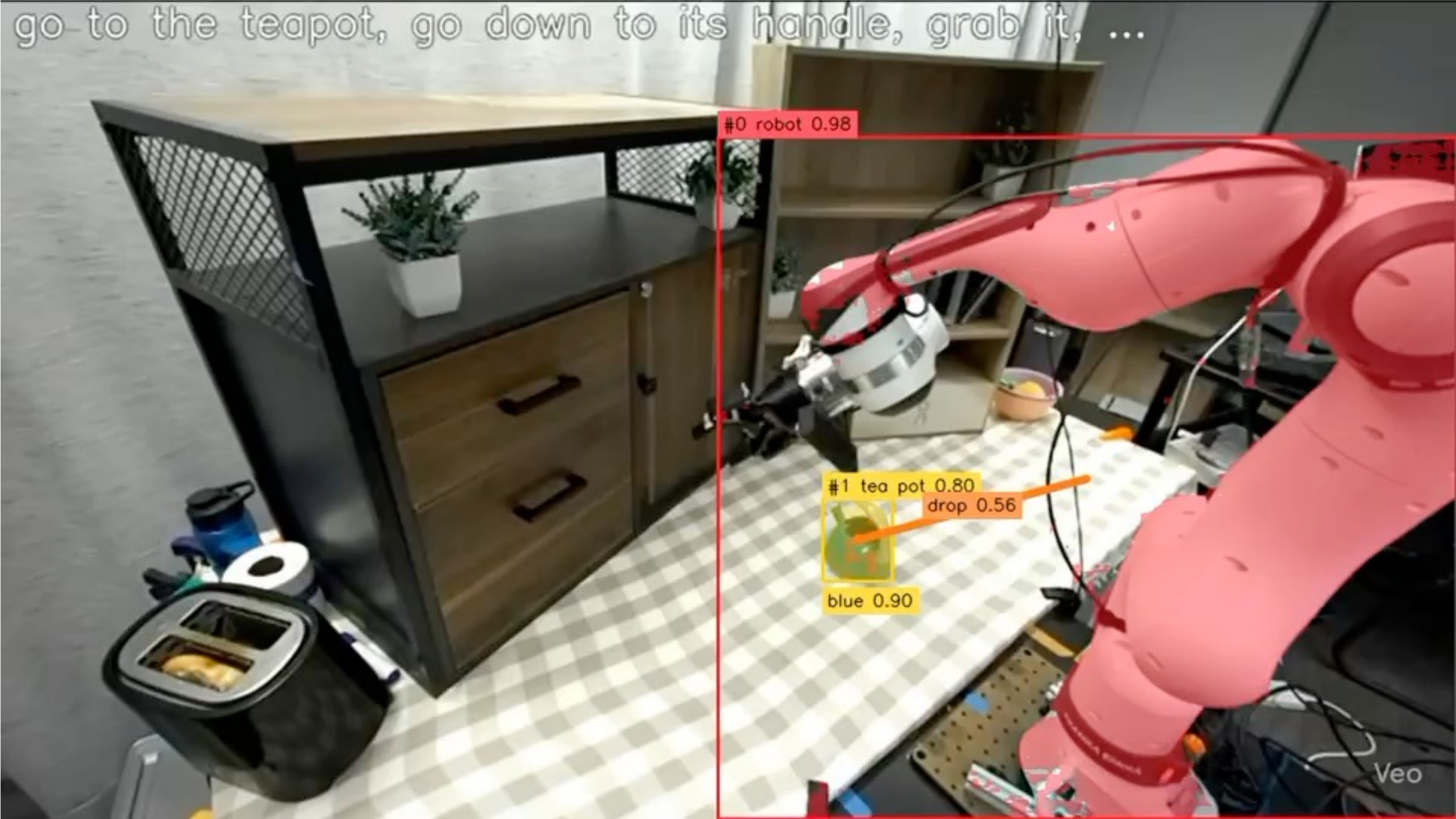



(a) EB-Navigation

(b) EB-Manipulation

(c) EB-Habitat

(d) EB-Alfred


go to the teapot, go down to its handle, grab it, ...
#0 robot 0.98
#1 tea pot 0.80
drop 0.56
blue 0.90
Veo

# Agenda

## Scallop, A Language for Neurosymbolic Programming

Ziyang Li, Jiani Huang, Mayur Naik
[PLDI 2023]

## LASER: A Neurosymbolic Framework For Learning Spatio-temporal Scene Graphs with Weak Supervision

Jiani Huang, Ziyang Li, Ser-Nam Lim, Mayur Naik
[ICLR 2025]

## ESCA: Contextualizing Embodied Agents via Scene-Graph Generation

Jiani Huang, Ziyang Li, Matthew Kuo, Amish Sethi, Neelay Velingker, Mayank Keoliya, Ser-Nam Lim, Mayur Naik
[NeurIPS 2025]

# Key Takeaway

- Grounded and accurate perception is key to embodied systems
- Structured representation for perception provides trustworthiness and transparency to downstream planning and control
- Neurosymbolic learning alleviates the need for full-supervision labels
- Neurosymbolic learning pipeline trains better mid-level and high-level perception systems
- Formal domain-specific languages (DSL) and program synthesis is the bridge connecting natural and symbolic domains

# General Messages

- Programming Languages create abstractions
  - Scallop abstracts away probabilistic and differentiable reasoning
  - Spatial-Temporal Specifications abstract away naturalness
- Synthesized programs have usages beyond execution
  - Specifications help training the underlying vision-language model
    - (Similar to synthesized reward function or value functions during RL)
  - Synthesized Scallop program can be part of end-to-end neurosymbolic learning pipeline as a reasoning module

# References


[1] Scallop: From Probabilistic Deductive Databases to Scalable Differentiable Reasoning, Li et. al., NeurIPS 2021
[2] Scallop, A Language for Neurosymbolic Programming, Li et. al., PLDI 2023
[3] Improved Logical Reasoning of Language Models via Differentiable Symbolic Programming, Huang et. al., ACL 2023
[4] Relational Programming with Foundation Models, Li et. al., AAAI 2024
[5] Neurosymbolic Programming with Scallop: Principles and Practice, Li et. al., Foundations and Trends in PL, 2024
[6] LASER: A Neuro-Symbolic Framework for Learning STSGs with Weak Supervision, Huang et. al., ICLR 2025
[7] Lobster: A GPU-Accelerated Framework for Neurosymbolic Programming, Biberstein et. al., ASPLOS 2026
[8] ESCA: Contextualizing Embodied Agents via Scene-Graph Generation, Huang et. al., In Submission