Probabilistic Programming with Deep Learning

A Neural-Symbolic Perspective

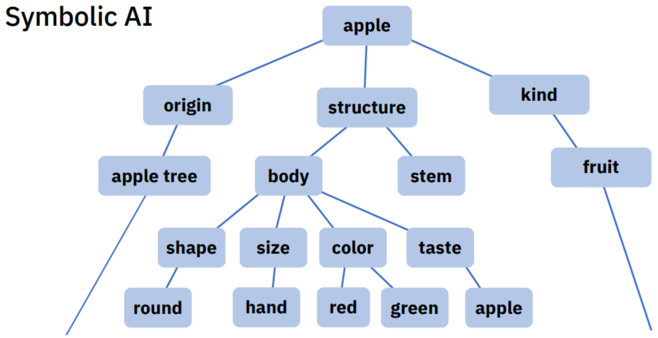
Xin Zhang

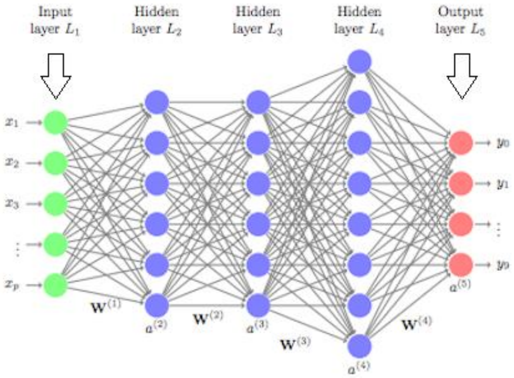
Peking University

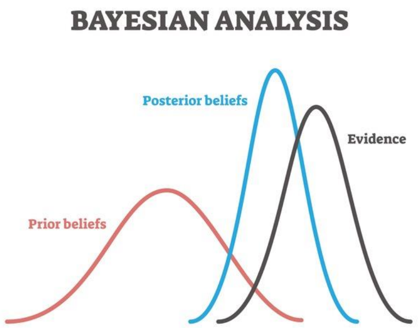
Different Styles of AI

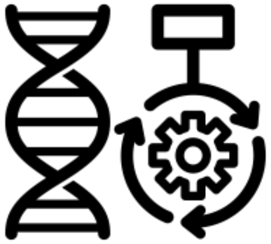
Five Tribes of Machine Learning

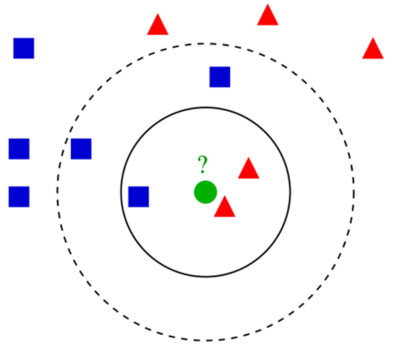
Symbolists	Connectionists	Bayesians	Evolutionaries	Analogizers









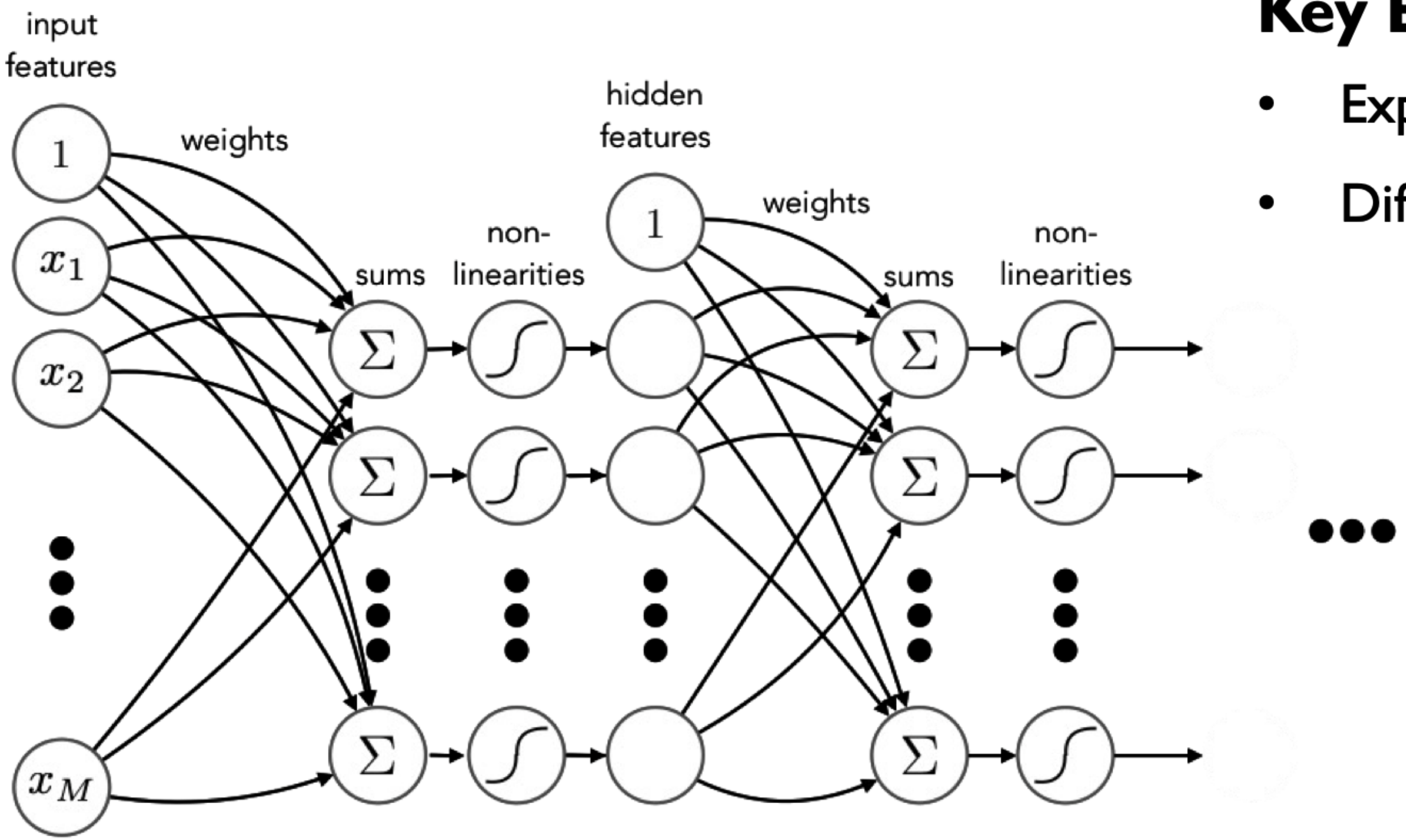


"The Master Algorithm", Pedro Domingos

How about combing PP with DL?

- Making neural networks Bayesian
  - Bayesian neural networks
- Using neural networks to compute probabilistic programs
  - Edwards
- Treat neural networks as input to probabilistic programs
  - Neural-symbolic programming

Deep Neural Networks

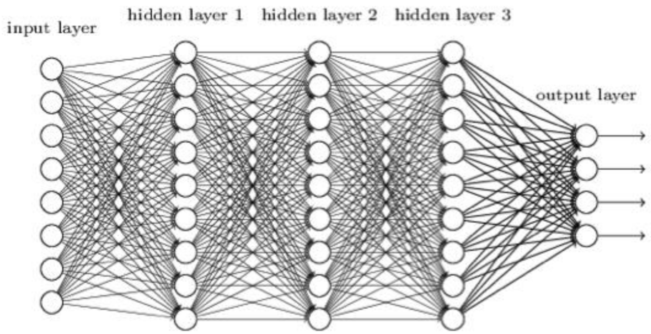


Key Benefits:

- Expressiveness
- Differentiable Learning

Slide by Joe Marino

Weaknesses of Deep Learning



Uninterpretable → Hard to trust/control

**Lack of domain knowledge →
Unreliable training, high sample complexity**

Opaque inductive bias → Brittle model

Slide by Chaudhuri, Sun, Solar-Lezama

Neurosymbolic Programs

Symbolic Programs

Interpretable

Verifiable

Structured domain knowledge

Data efficient

Neural Networks

Scalable algorithms

Flexible

Handles messy data

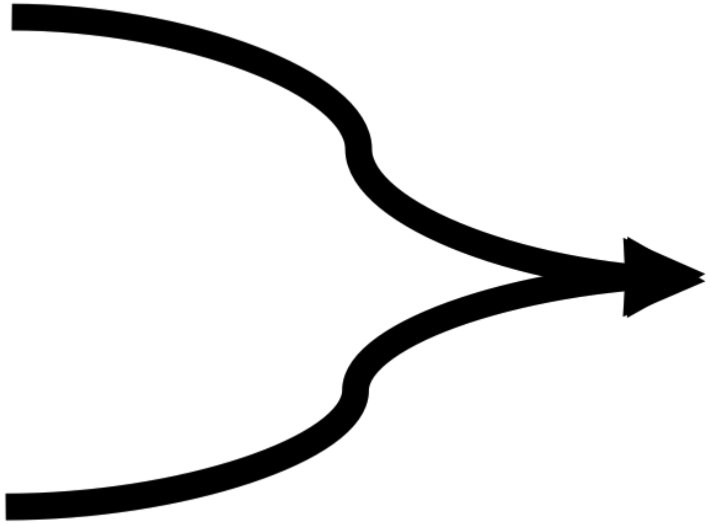
Easy to get started

Slide by Chaudhuri, Sun, Solar-Lezama

Slide by Chaudhuri, Sun, Solar-Lezama

Neurosymbolic Programming

Neurosymbolic Function Representations (Symbolic code + neural networks)



Neurosymbolic Learning Algorithms

Neurosymbolic Programming

Neurosymbolic Programming. Chaudhuri, Ellis, Polozov, Singh, Solar-Lezama, Yue. Foundations and Trends in Programming Languages, 2021.

Neurosymbolic Programming

Swarat Chaudhuri
UT Austin
swarat@cs.utexas.edu

Kevin Ellis
Cornell
kellis@cornell.edu

Oleksandr Polozov
Google
polozov@google.com

Rishabh Singh
Google
rising@google.com

Armando Solar-Lezama
MIT
asolar@csail.mit.edu

Yisong Yue
Caltech
yyue@caltech.edu

Neural-Symbolic Programing in PP

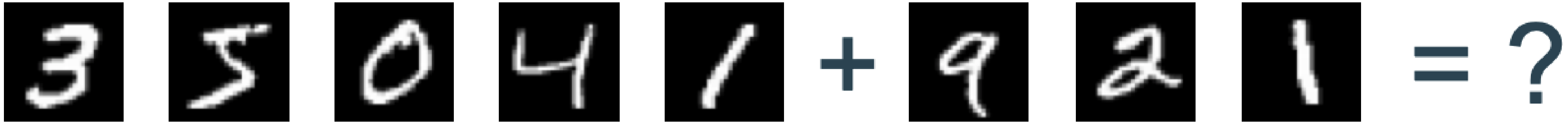
- Treating neural networks as part of probabilistic programs' inputs
  - DeepProblog [Manhaeve et al., NeurIPS'18]
  - Scallop [Li et al., PLDI'23]

- Key Challenge: End-to-end training

DeepProbLog

Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, Luc De Raedt: DeepProbLog: Neural Probabilistic Logic Programming. NeurIPS 2018: 3753-3763

Example Task: MNIST Addition



What if we only labeled sums, not single digits?

DeepProbLog Program for MNIST Addition

```
nn(m_digit,[X],Y,[0,1,2,3,4,5,6,7,8,9]) :: digit(X,Y).

addition(X,Y,Z) :- digit(X,X2), digit(Y,Y2),digit(Z,Z2), Z2 is X2+Y2.
```

Neural Annotated Disjunctions

Keyword | Neural Network | Input Variables | Output Variable | Output Domain

nn(m_digit, [X], Y, [0, . . . , 9]) :: digit(X, Y). — Neural Annotated Disjunction

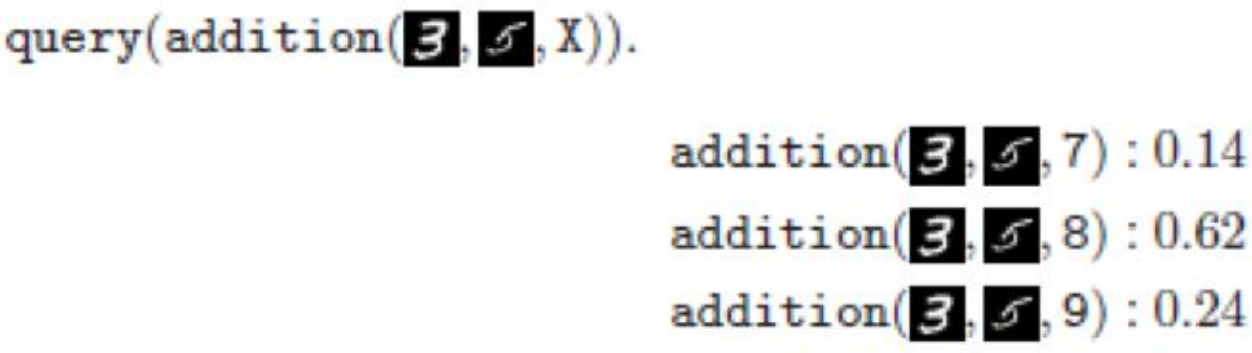
nn(m_digit, [], 0) :: digit(, 0) ; . . . ; nn(m_digit, [], 9) :: digit(, 9). — Grounded Neural Annotated Disjunction

p_0 :: digit(, 0) ; . . . ; p_9 :: digit(, 9). — Grounded Annotated Disjunction

DeepProbLog Program for MNIST Addition

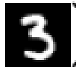
nn(m_digit,[X],Y,[0,1,2,3,4,5,6,7,8,9]) :: digit(X,Y).

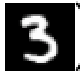
addition(X,Y,Z) :- digit(X,X2), digit(Y,Y2),digit(Z,Z2), Z2 is X2+Y2.



Neural Facts

`nn(m, [X, Y]) :: similar(X, Y).`

`nn(m, [`, `]) :: similar(`, `).`

`p :: similar(`, `).`

Summary of DeepProblog Syntax

- Problog + Neural Annotated Disjunctions + Neural Facts

$$\texttt{nn}(\texttt{m\_digit}, [\texttt{X}], \texttt{Y}, [\texttt{0}, \ldots, \texttt{9}]) :: \texttt{digit}(\texttt{X}, \texttt{Y}).$$

$$\texttt{nn}(\texttt{m}, [\texttt{X}, \texttt{Y}]) :: \texttt{similar}(\texttt{X}, \texttt{Y}).$$

Inference of DeepProblog

- After grounding the neural parts: nothing special
  - Run neural networks
  - Run Problog

Learning of DeepProblog: Problem

Definition 5

Learning from entailment Given a DeepProbLog program with parameters Θ, a set $\mathcal{Q}$ of pairs (q, p) with q a query and p its desired success probability, and a loss function $\mathcal{L}$, compute:

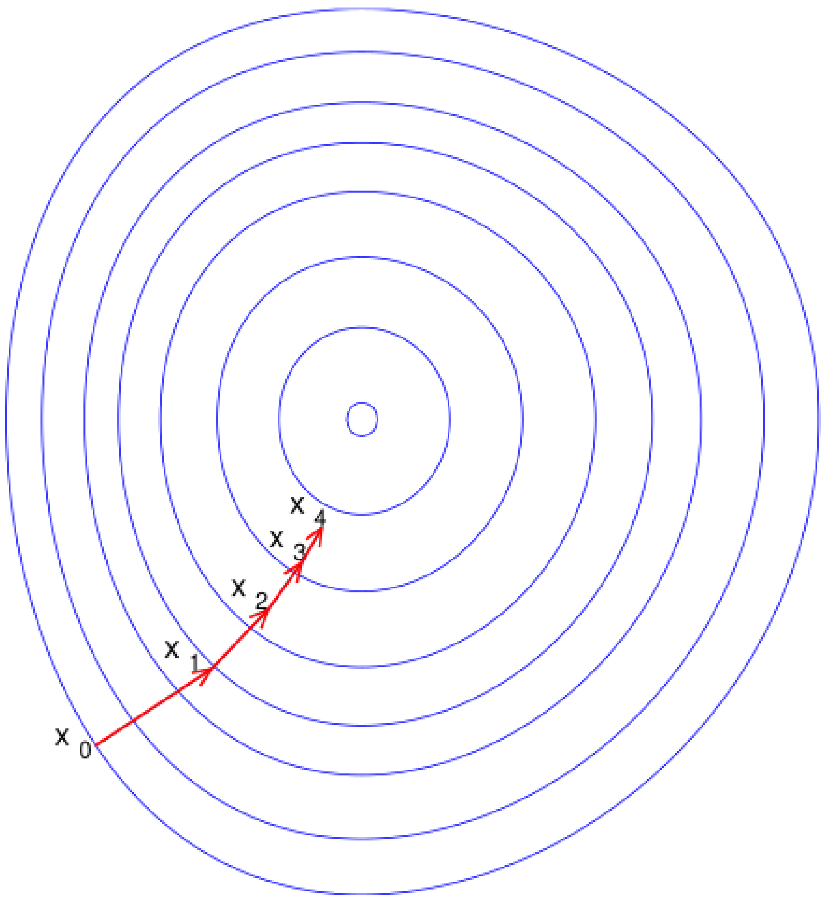
$$\arg\min_{\Theta} \frac{1}{|\mathcal{Q}|} \sum_{(q,p)\in\mathcal{Q}} \mathcal{L}(P(q|\Theta), p)$$

Assuming desired probability $p = 1$, the problem reduces to

$$\arg\min_{\Theta} \frac{1}{|\mathcal{Q}|} \sum_{(q,p)\in\mathcal{Q}} -\log P_{\Theta}(q)$$

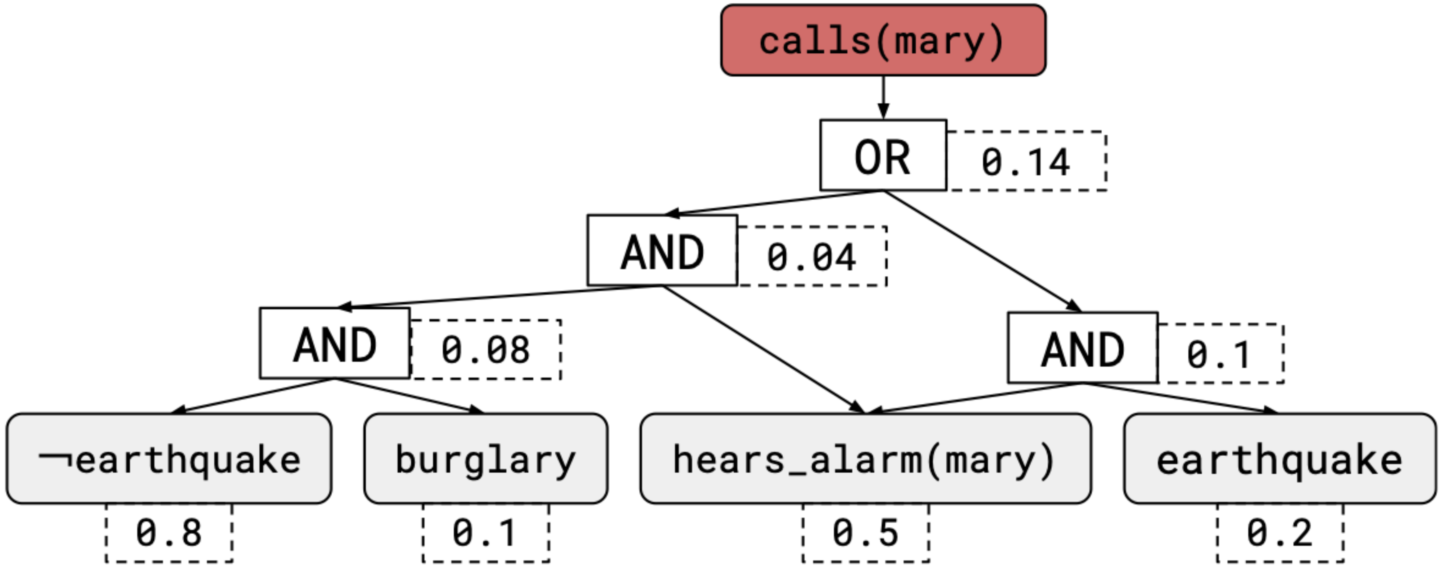
Learning of DeepProblog: Gradient Descent

To minimize $F(x)$: $\mathbf{a}_{n+1} = \mathbf{a}_n - \gamma \nabla F(\mathbf{a}_n)$



Gradient Descent in Problog

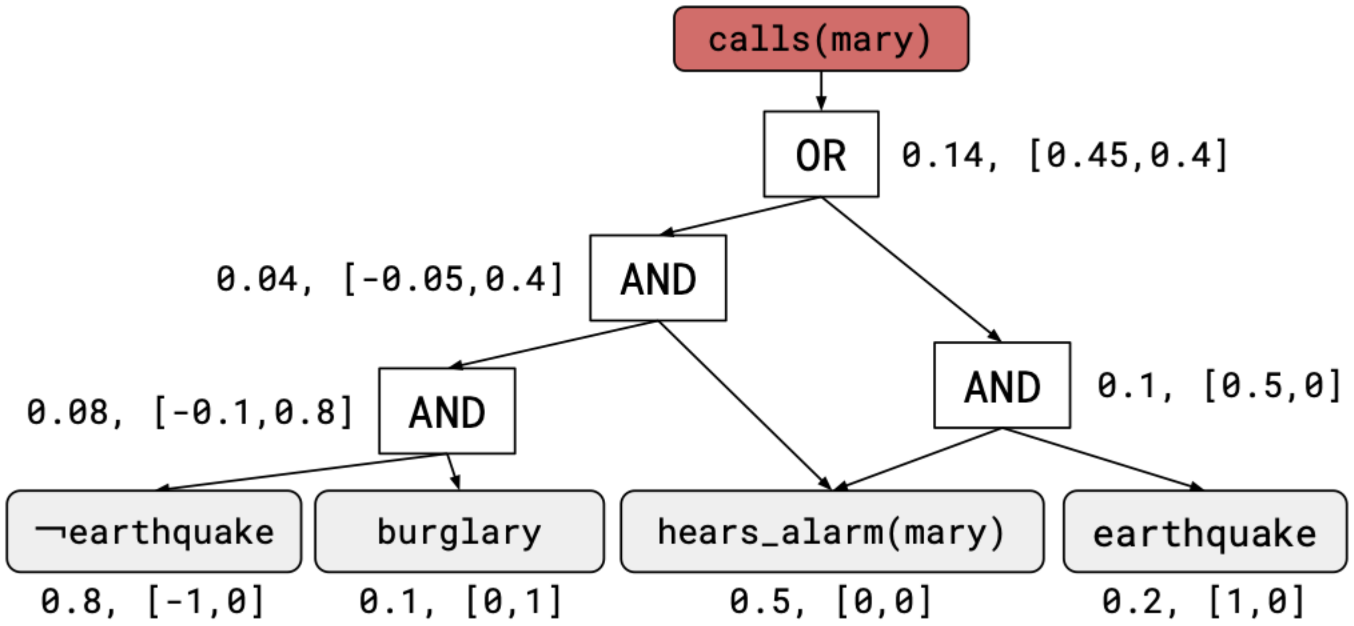
```
0.2::earthquake.
0.1::burglary.
0.5::hears_alarm(mary).
0.4::hears_alarm(john).
alarm :- earthquake.
alarm :- burglary.
calls(X):-alarm,hears_alarm(X).
```



Algebraic Circuit
(Support Efficient Inference based on BDD)

Gradient Descent in Problog

```
0.2::earthquake.
0.1::burglary.
0.5::hears_alarm(mary).
0.4::hears_alarm(john).
alarm :- earthquake.
alarm :- burglary.
calls(X):-alarm,hears_alarm(X).
```



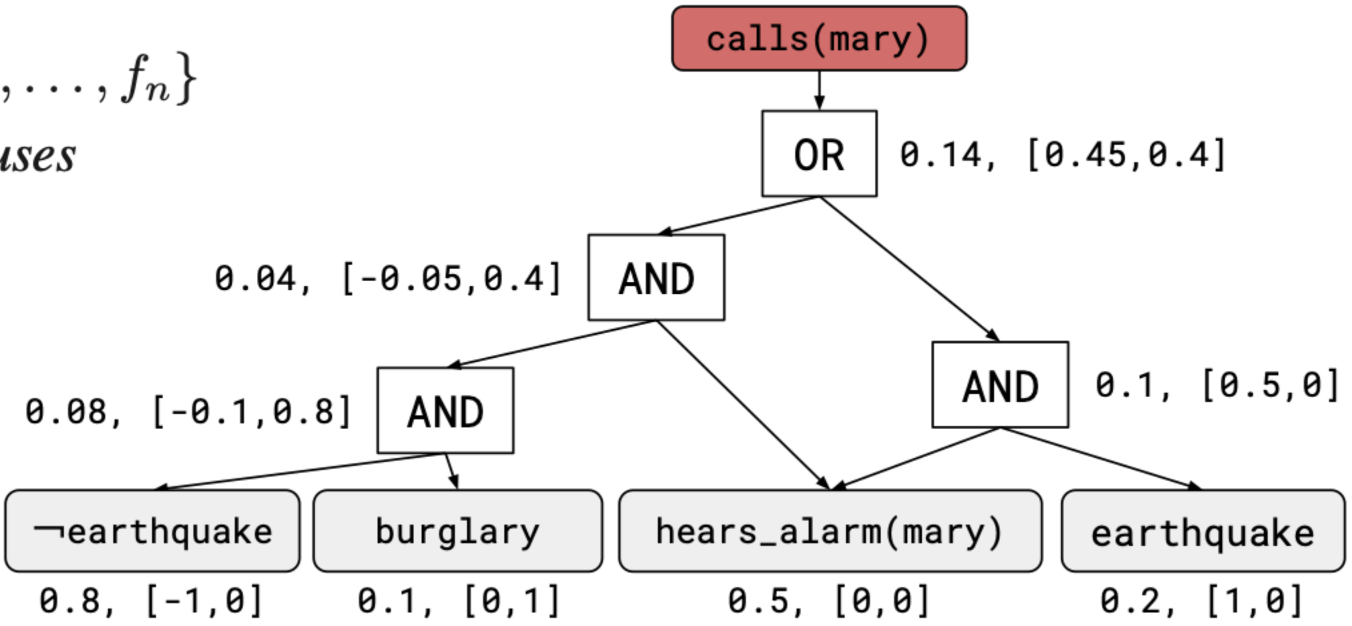
$$\arg\min_{\Theta} \frac{1}{|\mathcal{Q}|} \sum_{(q,p)\in\mathcal{Q}} -\log P_{\Theta}(q)$$

Algebraic Prolog

An algebraic Prolog (aProbLog) program consists of

- a *commutative semiring* $(\mathcal{A}, \oplus, \otimes, e^{\oplus}, e^{\otimes})$[1]
- a finite set of ground *algebraic facts* $\mathrm{F} = \{f_1, \ldots, f_n\}$
- a finite set BK of *background knowledge clauses*
- a *labeling function* $\alpha : \mathrm{L(F)} \rightarrow \mathcal{A}$

$$(a_1, \vec{a_2}) \oplus (b_1, \vec{b_2}) = (a_1 + b_1, \vec{a_2} + \vec{b_2})$$
$$(a_1, \vec{a_2}) \otimes (b_1, \vec{b_2}) = (a_1 b_1, b_1 \vec{a_2} + a_1 \vec{b_2})$$
$$e^{\oplus} = (0, \vec{0})$$
$$e^{\otimes} = (1, \vec{0})$$

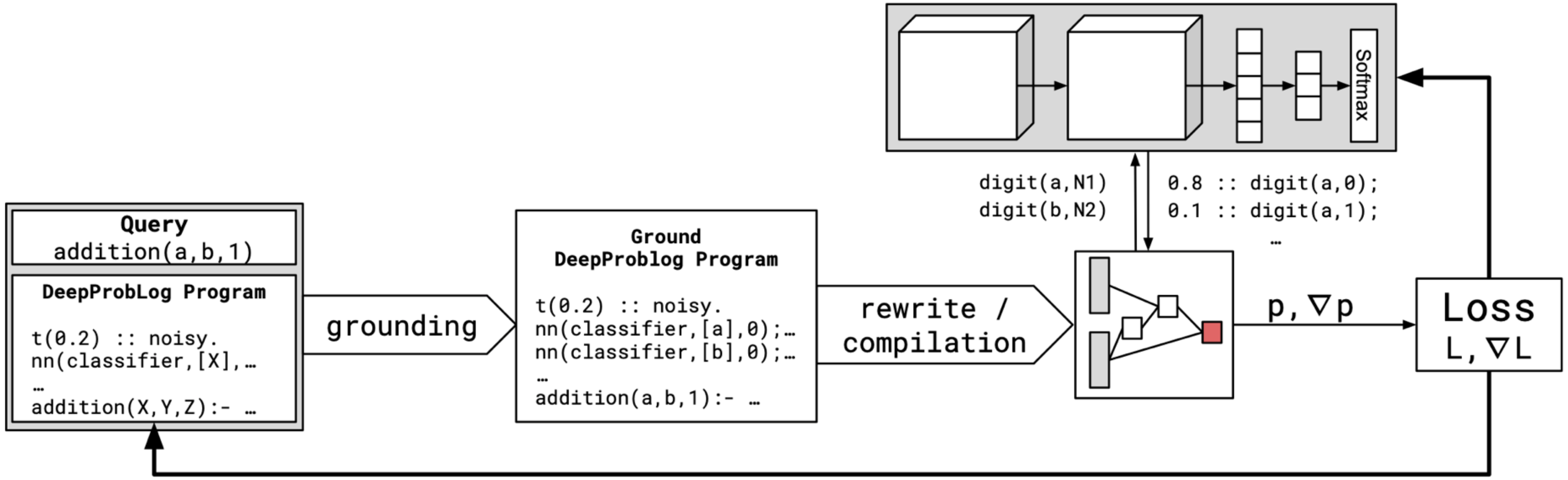


Algebraic Circuit
(Support Efficient Inference based on BDD)

Gradient Descent in DeepProbLog

$$\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx}$$

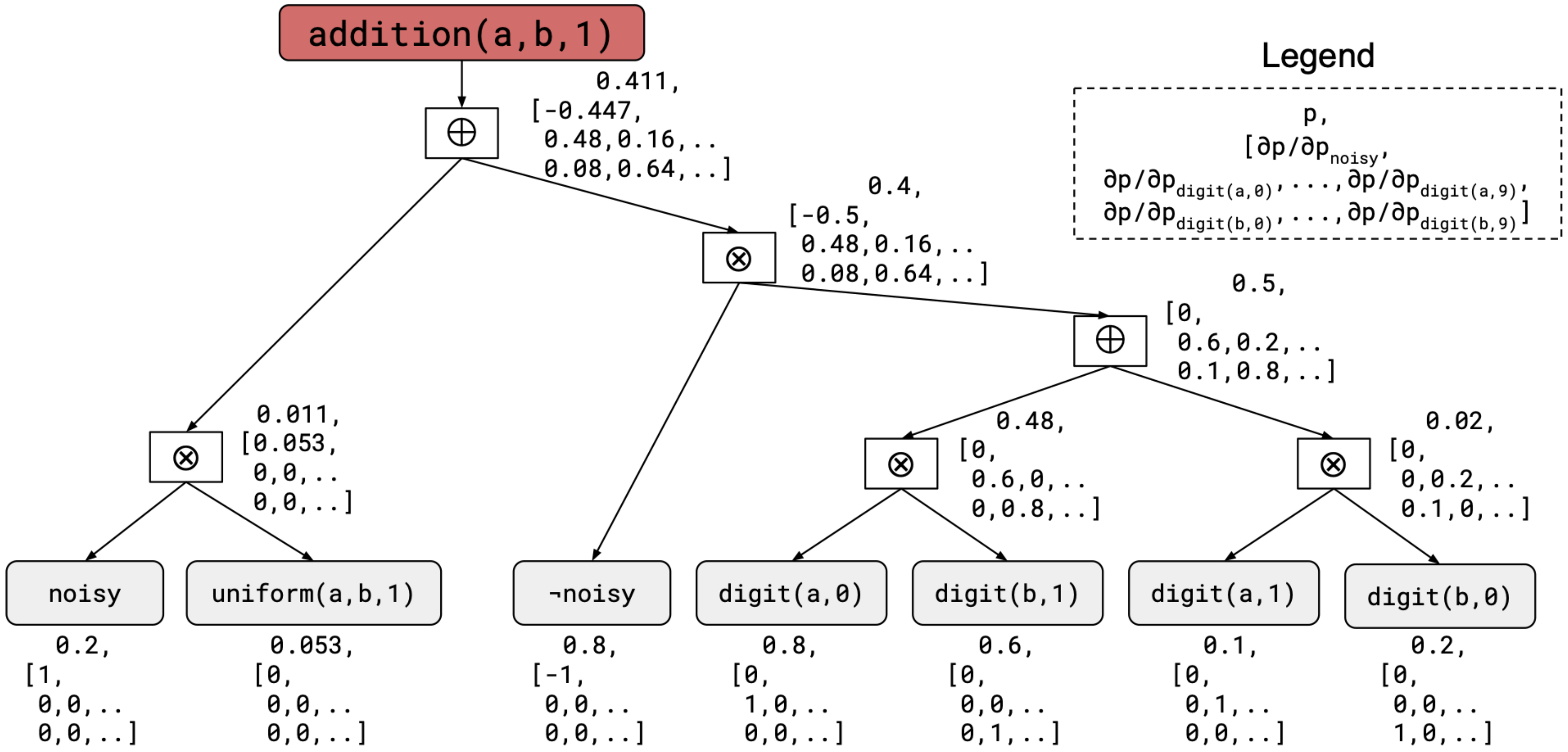
$$\frac{d\mathcal{L}}{d\theta_k} = \frac{\partial \mathcal{L}}{\partial P(q)} \sum_i \frac{\partial P(q)}{\partial \hat{p}_i} \frac{\partial \hat{p}_i}{\partial \theta_k}$$



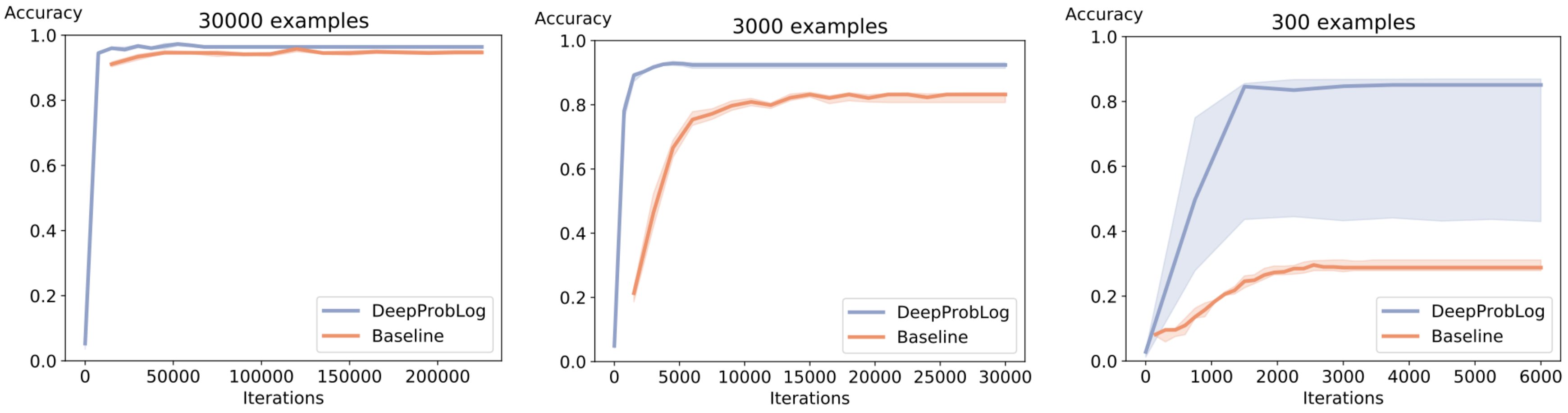
Gradient Descent in DeepProbLog

```
nn(classifier, [X], Y, [0 .. 9]) :: digit(X,Y).
t(0.2) :: noisy.

1/19 :: uniform(X,Y,0) ; ... ; 1/19 :: uniform(X,Y,18).

addition(X,Y,Z) :- noisy, uniform(X,Y,Z).
addition(X,Y,Z) :- \+noisy, digit(X,N1), digit(Y,N2), Z is N1+N2.
```

Gradient Descent in DeepProbLog



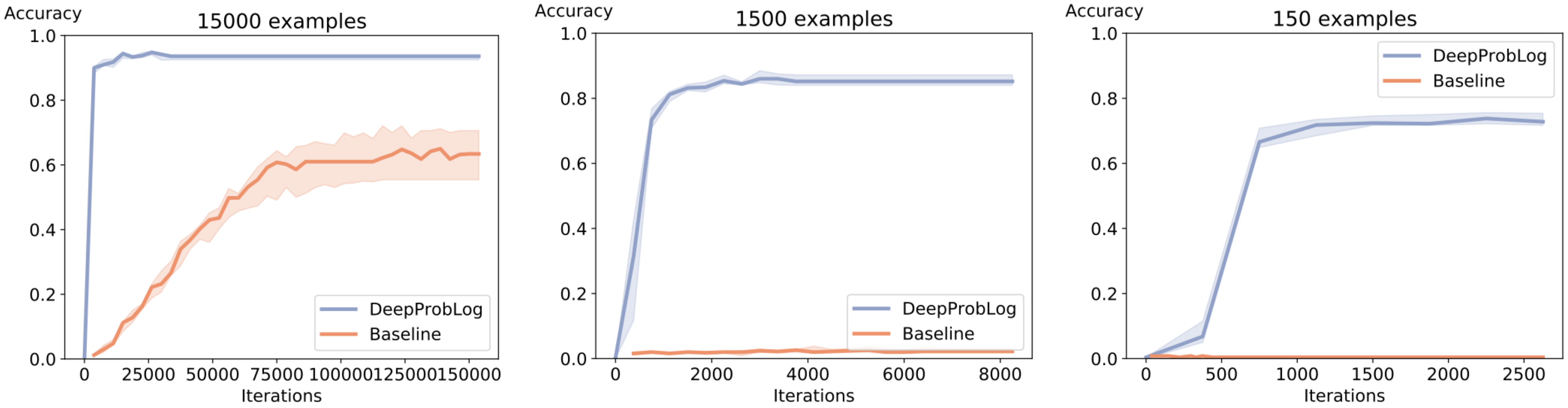
Evaluation



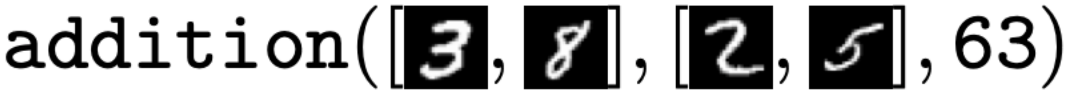
(a) 30 000 examples (b) 3 000 examples (c) 300 examples

Baseline: CNN

Evaluation



(a) 15 000 examples (b) 1 500 examples (c) 150 examples



Baseline: CNN

Conclusion

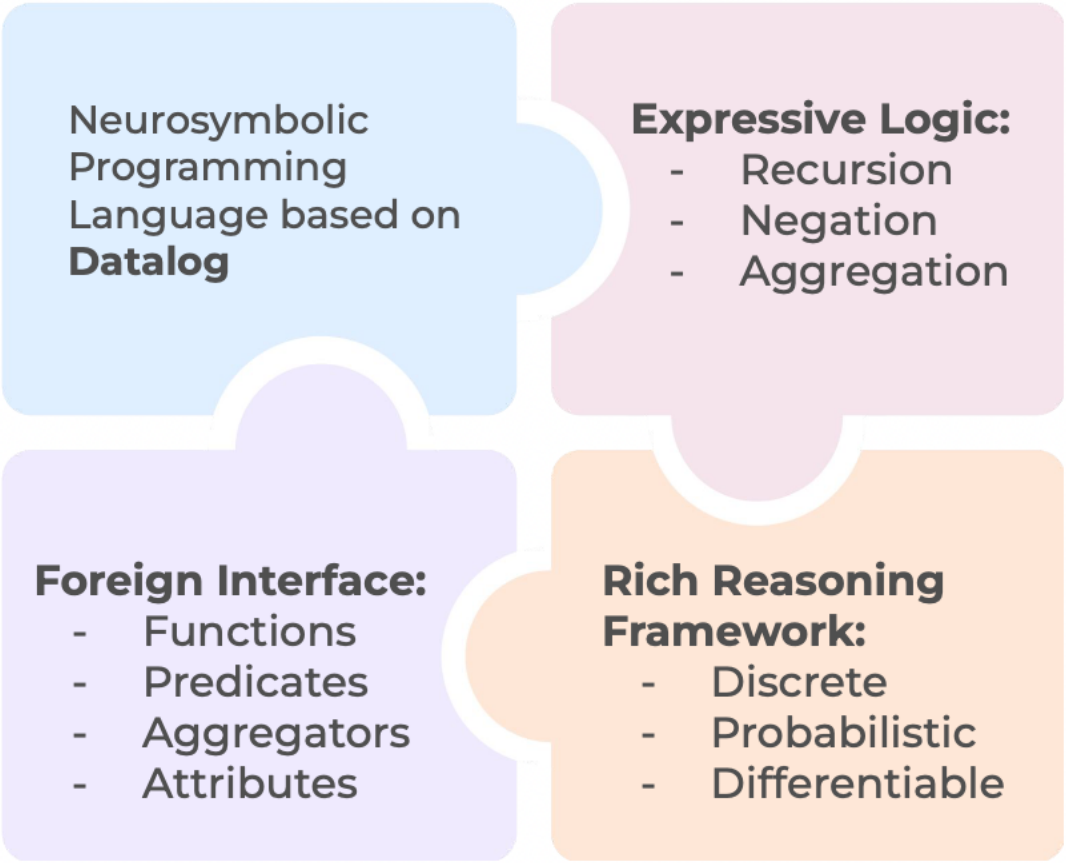
- DeepProbLog = ProbLog + Neural Networks
- Inference: Trivial, neural networks outputs as input distributions
- Learning
  - Chain rule
  - Algebraic Prolog enables efficient gradient computation

Scallop

Scallop: A Language for Neurosymbolic Programming. Proc. ACM Program. Lang. 7(PLDI): 1463-1487 (2023)

Part of the slides are from Ziyang Li's PKU talk

Scallop’s Goal



Differences from ProbLog

- Better engineered

- More efficient in training

- Overall more practical

A Motivating Example

PacMan-Maze

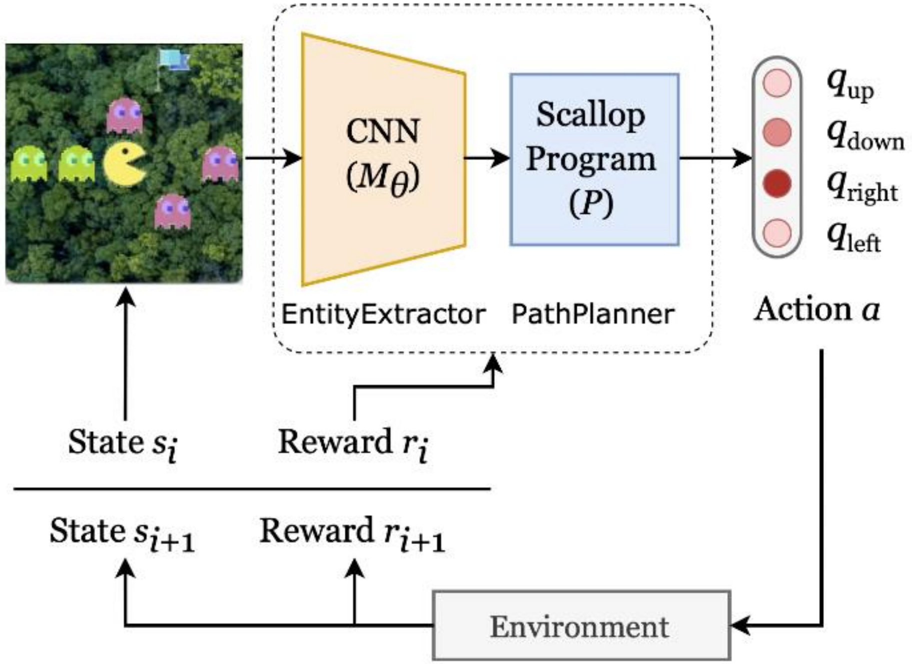
Step 0 Step 4 Step 7

State: 200x200 colored image
Action: Up, Down, Left, Right
(Environments are 5x5 grids randomized for each session)

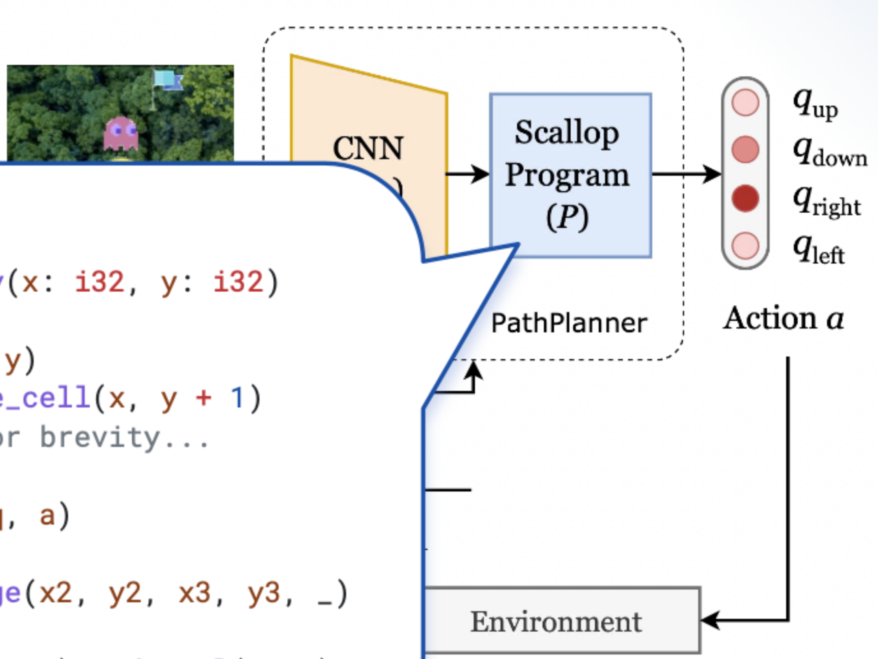
A Motivating Example

State: 200x200 colored image
Action: Up, Down, Left, Right

(Environments are 5x5 grids randomized for each session)



A Motivating Example




```
type Action = UP | RIGHT | DOWN | LEFT
type actor(x: i32, y: i32), goal(x: i32, y: i32), enemy(x: i32, y: i32)

rel safe_cell(x, y) = grid_cell(x, y) and not enemy(x, y)
rel edge(x, y, x, y + 1, UP) = safe_cell(x, y) and safe_cell(x, y + 1)
// Rules for RIGHT, DOWN, and LEFT edges are omitted for brevity...

rel next_pos(p, q, a) = actor(x, y) and edge(x, y, p, q, a)
rel path(x, y, x, y) = next_pos(x, y, _)
rel path(x1, y1, x3, y3) = path(x1, y1, x2, y2) and edge(x2, y2, x3, y3, _)

rel next_action(a) = next_pos(p, q, a) and path(p, q, r, s) and goal(r, s)
```

A Motivating Example

Step 0 Step 4 Step 7

State: 200x200 colored image
Action: Up, Down, Left, Right

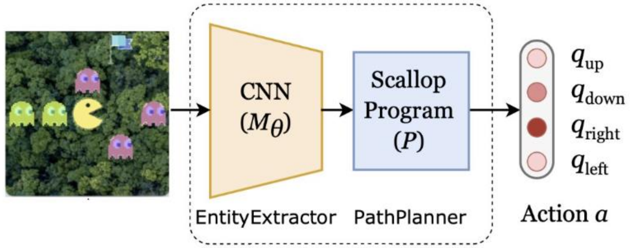
(Environments are 5x5 grids randomized for each session)

	Neurosymbolic (with Scallop)	**DQN**
Success rate (reaches the goal within 50 steps)	**99.4%**	84.9%
# of Training episodes (to achieve the success rate)	**50**	50K

(Note: this is not entirely a fair comparison since our Scallop program encodes system dynamics and human knowledge)

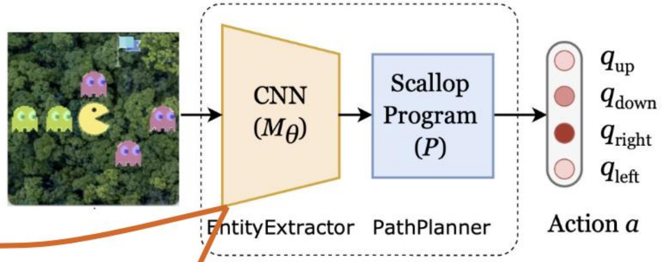
Integration with PyTorch

```python
class PolicyNet(nn.Module):
  def __init__(self, grid_x, grid_y, cell_pixel_size):
    super(PolicyNet, self).__init__()
    self.grid_x, self.grid_y = grid_x, grid_y
    self.cells = [(x, y) for x in range(grid_x) for y in range(grid_y)]

    # Setup CNNs that process the image and extract features
    self.extract_entity = EntityExtractor(grid_x, grid_y, cell_pixel_size)

    # Setup scallop context and scallop forward functions
    self.path_planner = scallopy.Module(
      file="pacman_agent.scl",
      provenance="difftopkproofs", k=1,
      facts={"grid_node": [(torch.tensor(args.attenuation, requires_grad=False), c) for c in self.cells]},
      input_mappings={"actor": self.cells, "goal": self.cells, "enemy": self.cells},
      output_mappings={"next_action": list(range(4))})

  def forward(self, x):
    actor, goal, enemy = self.extract_entity(x)
    result = self.path_planner(actor=actor, goal=goal, enemy=enemy)
    return torch.softmax(result["next_action"], dim=1)
```



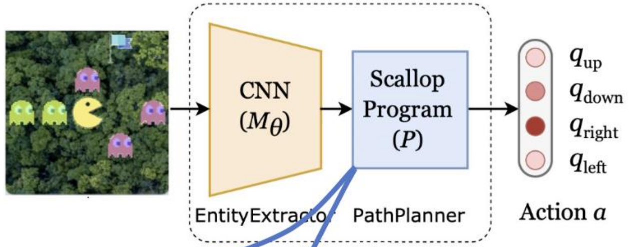
Integration with PyTorch

```
class PolicyNet(nn.Module):
  def __init__(self, grid_x, grid_y, cell_pixel_size):
    super(PolicyNet, self).__init__()
    self.grid_x, self.grid_y = grid_x, grid_y
    self.cells = [(x, y) for x in range(grid_x) for y in range(grid_y)]

    # Setup CNNs that process the image and extract features
    self.extract_entity = EntityExtractor(grid_x, grid_y, cell_pixel_size)

    # Setup scallop context and scallop forward functions
    self.path_planner = scallopy.Module(
      file="pacman_agent.scl",
      provenance="difftopkproofs", k=1,
      facts={"grid_node": [(torch.tensor(args.attenuation, requires_grad=False), c) for c in self.cells]},
      input_mappings={"actor": self.cells, "goal": self.cells, "enemy": self.cells},
      output_mappings={"next_action": list(range(4))})

  def forward(self, x):
    actor, goal, enemy = self.extract_entity(x)
    result = self.path_planner(actor=actor, goal=goal, enemy=enemy)
    return torch.softmax(result["next_action"], dim=1)
```



Integration with PyTorch

```
class PolicyNet(nn.Module):
  def __init__(self, grid_x, grid_y, cell_pixel_size):
    super(PolicyNet, self).__init__()
    self.grid_x, self.grid_y = grid_x, grid_y
    self.cells = [(x, y) for x in range(grid_x) for y in range(grid_y)]

    # Setup CNNs that process the image and extract features
    self.extract_entity = EntityExtractor(grid_x, grid_y, cell_pixel_size)

    # Setup scallop context and scallop forward functions
    self.path_planner = scallopy.Module(
      file="pacman_agent.scl",
      provenance="difftopkproofs", k=1,
      facts={"grid_node": [(torch.tensor(args.attenuation, requires_grad=False), c) for c in self.cells]},
      input_mappings={"actor": self.cells, "goal": self.cells, "enemy": self.cells},
      output_mappings={"next_action": list(range(4))})

  def forward(self, x):
    actor, goal, enemy = self.extract_entity(x)
    result = self.path_planner(actor=actor, goal=goal, enemy=enemy)
    return torch.softmax(result["next_action"], dim=1)
```



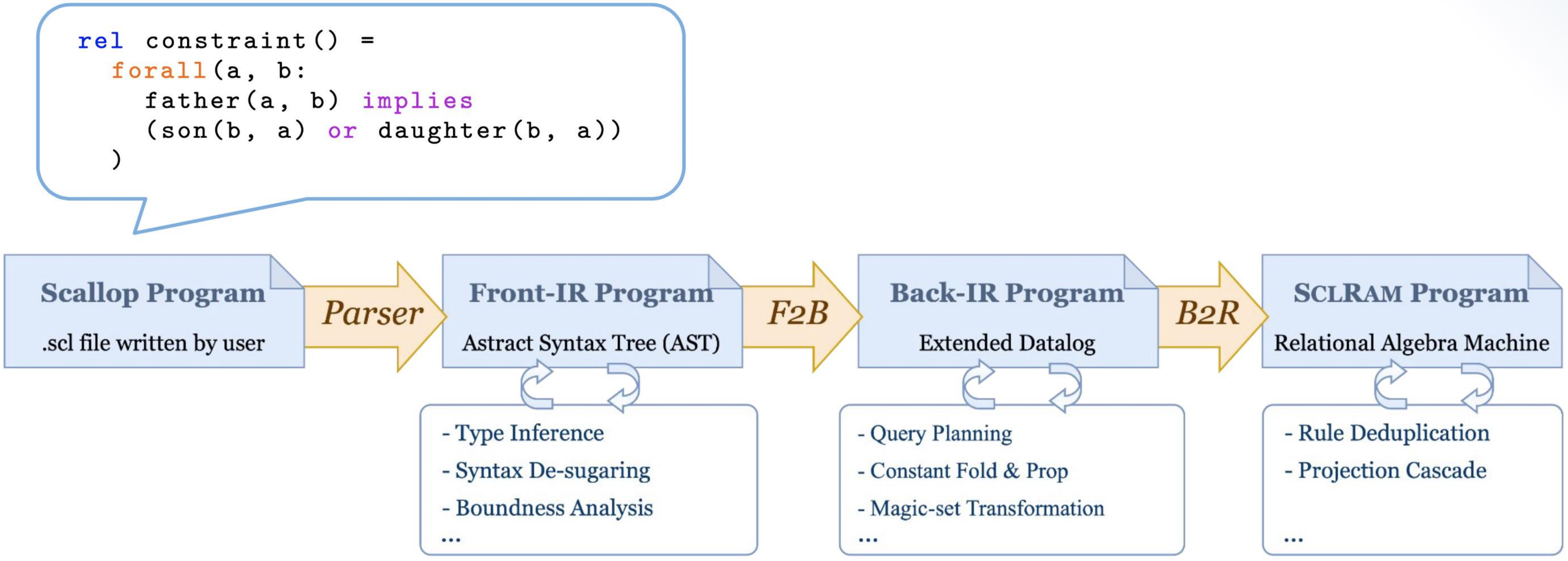
Support for Probabilities

```
rel 0.03::earthquake()
```

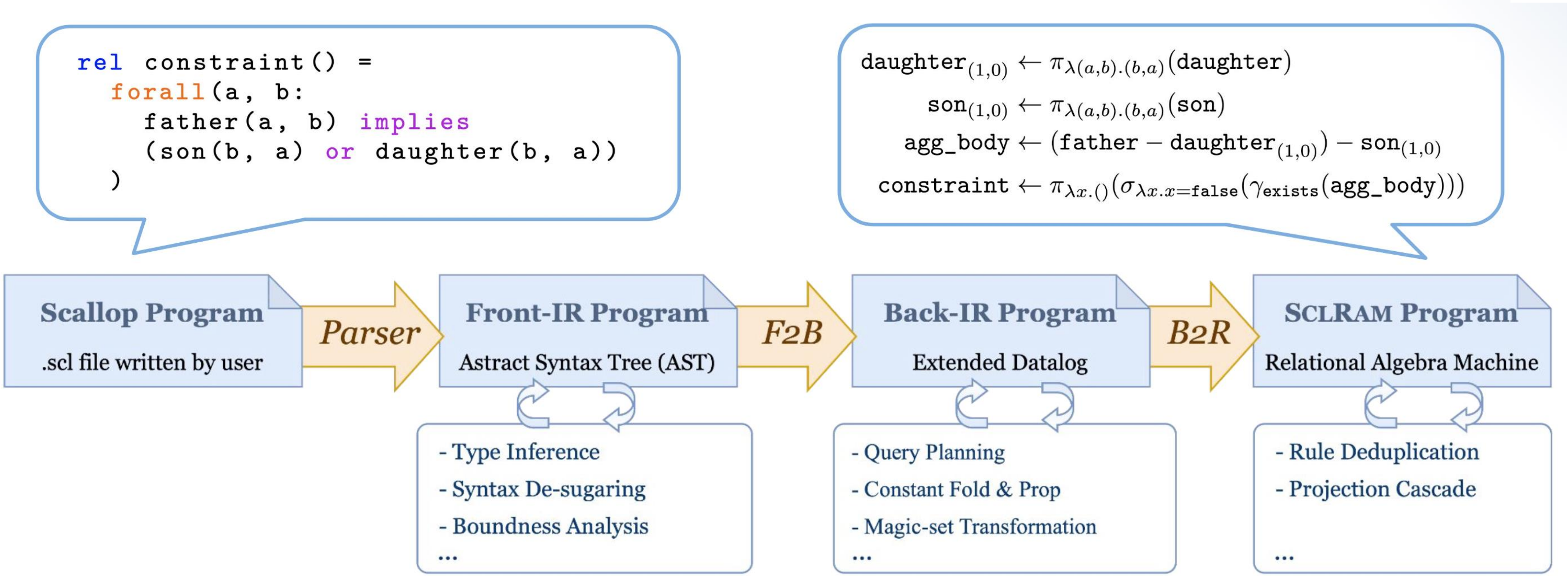
```
rel 0.20::burglary()
```

```
rel alarm() = earthquake() or burglary()
query alarm
```

Scallop's Compiler Architecture



Scallop's Compiler Architecture



Semantics and Provance Framework

- The formal semantics of SCLRAM is parameterized by a provenance structure inspired by the theory of Provenance Semirings [PODS'07]

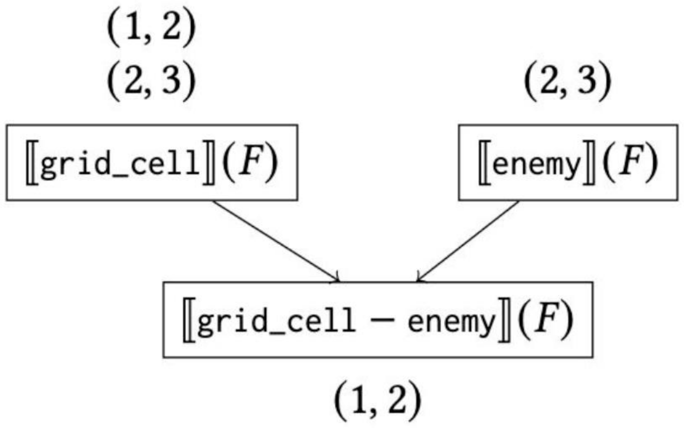
- A Provenance Structure is an algebraic structure that specifies:
  - Tag Space: the space of additional information associated with each tuple
  - Operations: how tags propagate during execution

	Abstract Provenance			max-min-prob(mmp)
(Tag Space)	t	$\in$	T	$[0, 1]$
(False)	$\mathbf{0}$	$\in$	T	0
(True)	$\mathbf{1}$	$\in$	T	1
(Disjunction)	$\oplus$	:	$T \times T \to T$	max
(Conjunction)	$\otimes$	:	$T \times T \to T$	min
(Negation)	$\ominus$	:	$T \rightharpoonup T$	$\lambda p.(1 - p)$
(Saturation)	$\circleddash$	:	$T \times T \to \text{Bool}$	==

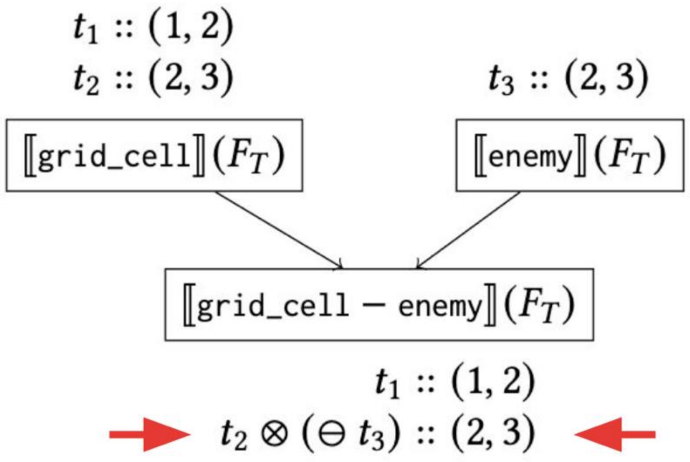
Semantics and Provance Framework

Scallop program `rel safe_cell(x, y) = grid_cell(x, y) and not enemy(x, y)`

SclRam program `safe_cell ← grid_cell - enemy`



vs.



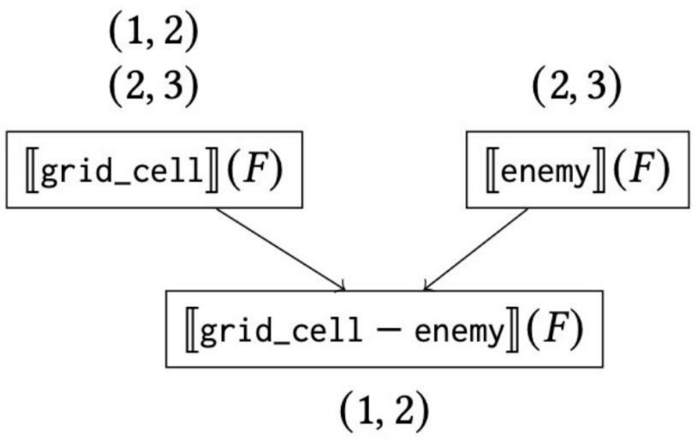
Untagged Semantics

Tagged Semantics

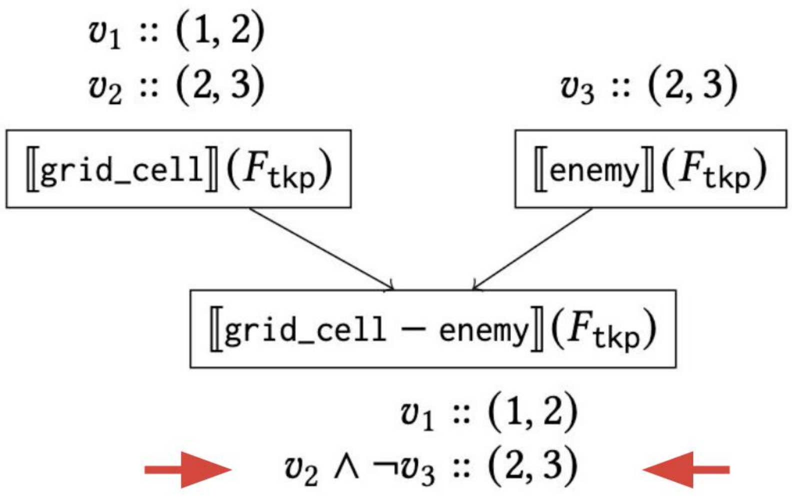
Semantics and Provance Framework

Scallop program `rel safe_cell(x, y) = grid_cell(x, y) and not enemy(x, y)`

SclRam program `safe_cell ← grid_cell - enemy`



vs.



Untagged Semantics

Tagged Semantics with `top-k-proofs`

Semantics and Provance Framework

Scallop program

```
rel safe_cell(x, y) = grid_cell(x, y) and not enemy(x, y)
```

SclRam program

```
safe_cell ← grid_cell - enemy
```

Recover Probability from Bool Formula

Using Weighted Model Counting (WMC)

$$\Pr(v_1) = 0.9, \Pr(v_2) = 0.9, \Pr(v_3) = 0.2$$

$$\Pr(v_2 \land \neg v_3) = \Pr(v_2) \cdot (1 - \Pr(v_3))$$
$$= 0.9 \cdot (1 - 0.2)$$
$$= 0.72$$

$0.9 \quad v_1 :: (1, 2)$

$0.9 \quad v_2 :: (2, 3)$ $\qquad 0.2 \quad v_3 :: (2, 3)$

$$\begin{array}{ccc} [\![\texttt{grid\_cell}]\!](F_{\text{tkp}}) & & [\![\texttt{enemy}]\!](F_{\text{tkp}}) \\ & \searrow \quad \swarrow & \\ & [\![\texttt{grid\_cell} - \texttt{enemy}]\!](F_{\text{tkp}}) & \end{array}$$

$v_1 :: (1, 2)$

$v_2 \land \neg v_3 :: (2, 3)$

Tagged Semantics with `top-k-proofs`

Built-in Library of Provenance Structures

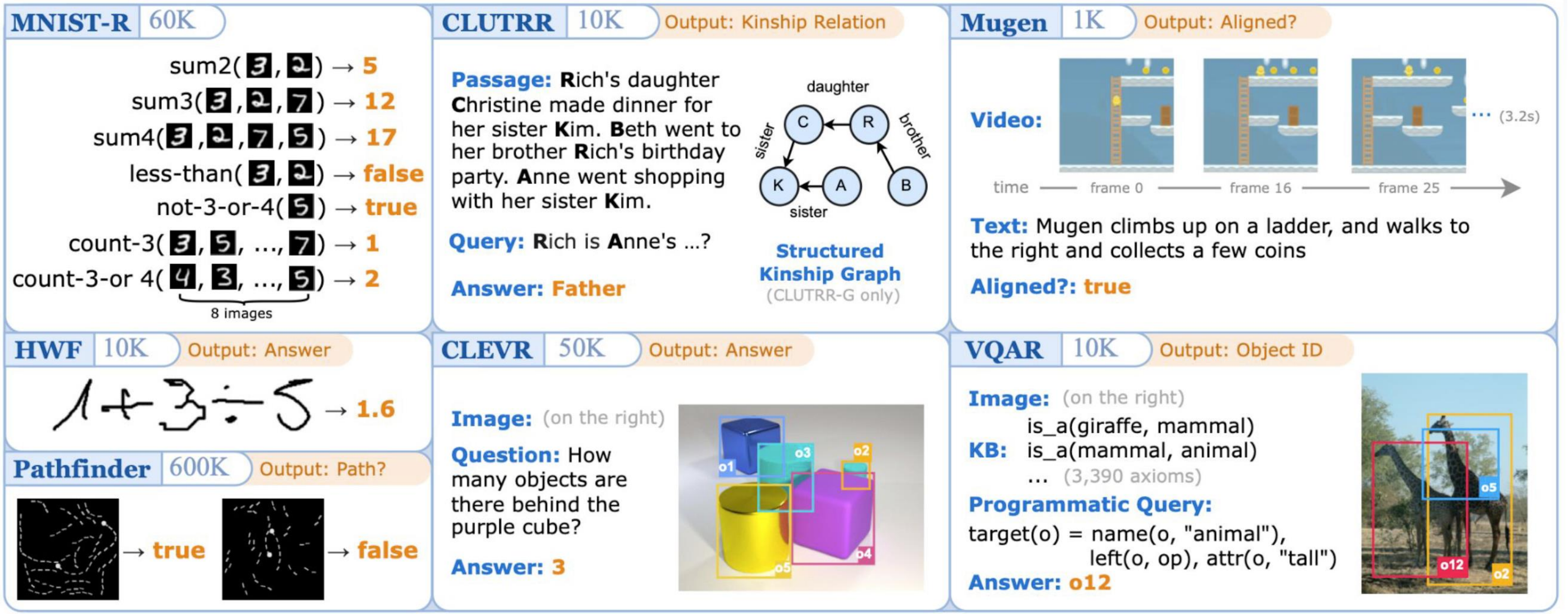
Kind	Provenance	T	$\mathbf{0}$	$\mathbf{1}$	$\oplus$	$\otimes$	$\ominus$	$\circleddash$	τ	ρ
Discrete	`unit`	$\{()\}$	$()$	$()$	$\lambda t_1, t_2.()$	$\lambda t_1, t_2.()$	$\lambda a.\text{FAIL}$	$==$	$\lambda i.()$	$\lambda t.()$
	`bool`	$\{\top, \bot\}$	$\bot$	$\top$	$\vee$	$\wedge$	$\neg$	$==$	id	id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

Built-in Library of Provenance Structures

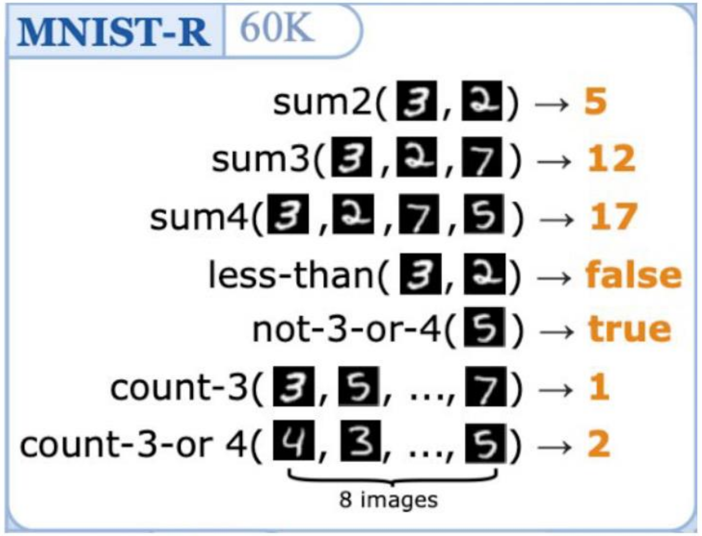
Kind	Provenance	T	$\mathbf{0}$	$\mathbf{1}$	$\oplus$	$\otimes$	$\ominus$	$\circleddash$	τ	ρ
Discrete	unit	$\{()\}$	$()$	$()$	$\lambda t_1, t_2.()$	$\lambda t_1, t_2.()$	$\lambda a.\mathrm{FAIL}$	$==$	$\lambda i.()$	$\lambda t.()$
	bool	$\{\top, \bot\}$	$\bot$	$\top$	$\vee$	$\wedge$	$\neg$	$==$	id	id
	natural	$\mathbb{N}$	0	1	$+$	$\times$	$\lambda n.\mathbb{1}[n > 0]$	$==$	id	id
Probabil...										
Differentiable	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	Φ	$\emptyset$	$\{\emptyset\}$	$\vee_{\text{top-}k}$	$\wedge_{\text{top-}k}$	$\neg_{\text{top-}k}$	$==$	$\lambda \hat{p}_i.\{\{\mathrm{pos}(i)\}\}$	$\lambda \varphi.\mathrm{WMC}(\varphi, \hat{\Gamma})$
	diff-sample-k-proofs	Φ	$\emptyset$	$\{\emptyset\}$	$\vee_{\text{sample-}k}$	$\wedge_{\text{sample-}k}$	$\neg_{\text{sample-}k}$	$==$	$\lambda \hat{p}_i.\{\{\mathrm{pos}(i)\}\}$	$\lambda \varphi.\mathrm{WMC}(\varphi, \hat{\Gamma})$
...	...	...	...	...	...	...	...	...	...	...

Syntax and semantics of Scallop programs remains familiar to users. The provenance framework allows to customize learning performance and scalability via a rich and extensible library.

Diverse Learning Tasks in Scallop

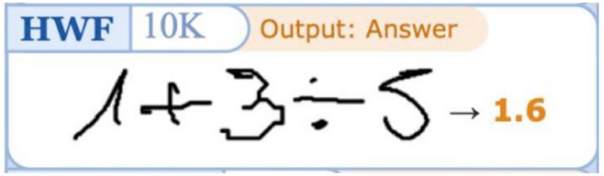


Diverse Learning Tasks in Scallop



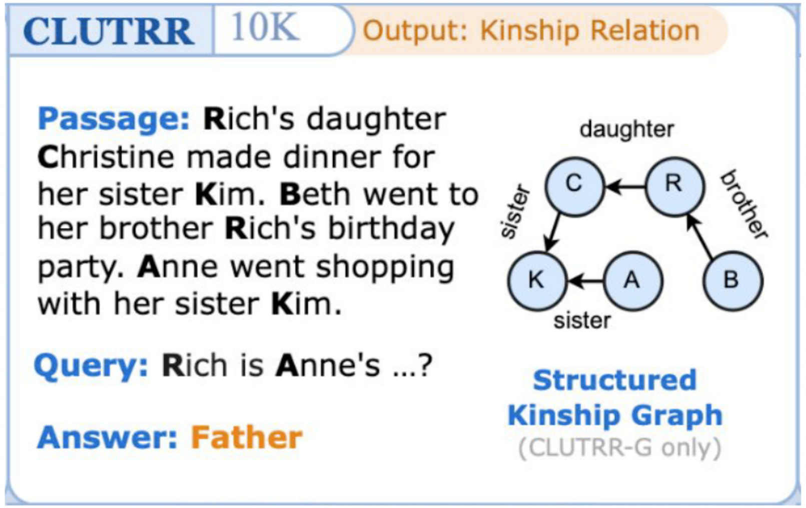

```
// summation of 2 numbers
rel sum_2(a + b) = digit_1(a), digit_2(b)

// summation of 3 numbers
rel sum_3(a + b + c) = digit_1(a), digit_2(b), digit_3(c)

// less_than between two digits
rel less_than(a < b) = digit_1(a), digit_2(b)

// Count the number of “3”-s in a list of digits
rel count_3 = count(i: digit(i, d) and d == 3)

// ...
```

Diverse Learning Tasks in Scallop



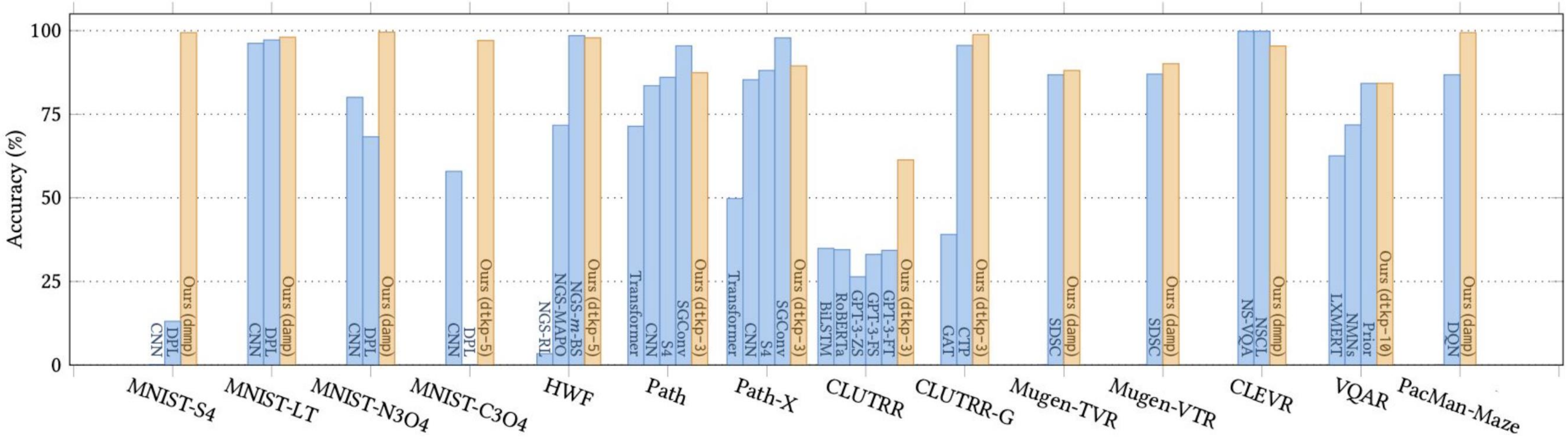

```
type symbol(index: usize, symbol: String)
type length(n: usize)

rel digit = {"0", "1", "2", "3", "4", "5", "6", "7", "8", "9"}

type factor(value: f32, begin: usize, end: usize)
rel factor(x as f32, b, b + 1) = symbol(b, x) and digit(x)

type mult_div(value: f32, begin: usize, end: usize)
rel mult_div(x, b, r) = factor(x, b, r)
rel mult_div(x * y, b, e) = mult_div(x, b, m) and symbol(m, "*") and factor(y, m + 1, e)
rel mult_div(x / y, b, e) = mult_div(x, b, m) and symbol(m, "/") and factor(y, m + 1, e)

type add_minus(value: f32, begin: usize, end: usize)
rel add_minus(x, b, r) = mult_div(x, b, r)
rel add_minus(x + y, b, e) = add_minus(x, b, m) and symbol(m, "+") and mult_div(y, m + 1, e)
rel add_minus(x - y, b, e) = add_minus(x, b, m) and symbol(m, "-") and mult_div(y, m + 1, e)

type result(value: f32)
rel result(y) = add_minus(y, 0, l) and length(l)

query result
```

Diverse Learning Tasks in Scallop




```
// Relationships and question extracted from the passage
type kinship(p1: String, p2: String, rela: String)
type question(p1: String, p2: String)

// Composition rules among relationships
rel composition = {
  ("daughter", "daughter", "granddaughter"),
  ("daughter", "sister", "daughter"),
  ("daughter", "son", "grandson"),
  // ...
}

// Transitive closure
rel relate(p1, p2, rela) = kinship(p1, p2, rela)
rel relate(p1, p3, r3) = relate(p1, p2, r1) and relate(p2, p3, r2)
  and composition(r2, r1, r3)

// Obtaining the result relationship
rel result(r) = question(p1, p2) and relate(p1, p2, r)
```

Performance: Scallop vs. Baselines



Testing Accuracy (%) on Selected Benchmark Tasks

Performance: Scallop vs. Baselines

Table 4. Runtime efficiency comparison on selected benchmark tasks. Numbers shown are average training time (sec.) per epoch. Our variants attaining the best accuracy are indicated in bold.

Task	Scallop				Baseline
	dmmp	damp	dtkp-3	dtkp-10	
sum2	**34**	88	72	185	21,430 (DPL)
sum3	**34**	**119**	71	1,430	30,898 (DPL)
sum4	**34**	154	77	4,329	timeout (DPL)
less-than	35	**42**	34	43	2,540 (DPL)
not-3-or-4	37	**33**	**33**	**34**	3,218 (DPL)
HWF	89	107	**120**	8,435	79 (NGS-m-BS)
CLEVR	**1,964**	1,618	2,325	timeout	–

Summary

- Probabilistic programming + neural nets = Symbolic reasoning + Neural reasoning
  - Probabilities can serve as a connector

- Challenge in learning
  - Algebraic method
  - Provenance method