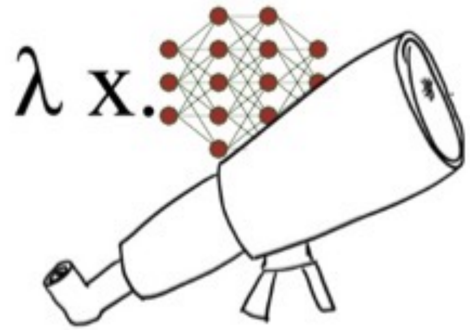


Neurosymbolic Programming

Swarat Chaudhuri **Jennifer J. Sun**

Armando Solar-Lezama

The University of Texas at Austin

Neurosymbolic Programming

Neurosymbolic Function Representations (Symbolic code + neural networks)

Neurosymbolic Learning Algorithms

Neurosymbolic Programming

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

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

Outline of Tutorial

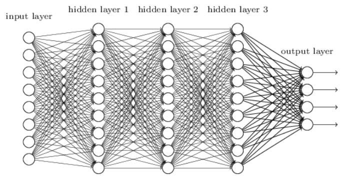
1. What is Neurosymbolic Programming?
2. Deep Dive: Neurosymbolic Programming for Science
3. Algorithmic Techniques
4. Deep Dive (continued)
5. Algorithmic Techniques (continued)
6. Conclusion

What is Neurosymbolic Programming?

Ingredients for Machine Learning

$$S = \{(x_i, y_i)\}_{i=1}^{N}$$

Training Data



Model Class(es)

This tutorial is mainly about these two

$$L(a,b) = (a-b)^2$$

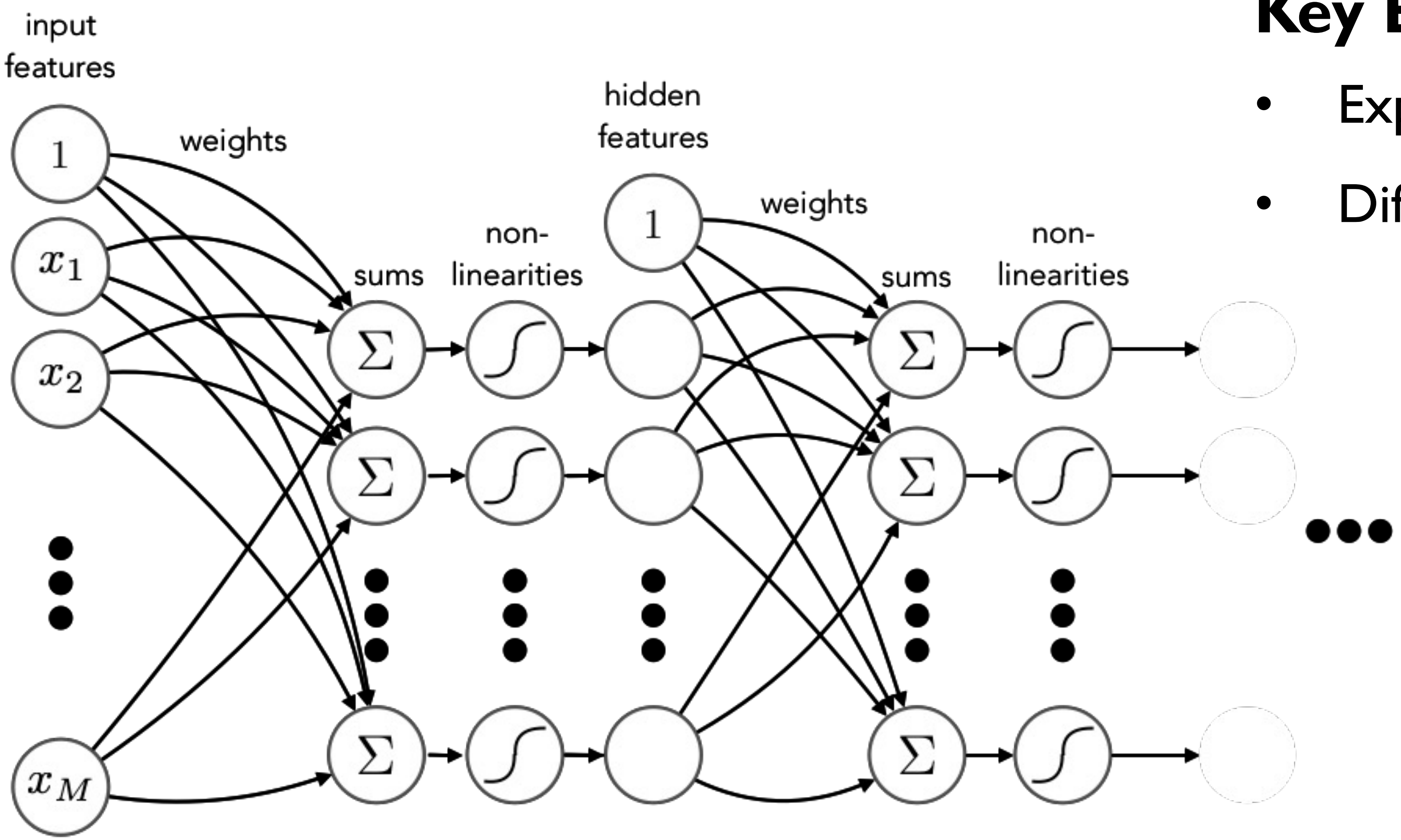
Loss Function

$$\underset{w,b}{\operatorname{argmin}} \sum_{i=1}^{N} L\big(y_i, f(x_i \mid w,b)\big)$$

Learning (Optimization)

Profit!

Deep Neural Networks

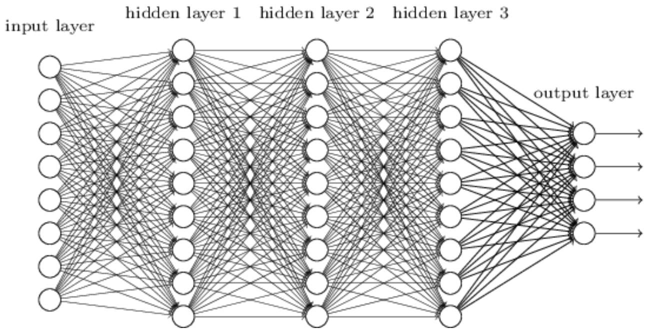


Key Benefits:

- 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

Key Insight: Use Symbolic Knowledge

A.k.a. “Code” or “Programs”

Two Ideas

A. Neurosymbolic function representations

B. Neurosymbolic learning algorithms

A. Neurosymbolic Function Representations

Neurosymbolic Programs

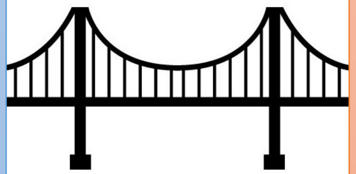
Symbolic Programs

Interpretable

Verifiable

Structured domain knowledge

Data efficient



Neural Networks

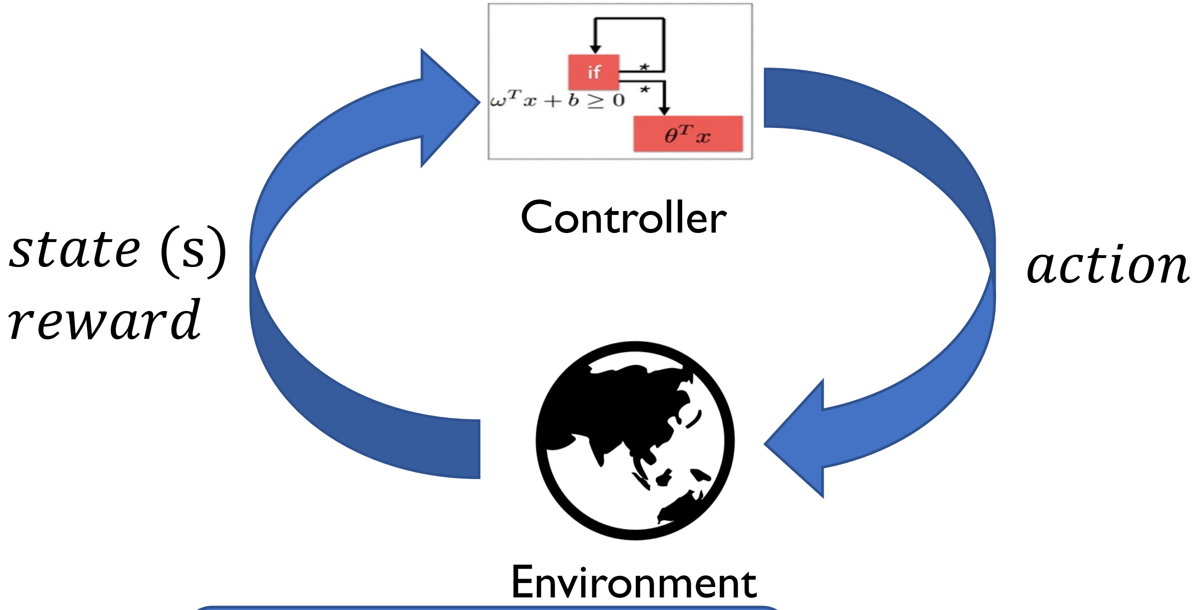
Scalable algorithms

Flexible

Handles messy data

Easy to get started

Application: Control



Goal: Control a car.

Symbolic code:

$$PID_{\langle i,s^*,k_P,k_I,k_D\rangle}(s) = k_P P(s - s^*) + k_I I(s - s^*) + k_D D(s - s^*)$$

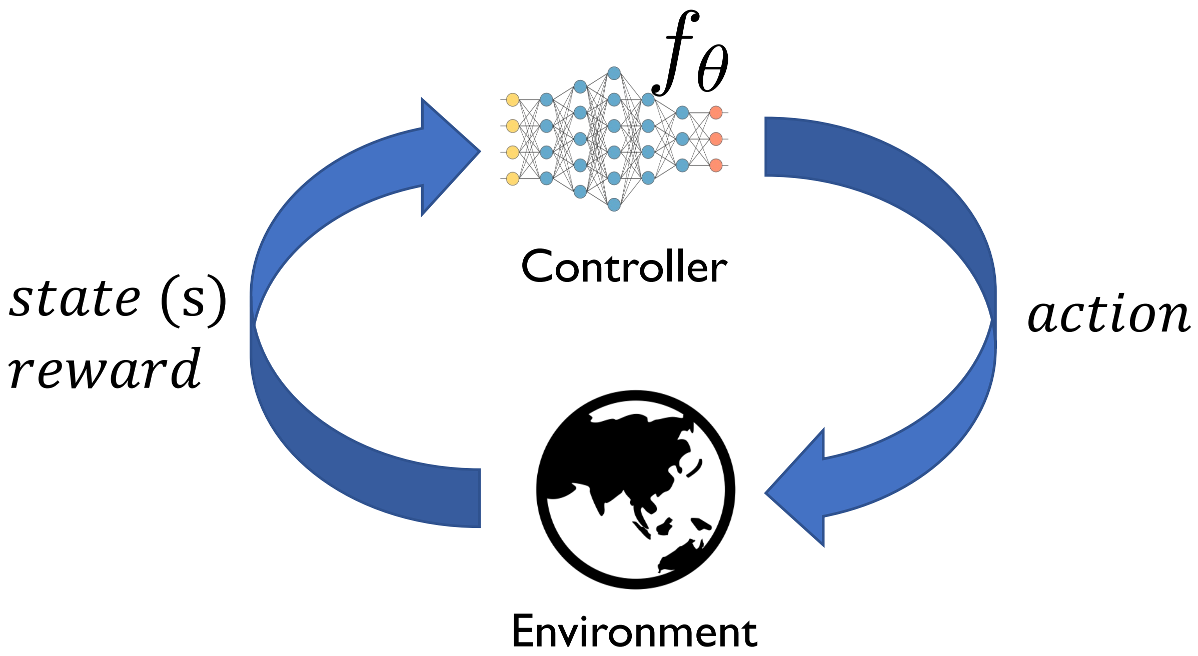
"If the car is aligned with the axis of the track…"

$$\begin{aligned} SwitchingPID(s) = \quad & \textbf{if } (s[\texttt{TrackPos}] < 0.011 \textbf{ and } s[\texttt{TrackPos}] > -0.011) \\ & \textbf{then } PID_{\langle \texttt{rpm}, 0.45, 3.54, 0.03, 53.39\rangle}(s) \\ & \textbf{else } PID_{\langle \texttt{rpm}, 0.39, 3.54, 0.03, 53.39\rangle}(s) \end{aligned}$$

"then accelerate, otherwise slow down"

[Verma et al., 2019]

Machine learning



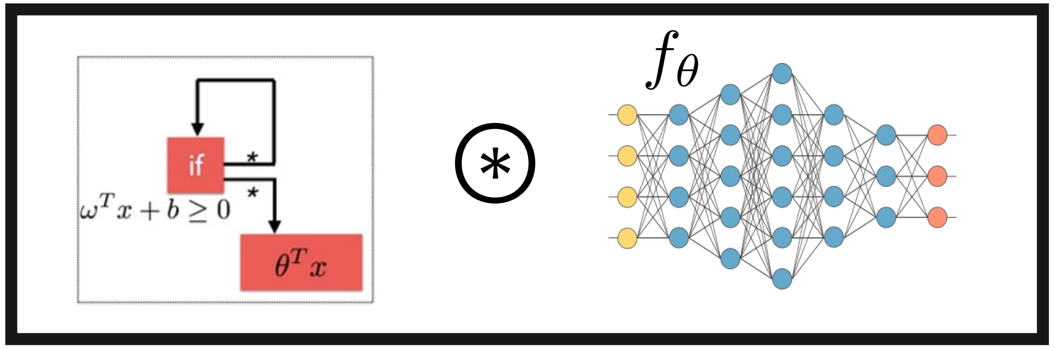
Learning a controller.

1. Collect data
    - For example, from exploration of the world or human demonstrations
2. Select a suitable model class (e.g., a category of neural networks)
3. Learn a function from the model class that maximizes reward.

Control with neurosymbolic programs

[Cheng et al., 2019]

Idea: Models are programs with neural and symbolic components



Neurosymbolic Controller

$$NS_\lambda(s) = (1 - \lambda)\ SwitchingPID(s) + \lambda\ f_\theta(s)$$

where $0 \leq \lambda \leq 1$

Symbolic controller used as a "regularizer"

Neurosymbolic vs. Neural [Verma et al., 2019]

Video by Abhinav Verma

Neural Control with Continuous-Time Symbolic Models [Shi et al., 2019]

NNs to model residual dynamics

$f(x)$ = Physics(x) + $f_a(x)$

Symbolic

- Dynamics:

$$\begin{cases} \dot{\mathbf{p}} = \mathbf{v}, & m\dot{\mathbf{v}} = m\mathbf{g} + R\mathbf{f}_u + \mathbf{f}_a \\ \dot{R} = RS(\boldsymbol{\omega}), & J\dot{\boldsymbol{\omega}} = J\boldsymbol{\omega} \times \boldsymbol{\omega} + \boldsymbol{\tau}_u + \boldsymbol{\tau}_a \end{cases}$$

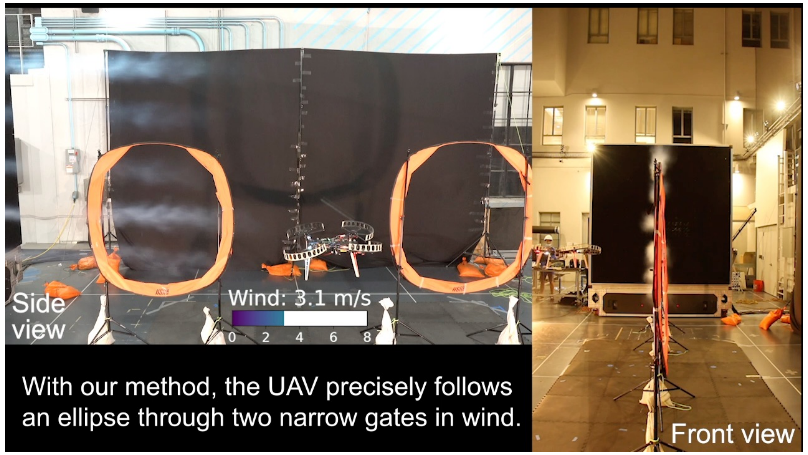
- Control:

$$\begin{cases} \mathbf{f}_u = [0, 0, T]^\top \\ \boldsymbol{\tau}_u = [\tau_x, \tau_y, \tau_z]^\top \\ \begin{bmatrix} T \\ \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} = \begin{bmatrix} c_T & c_T & c_T & c_T \\ 0 & c_T l_{\mathrm{arm}} & 0 & -c_T l_{\mathrm{arm}} \\ -c_T l_{\mathrm{arm}} & 0 & c_T l_{\mathrm{arm}} & 0 \\ -c_Q & c_Q & -c_Q & c_Q \end{bmatrix} \begin{bmatrix} n_1^2 \\ n_2^2 \\ n_3^2 \\ n_4^2 \end{bmatrix} \end{cases}$$

Symbolic

Concrete Instantiations

Boundary Conditions
https://arxiv.org/abs/1811.08027



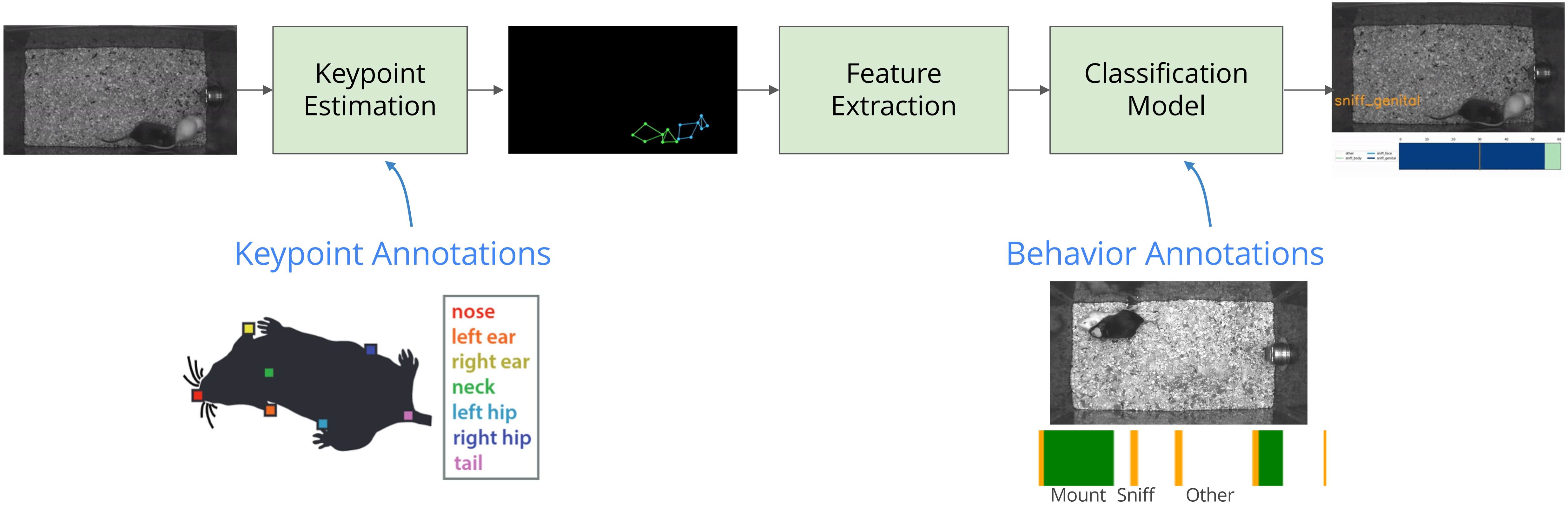
Dynamic Environments
https://arxiv.org/abs/2205.06908

Multi-agent Interactions
https://arxiv.org/abs/2003.02992

Neurosymbolic Programs in AI4Science

[Sun et al., 2022]

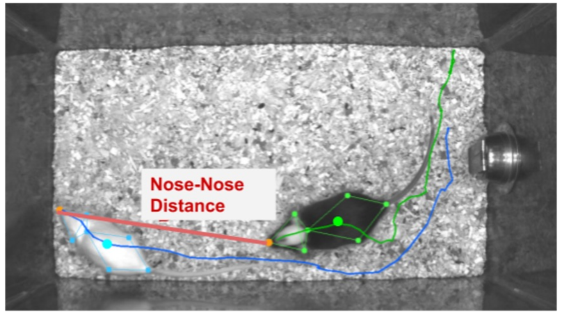
Goal: Quantify behavior from pose



Task Programming [Sun et al., 2021]

Step 1: Define important attributes (similar to feature design)

Identify important attributes



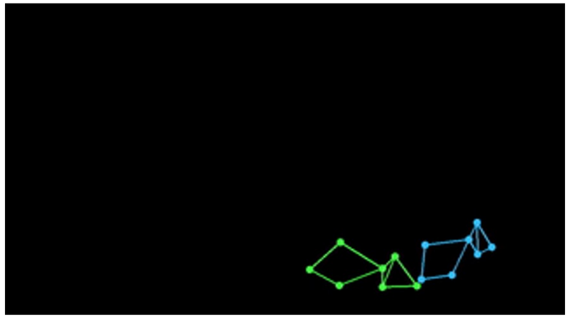
τ

Write programs

```
Nose coord mouse 1(x1, y1)
Nose coord mouse 2(x2, y2)

dist_nose(x1, y1, x2, y2):
    x_diff = x2 - x1
    y_diff = y2 - y1
    dist = norm(x_diff, y_diff)
```

Nose-Nose Distance

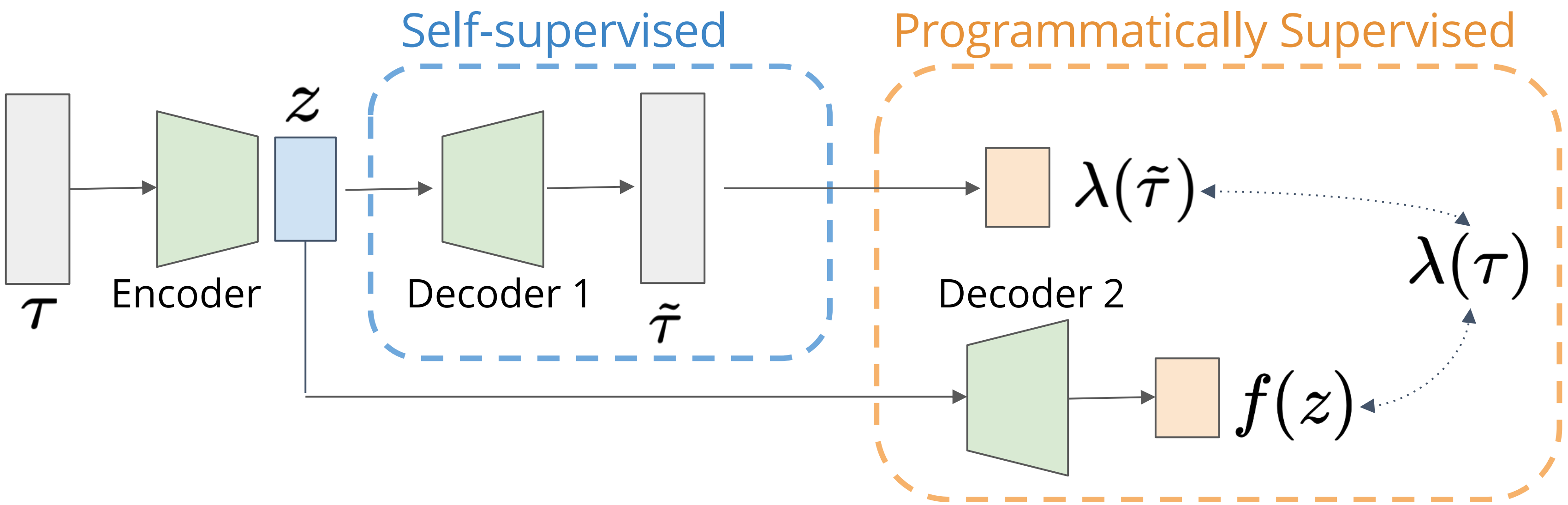
$\lambda(\tau)$



τ trajectory

Task Programming

Step 2: Use to structure representation learning



Self-supervised Loss

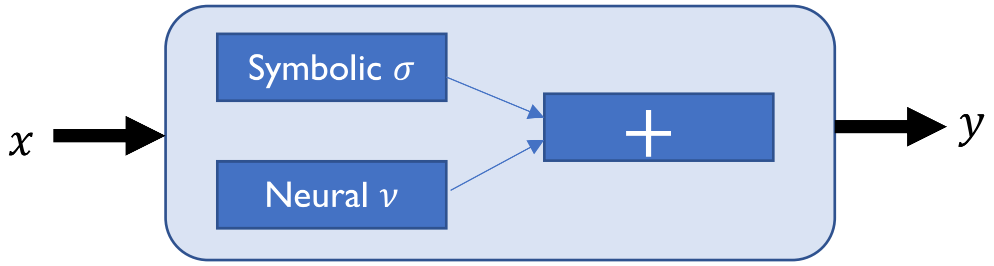
$$\mathcal{L}(\tau)$$

Programmatically Supervised Loss

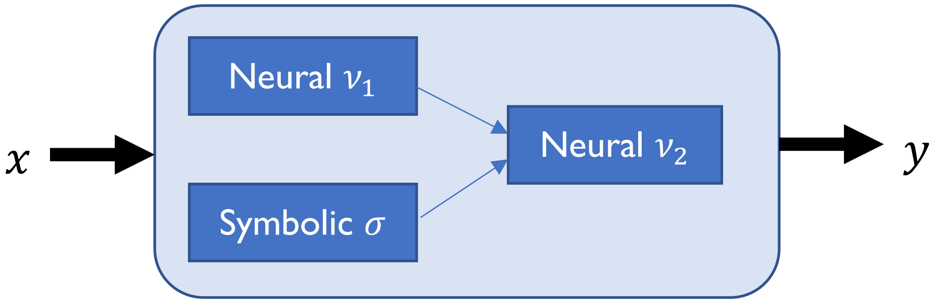
$$\mathcal{L}(\tau, \lambda(\tau))$$

Taxonomy of Neurosymbolic Programs

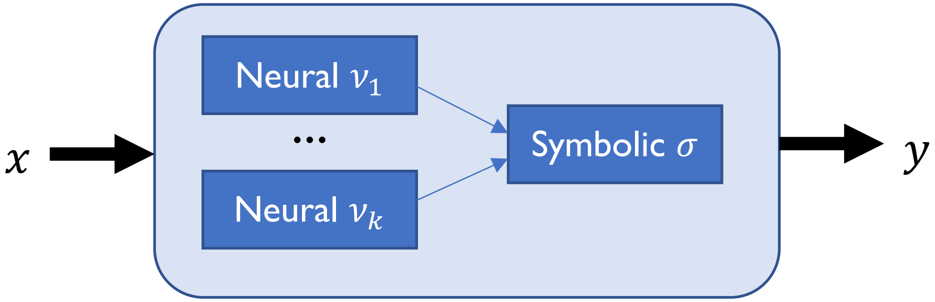
Additive composition [Cheng et al., 2019]



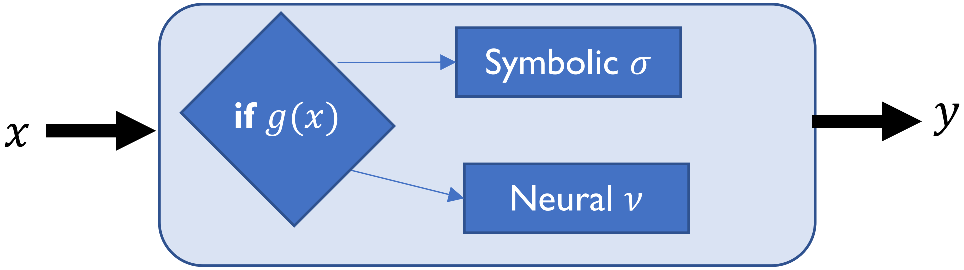
Neural-after-symbolic [Sun et al., 2021]



Symbolic-after-neural [Valkov et al., 2018]



Branching composition [Anderson et al., 2020]



B. Neurosymbolic Learning Algorithms

Learning as Symbolic Program Synthesis

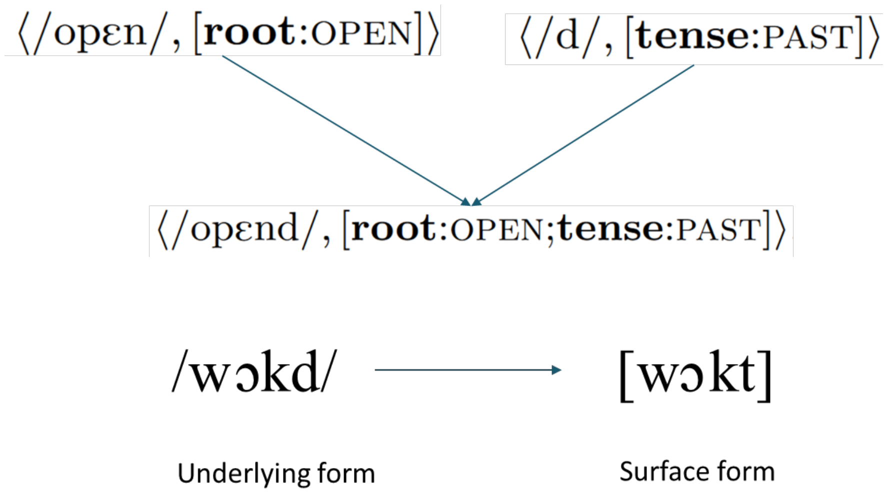
[Ellis et al., 2021]

Interpretable morpho-phonological rules for human languages from very few examples

Algorithm synthesizes rules using solvers for Boolean satisfiability (SAT)

Used to learn rules for 70 datasets spanning 58 languages

Understanding Morpho-phonology



Neurosymbolic Program Synthesis

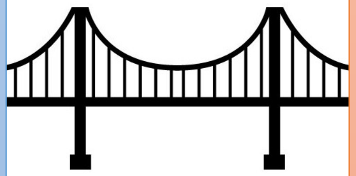
Program synthesis

Heuristic search

Solver-based search

Deductive pruning

Version spaces



Machine learning

Stochastic gradient descent

Sampling-based optimization

Variational approximations

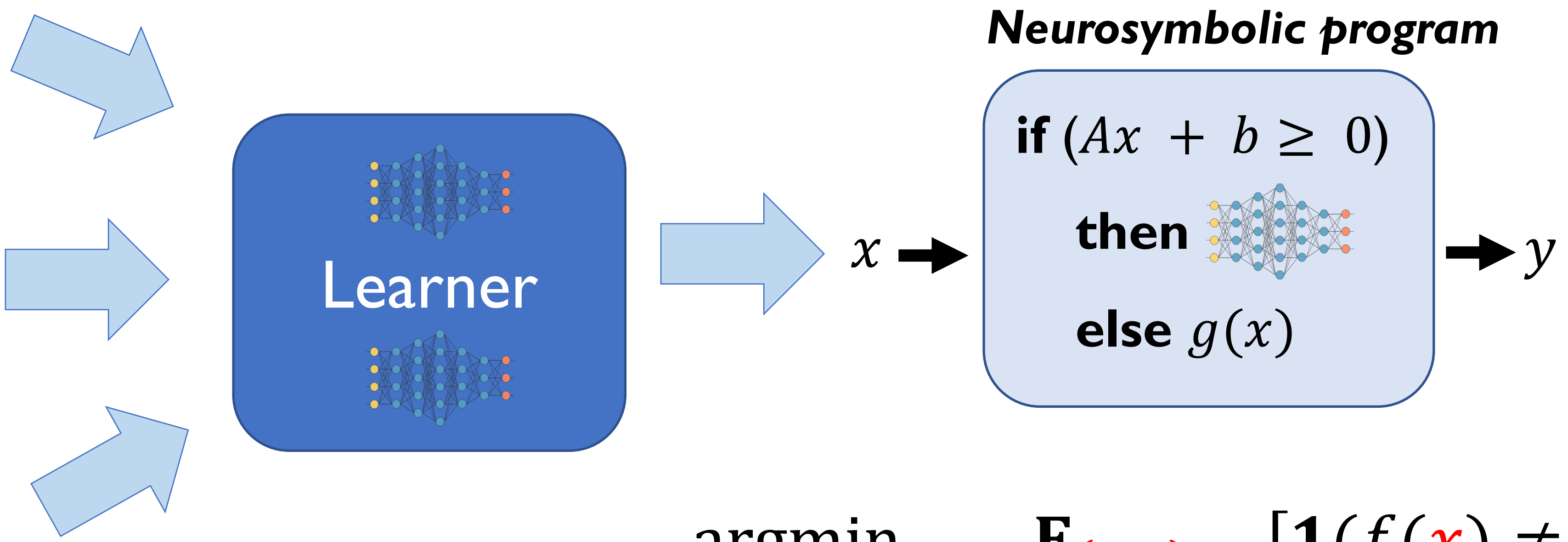
Learning to learn

Neurosymbolic Program Synthesis

(1) Data
$\langle x_i, y_i \rangle$

(2) Programming language (DSL L)

(3) Additional constraints (φ), e.g., safety, robustness



$$\mathrm{argmin}_{f \in L}\ \mathbf{E}_{(x,y)\sim D}[\mathbf{1}(f(x) \neq y)]$$
$$\text{s.t. } f \vDash \varphi$$

Domain-Specific Language ("Family of programs")

Program syntax defined as a grammar:

$$
\begin{aligned}
\alpha(s) \quad ::= \quad & a \\
& \mid Op(\alpha_1, \ldots, \alpha_k, s) \\
& \mid \textbf{if } b \textbf{ then } \alpha_1 \textbf{ else } \alpha_2 \\
& \mid \oplus_\theta(s)
\end{aligned}
$$

Action

Algebraic operations

Program structure

Current state

Predefined parametrized functions, capturing prior knowledge

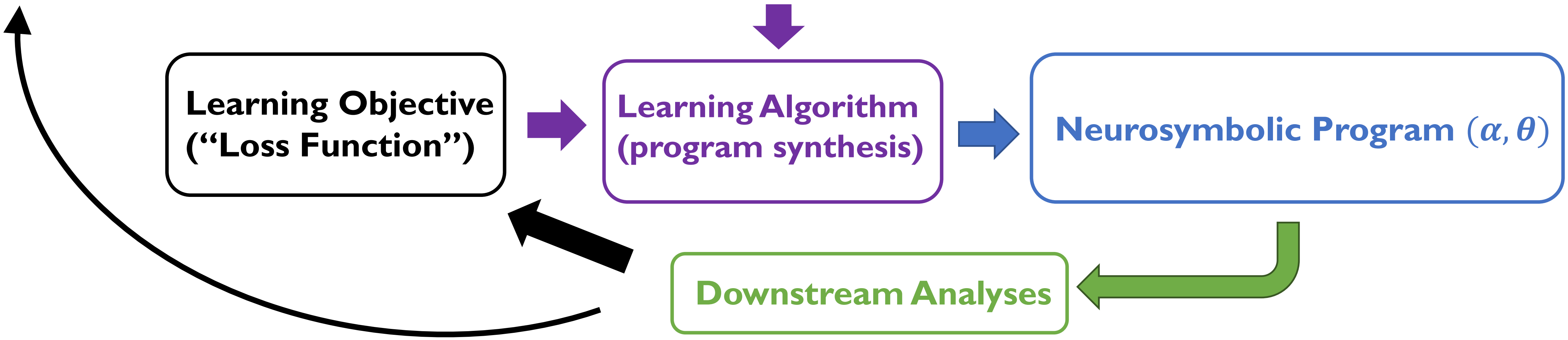
$$
\begin{aligned}
&\textbf{if } (s[\texttt{TrackPos}] < 0.011 \textbf{ and } s[\texttt{TrackPos}] > -0.011) \\
&\quad \textbf{then } PID_{\langle \texttt{rpm}, 0.45, 3.54, 0.03, 53.39 \rangle}(s) \\
&\quad \textbf{else } PID_{\langle \texttt{rpm}, 0.39, 3.54, 0.03, 53.39 \rangle}(s)
\end{aligned}
$$

Program semantics implemented using an interpreter

Neurosymbolic Program Synthesis

$$\begin{aligned}
\alpha(s) \quad &::= \quad a \mid Op(\alpha_1(s), \ldots, \alpha_k(s)) \mid \textbf{if } b \textbf{ then } \alpha_1(s) \textbf{ else } \alpha_2(s) \mid \\
&\qquad \oplus_\theta(\alpha_1(s), \ldots, \alpha_k(s)) \\
b \quad &::= \quad \phi(s) \mid BOp(b_1, \ldots, b_k)
\end{aligned}$$

Domain Specific Language (DSL): "Family of programs"



Observations

Traditional neurosymbolic learning

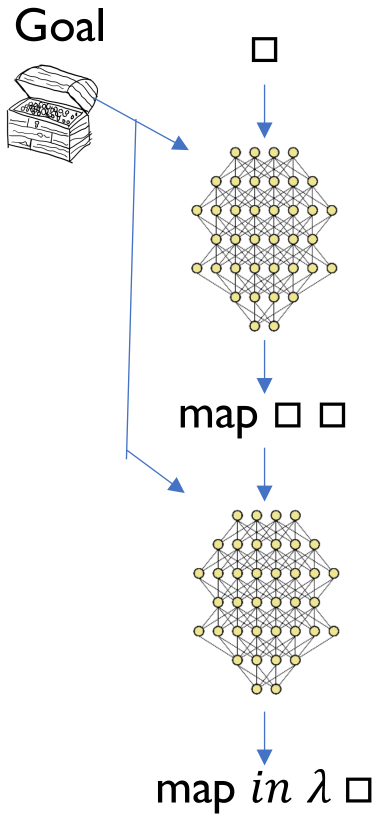
- Fixed program structure $\boldsymbol{\alpha}$ → train parameters $\boldsymbol{\theta}$ via gradient descent
- Setting $\boldsymbol{\alpha}$ as a neural network → standard deep learning
- Finding $\boldsymbol{\alpha}$ is analogous to neural architecture search
  - Sometimes call $\boldsymbol{\alpha}$ the "program architecture"
- Classic program synthesis focuses on $\boldsymbol{\alpha}$, with $\boldsymbol{\theta}$ being very simple

Example Program:

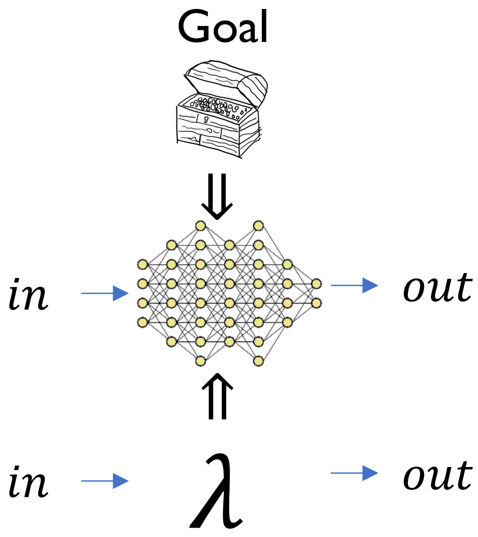
$$\begin{aligned}&\textbf{if } (s[\texttt{TrackPos}] < 0.011 \textbf{ and } s[\texttt{TrackPos}] > -0.011)\\ &\quad\textbf{then } PID_{\langle \texttt{rpm}, 0.45, 3.54, 0.03, 53.39\rangle}(s)\\ &\quad\textbf{else } PID_{\langle \texttt{rpm}, 0.39, 3.54, 0.03, 53.39\rangle}(s)\end{aligned}$$

Taxonomy of Neurosymbolic Program Synthesis

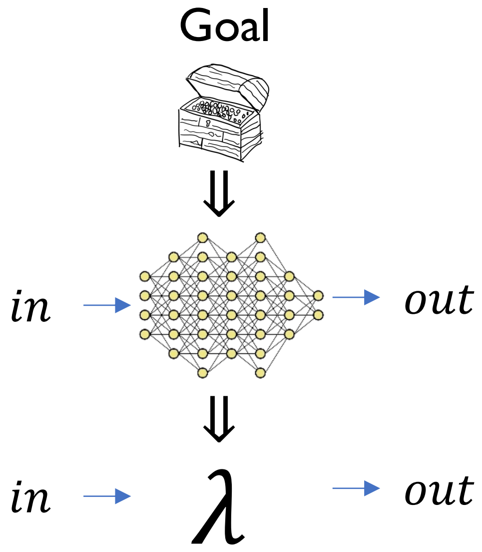
Neural-Guided Search



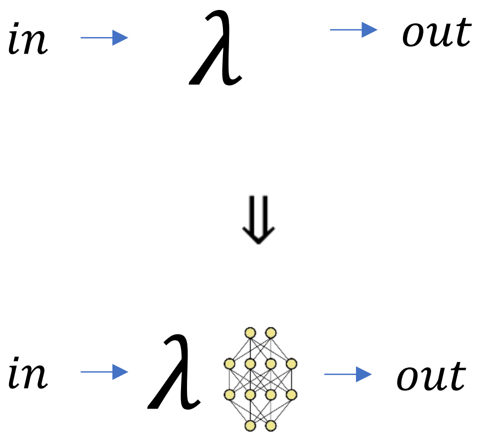
Symbolically Guided DL



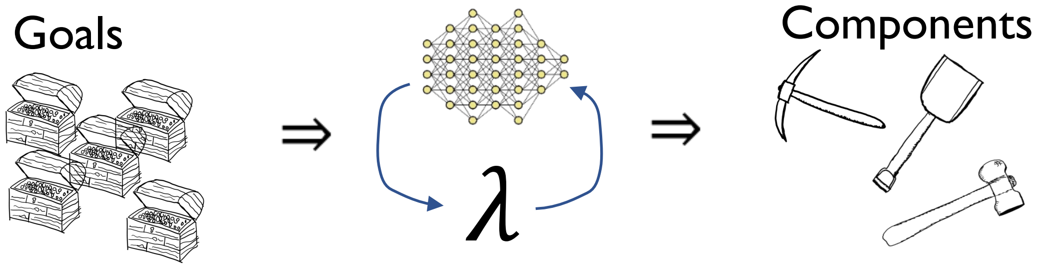
Distillation



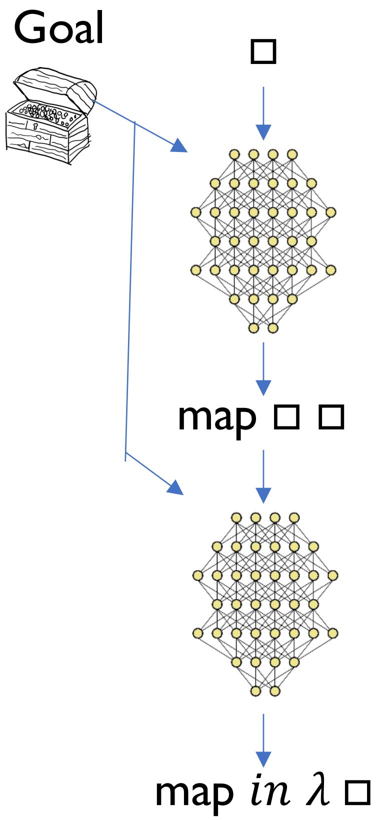
Relaxation



Component Discovery



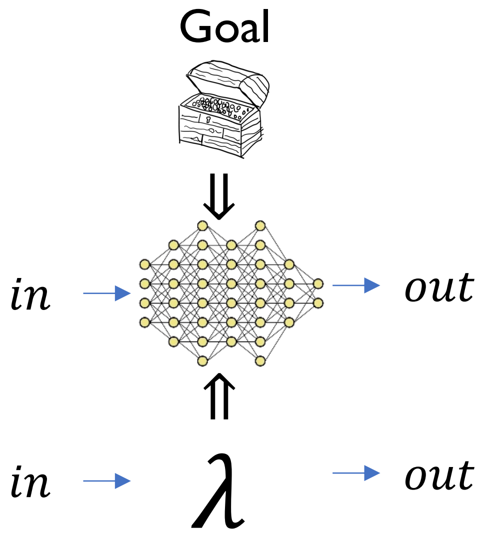
Neural-Guided Search [Devlin et al., 2017]



- Leverage the ability of NN to learn complex conditional distributions
- Network guides the search for programs that satisfy the goals

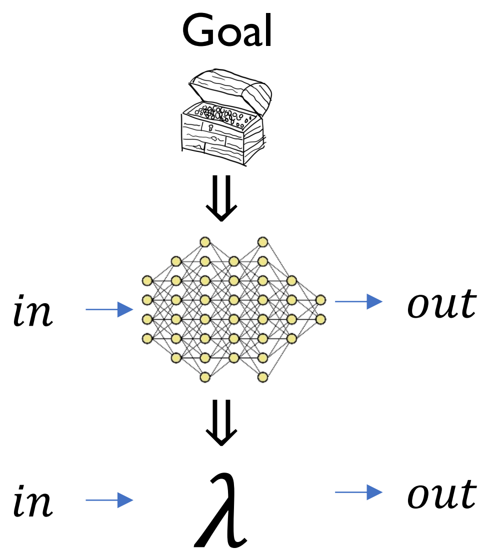
Symbolically Guided Deep Learning

[Sun et al., 2021]



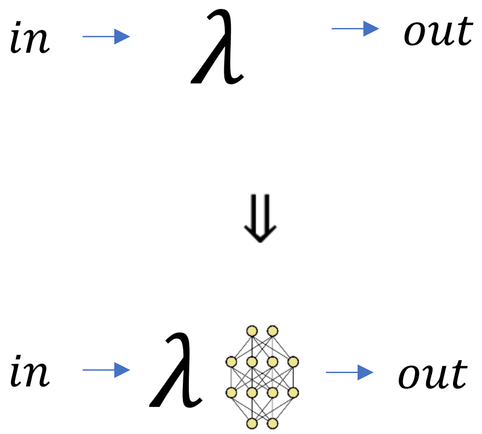
- Symbolic knowledge can be used to guide the training of neural networks
  - When you want a network that is consistent with prior knowledge
  - When you want to improve data efficiency and better generalization

Distillation [Verma et al., 2019]



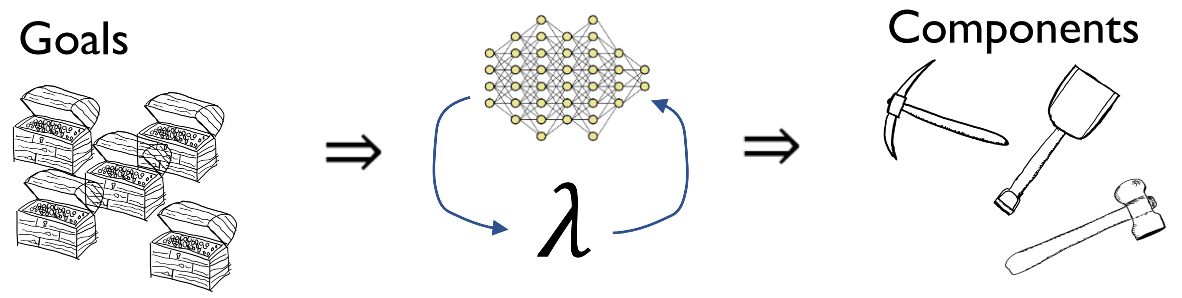
- Use the neural network as a starting point for program synthesis
- Replace neural components with symbolic ones
  - To improve interpretability and analyzability
  - To better generalize out of distribution
  - To ensure more predictable behavior

Relaxation [Shah et al., 2020]



- Replace symbolic components with neural proxies
  - Can help leverage the information in the symbolic component into a larger DL pipeline
  - Can help guide the search for symbolic components

Component Discovery [Ellis et al., 2020]



- Given a set of goals, we want to learn collections of components that can help us achieve those goals.
  - A form of abstraction where we want to identify the common structure in a set of goals and capture it symbolically into a set of useful components
  - Requires deep interaction between neural and symbolic reasoning

Neurosymbolic learning isn't new…

…but it's a good time to push on it!

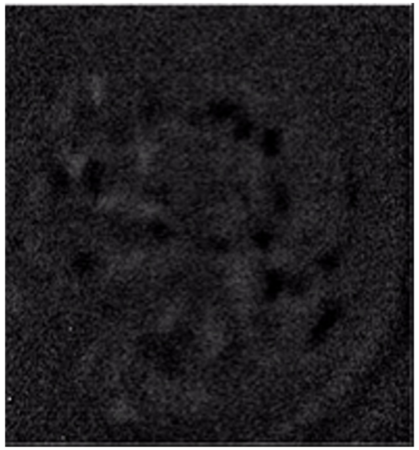
- Recent progress in symbolic reasoning as well as deep learning
- New algorithms that can scale
- Demand by domain experts

Deep Dive:

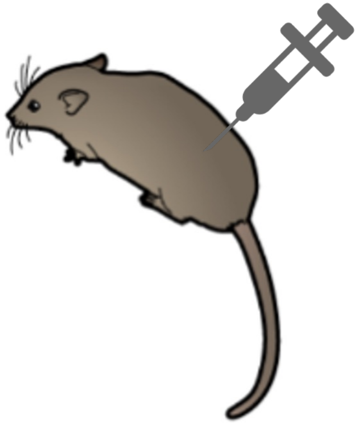
Neurosymbolic Programming for Science

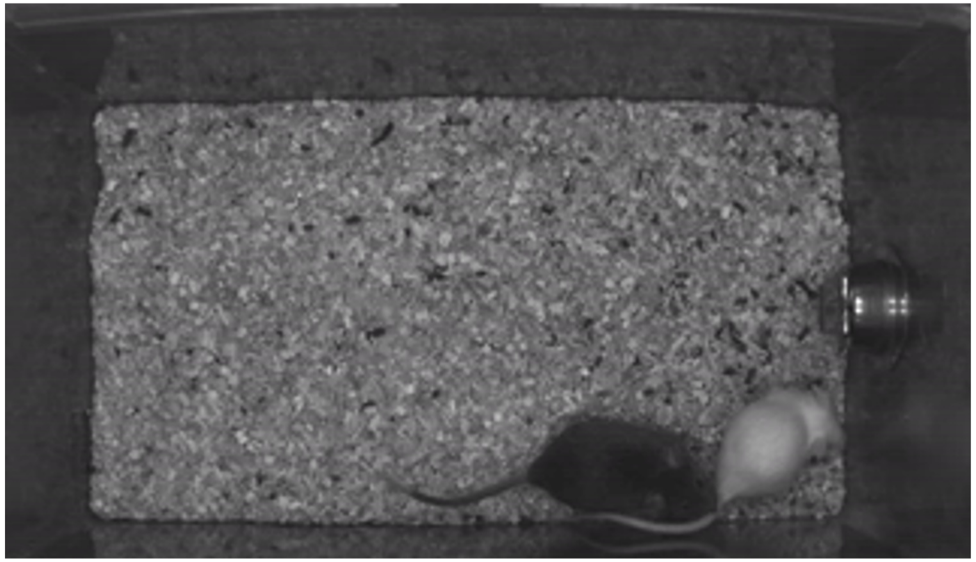
Behavior Analysis in Science

Neural Activity



Pharmacological Evaluation



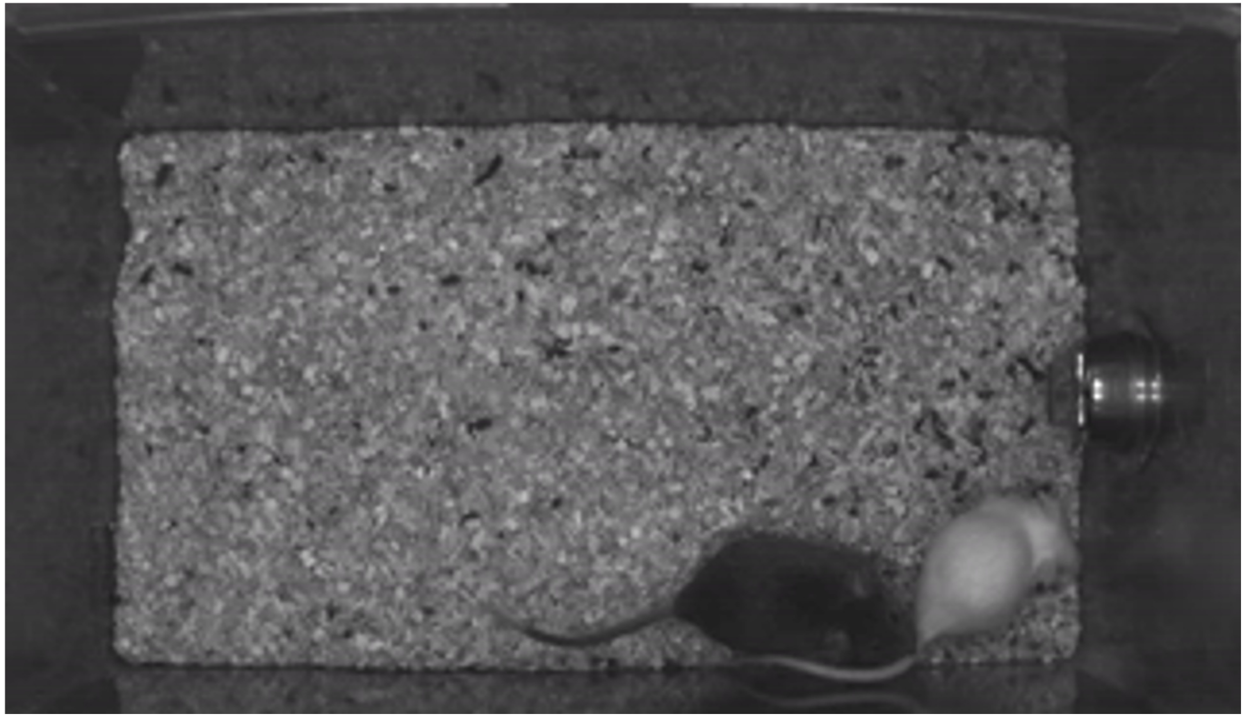


Observed Behavior

Strain variations

Behavior Quantification

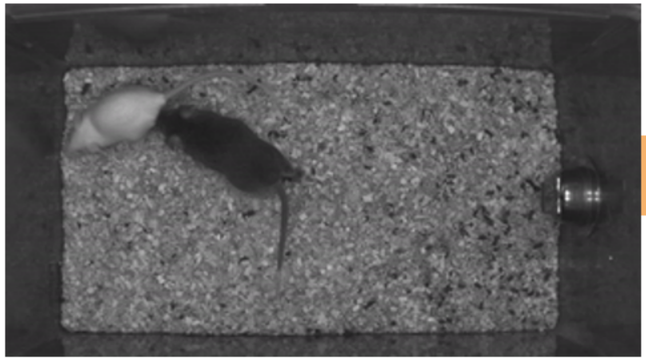
How to categorize behavior at each frame?



>100 annotation hours per day of recording

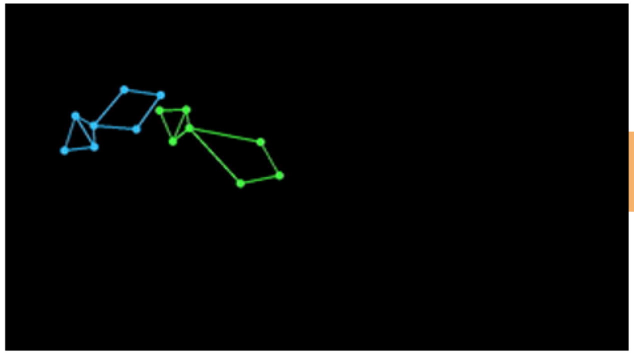
CalMS21 Dataset: https://arxiv.org/pdf/2104.02710.pdf

Dataset Overview



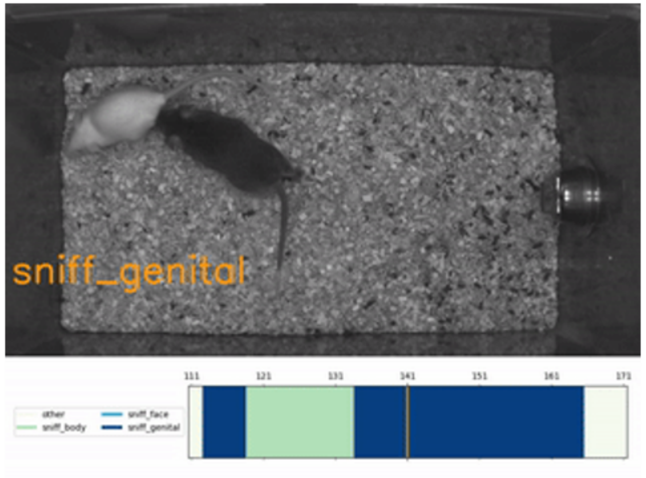
Observed Behavior

Keypoint Estimator



Keypoints

Behavior Classifier



Behavior Labels (from experts)

The Caltech Mouse Social Interactions 2021 Dataset

CalMS21

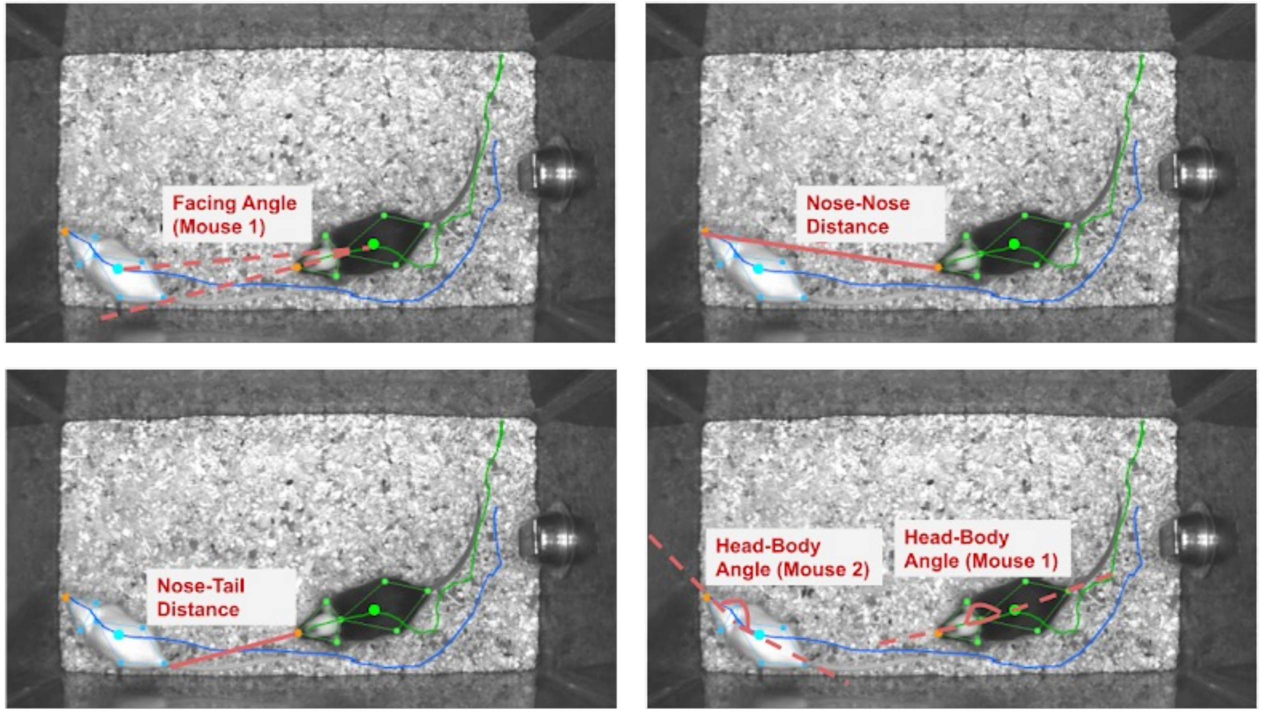
Code: Use neurosymbolic programming to learn relationship between pose and behavior

CalMS21 Dataset: https://arxiv.org/pdf/2104.02710.pdf

Behavioral Attributes / Features

Dataset:

- Raw trajectories
- Expert-annotated behaviors
- Behavioral attributes

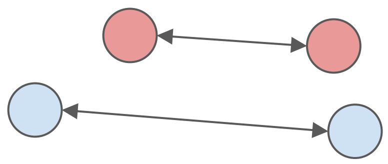


Designed by domain experts for behavior analysis

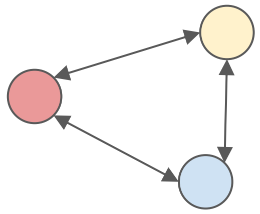
Task Programming: https://arxiv.org/pdf/2011.13917.pdf

Defining the space of programs

Feature Selections

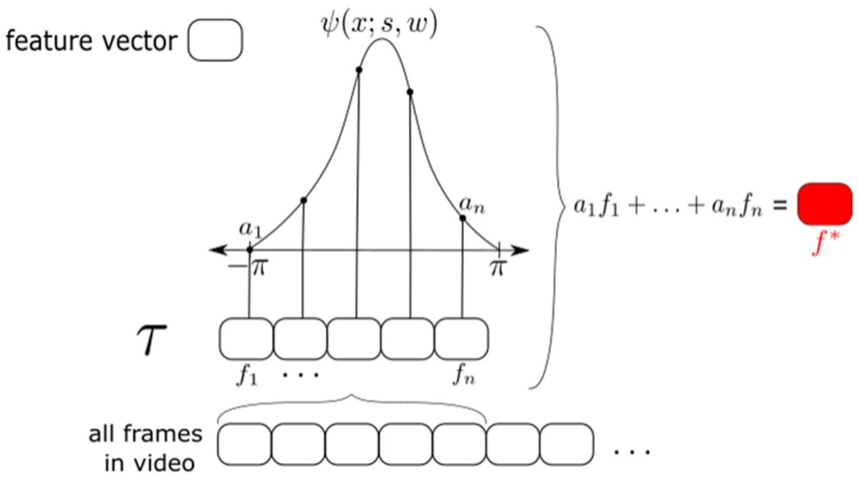


Speed



Distance

Temporal Filters



Compositions

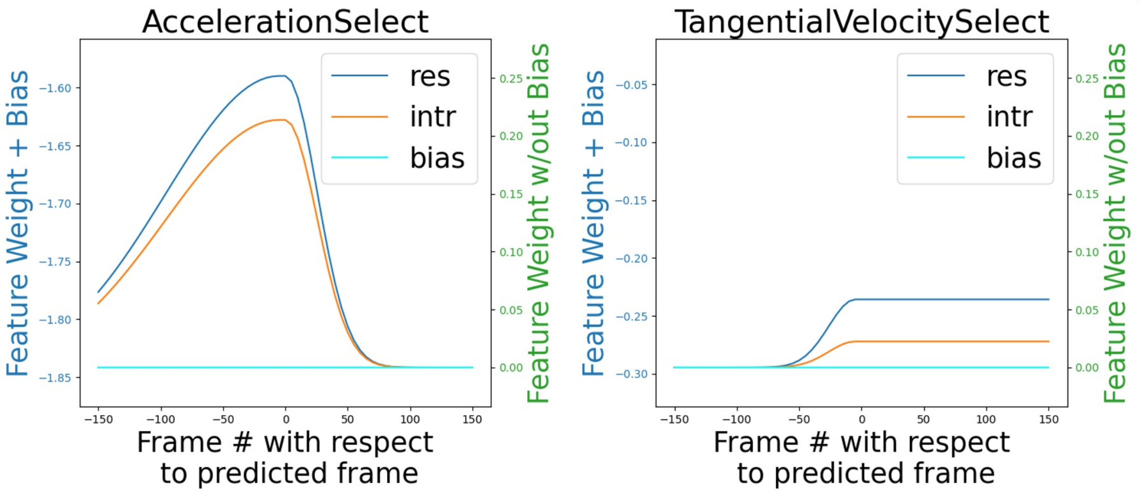
Subprogram 1

+

Subprogram 2

Program Examples

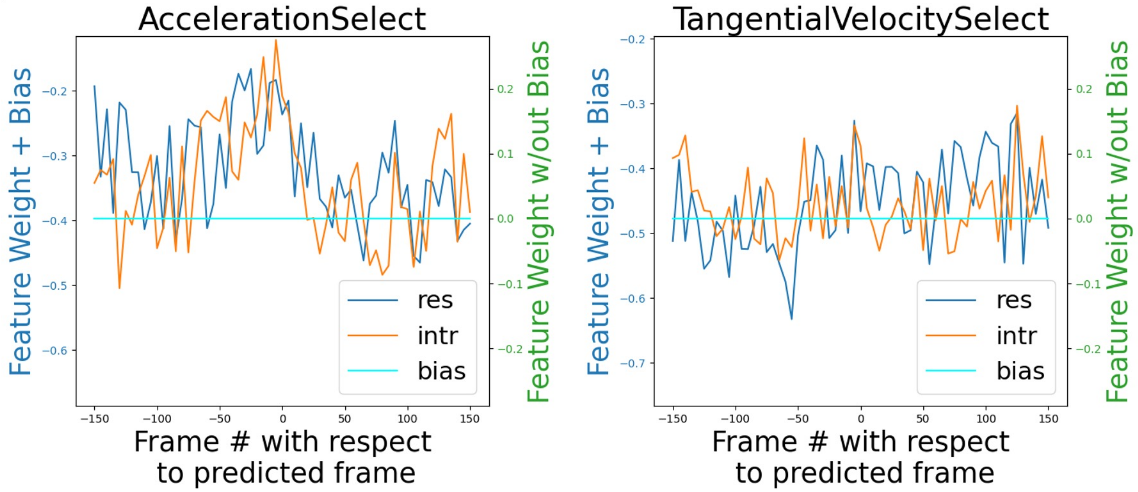
Program Learning



F1: 0.86

1D Conv Net

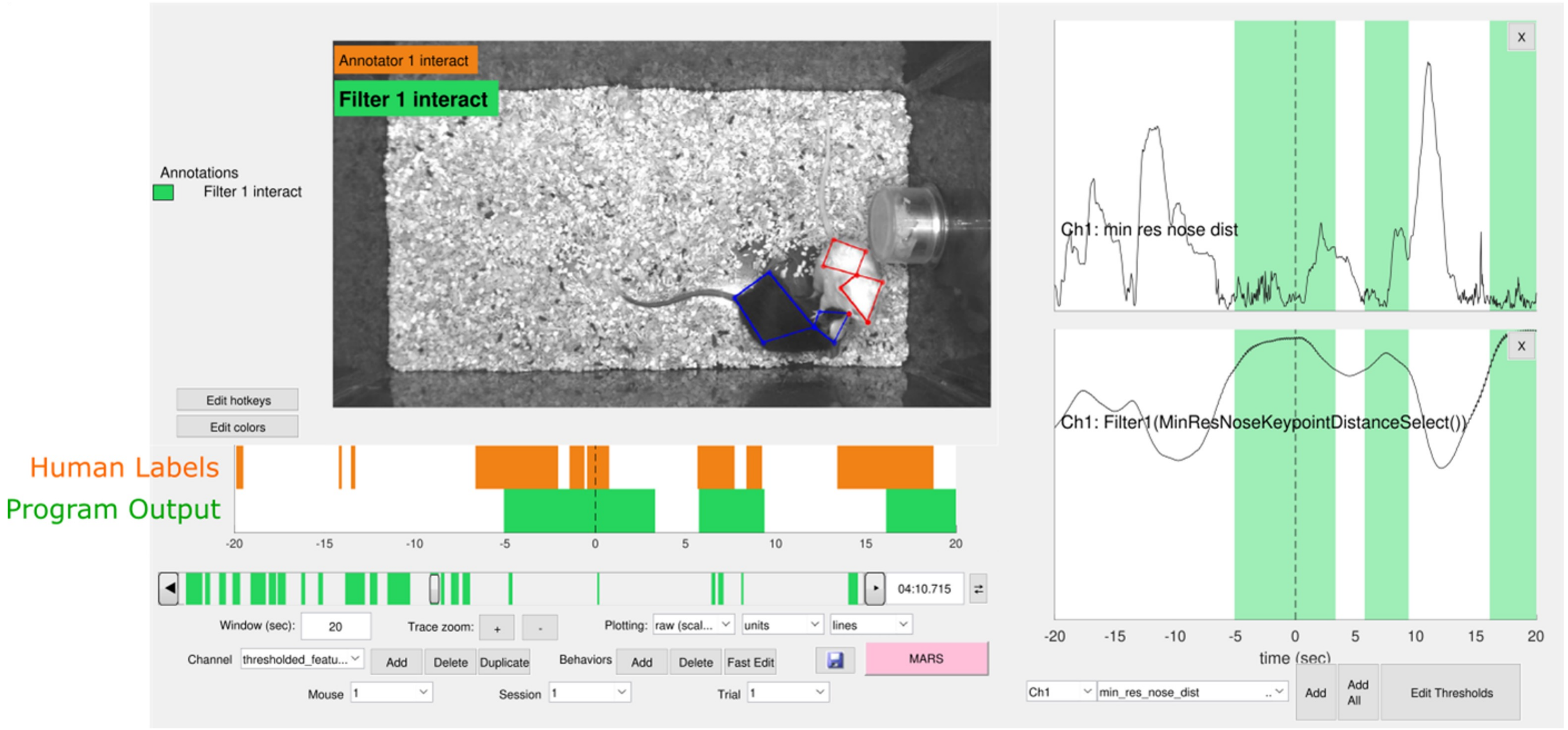
(Visualizing feature subset)



F1: 0.84

Interpreting Annotation Differences: https://arxiv.org/pdf/2106.06114.pdf

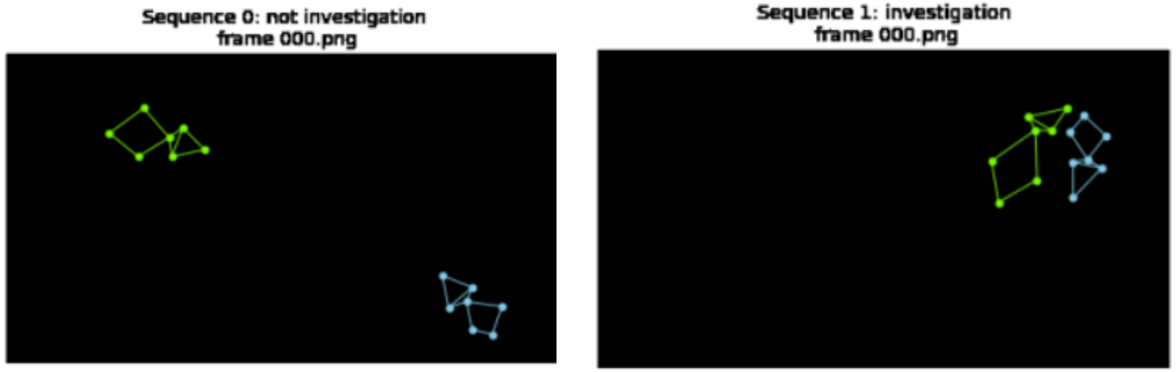
Program Visualizations



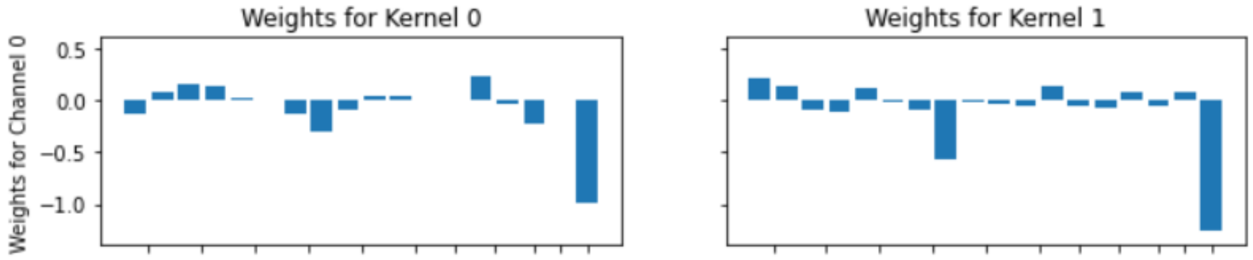
MARS (Segalin et al.)
neuroethology.github.com/MARS

Code Structure

- Data Visualization
  - Plot trajectory samples
- Neural Network
  - Train a 1D Conv Net
- Program Learning
  - Train program given structure
- Visualize Model Weights
- Open-Ended Exploration




```
Window5Avg( Or(AccelerationSelect, OverlapBboxesSelect) )
```



Code Walk-Through

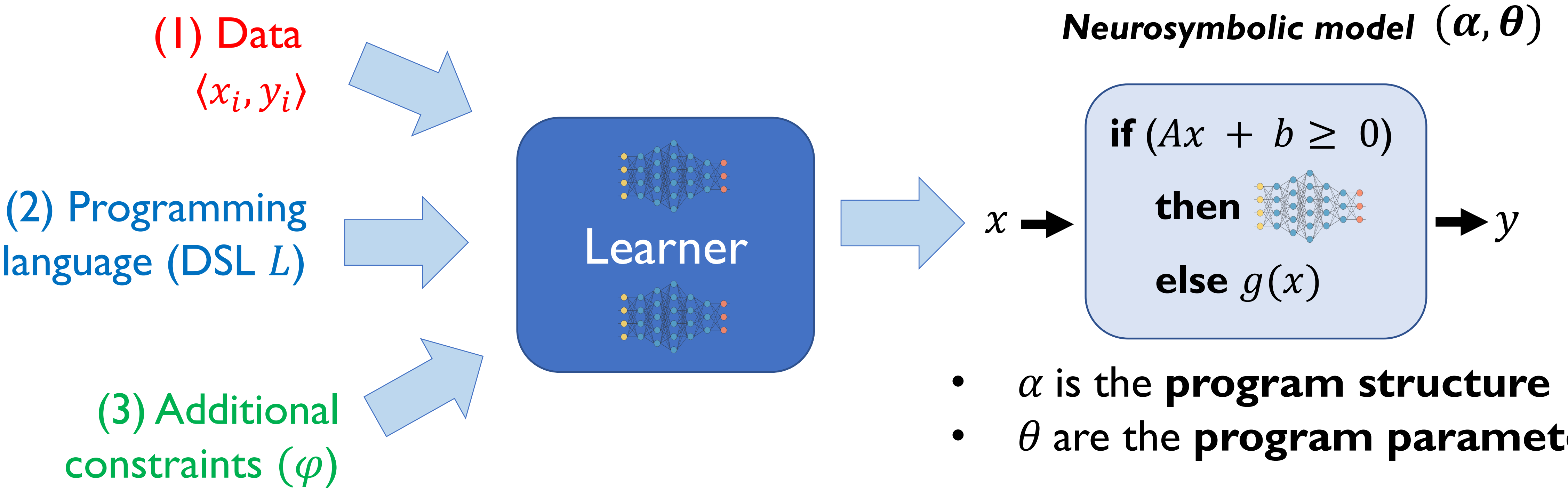
bit.ly/neurosym_tutorial

Outline of Tutorial

1. What is Neurosymbolic Programming?
2. Deep Dive: Neurosymbolic Programming for Science
3. Algorithmic Techniques
4. Deep Dive (continued)
5. Algorithmic Techniques (continued)
6. Conclusion

Algorithmic Techniques

Neurosymbolic Programming



- α is the **program structure**
- θ are the **program parameters**

Neurosymbolic models + neurosymbolic learning algorithms

Learning as Bilevel Optimization

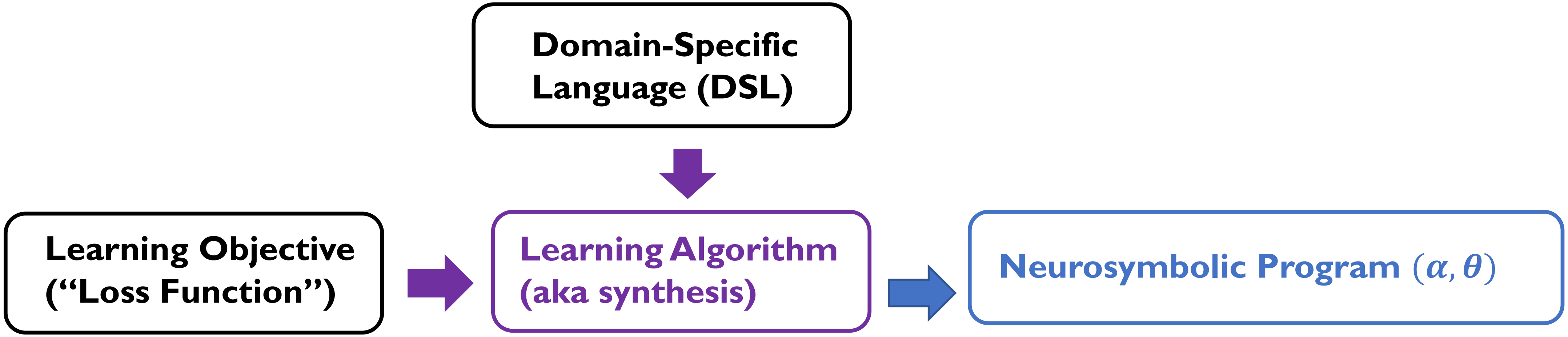
structure parameters

$$\min_{\alpha} \left(\min_{\theta} Loss(\alpha, \theta) + s(\alpha) \right)$$

- $Loss(\alpha, \theta)$ quantifies fit to the dataset
- The **structural cost** $s(\alpha)$ penalizes complex program structures.

$$\min_{\alpha} \left(\min_{\theta} Loss(\alpha, \theta) + s(\alpha)\right)$$

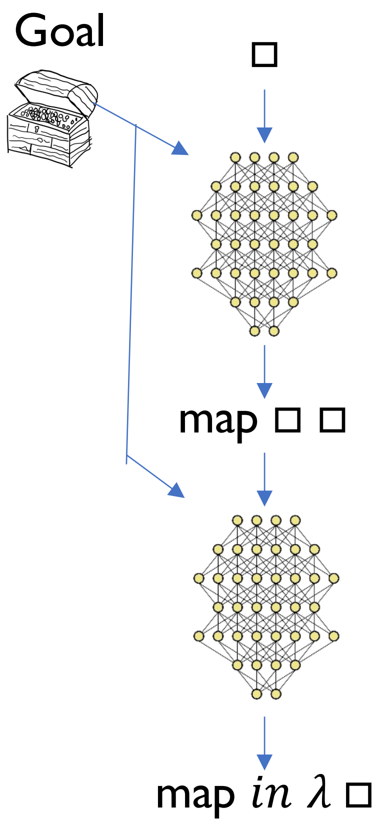
Learning Strategy



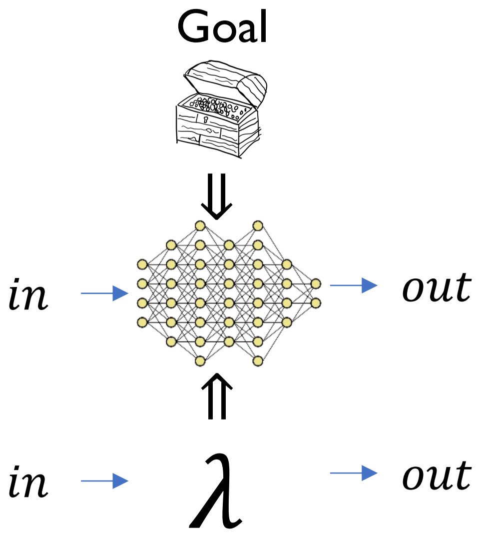
- Setting α as a neural network → standard deep learning
- Finding α is analogous to neural architecture search
  - Sometimes call α the "program architecture"
- Classic program synthesis focuses on α, with θ being very simple

Taxonomy of Neurosymbolic Program Synthesis

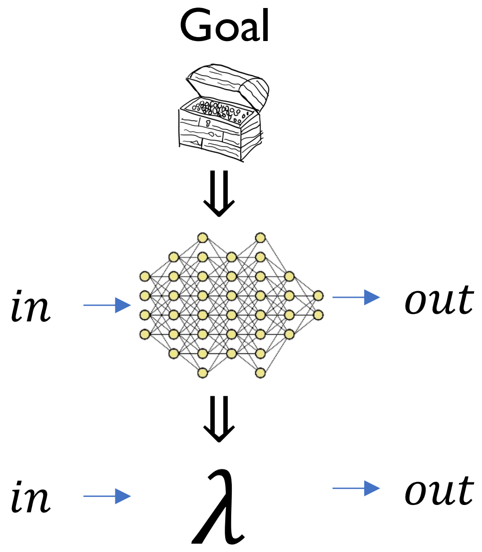
Neural-Guided Search



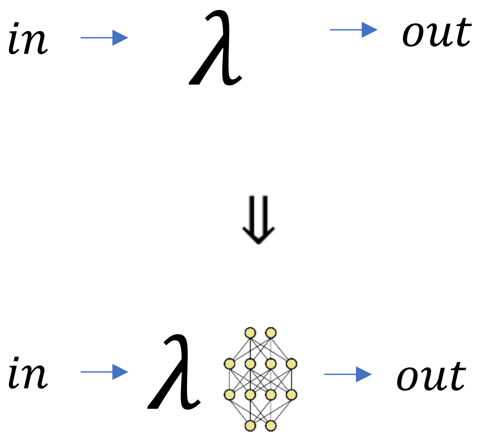
Symbolically Guided DL



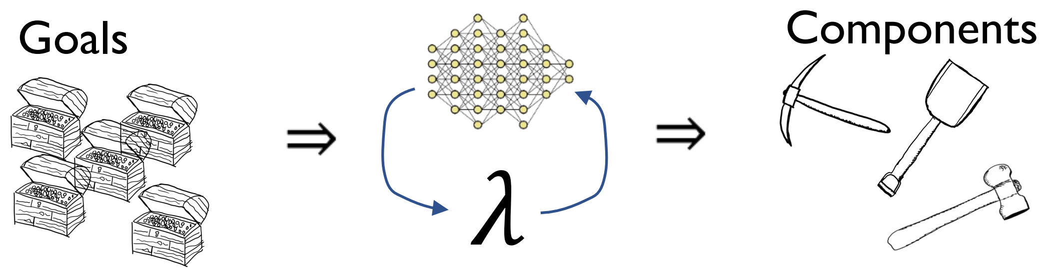
Distillation



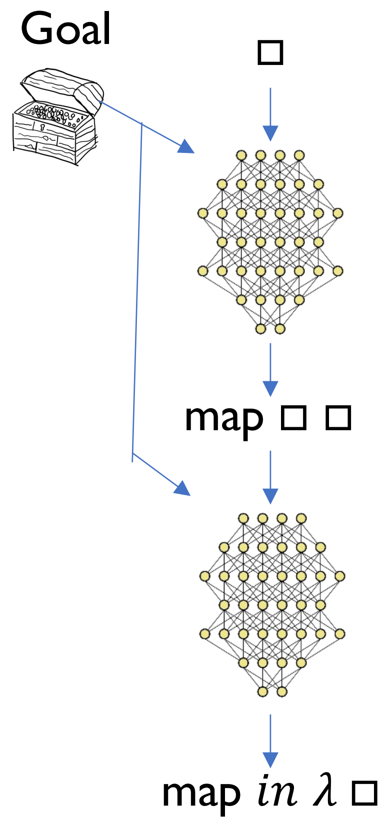
Relaxation



Component Discovery

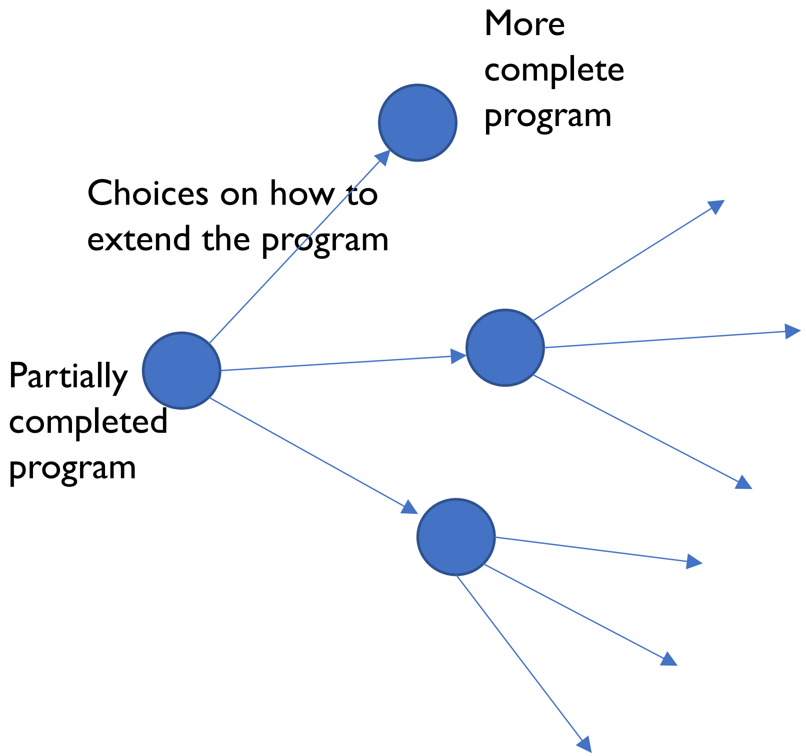


Neural-Guided Search



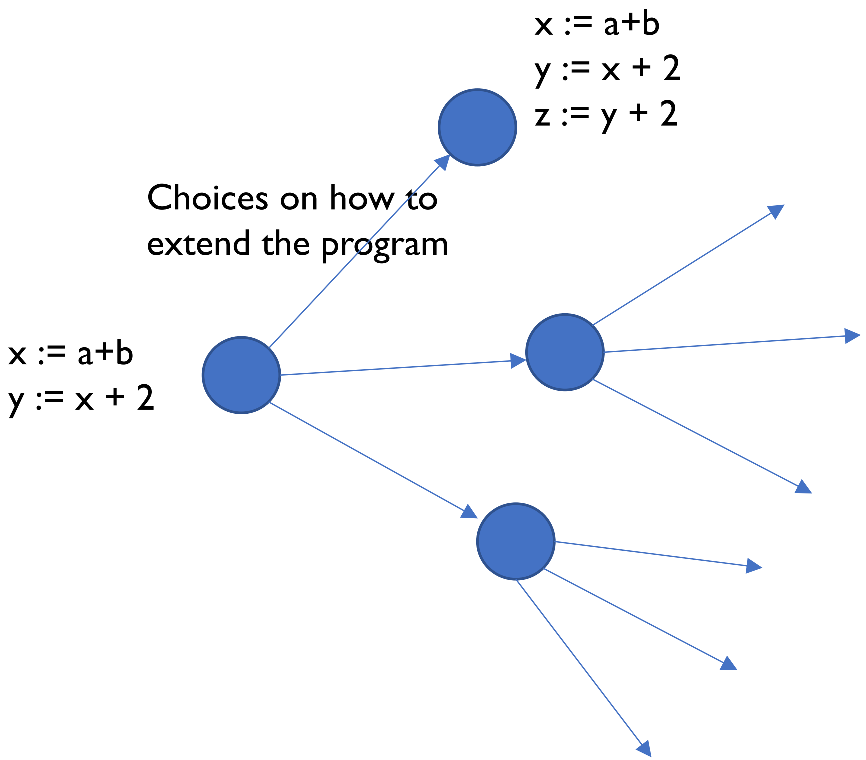
Enumerating programs

- Program enumeration is really a graph search problem



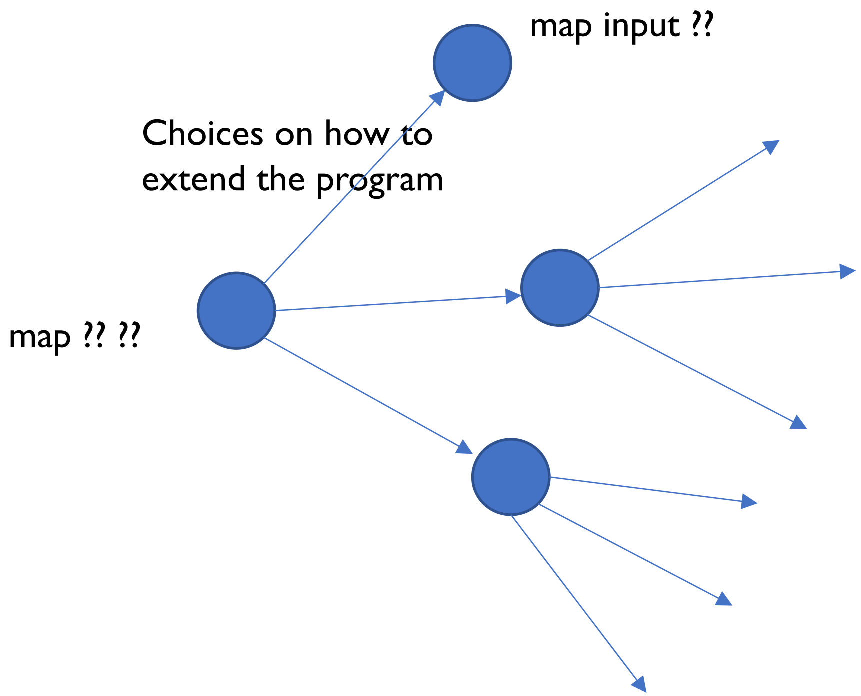
Enumerating programs

- Program enumeration is really a graph search problem



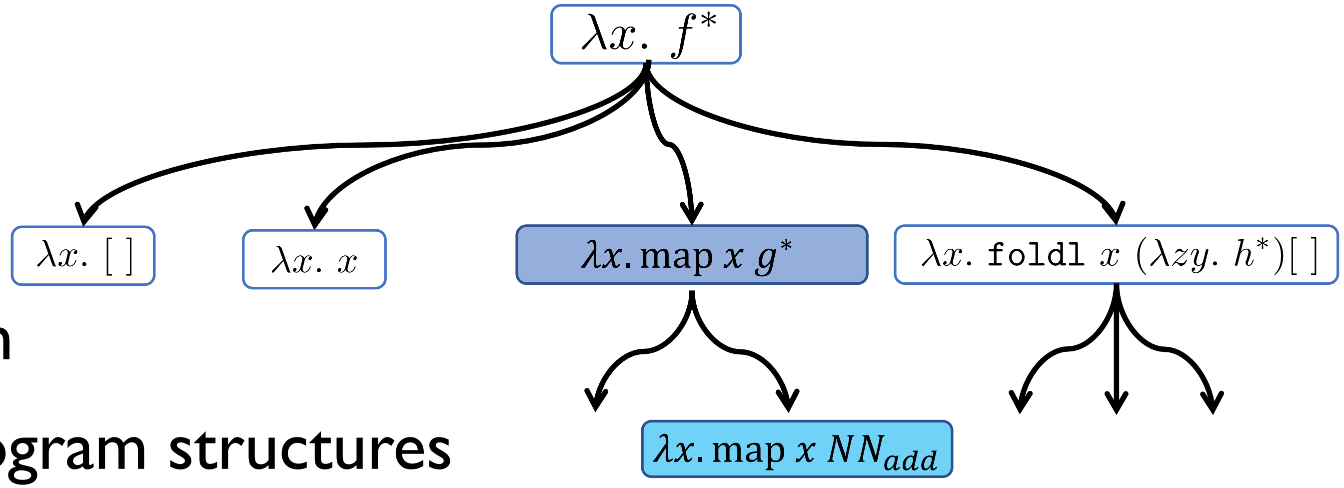
Enumerating programs

- Program enumeration is really a graph search problem



Algorithmic Idea: Type-Directed Enumeration

Top-Down Program Synthesis



Build up a search graph:

- The **root** is the empty program
- **Internal nodes** are partial program structures
- **Sinks** α are complete program structures
  - Come with costs $C(\alpha) = \min_{\theta} Loss(\alpha, \theta) + s(\alpha)$
- **Edges** model single derivatio

Goal: Find path from the root to a least cost sink.

Challenge: Too many programs!

Type-Directed Search

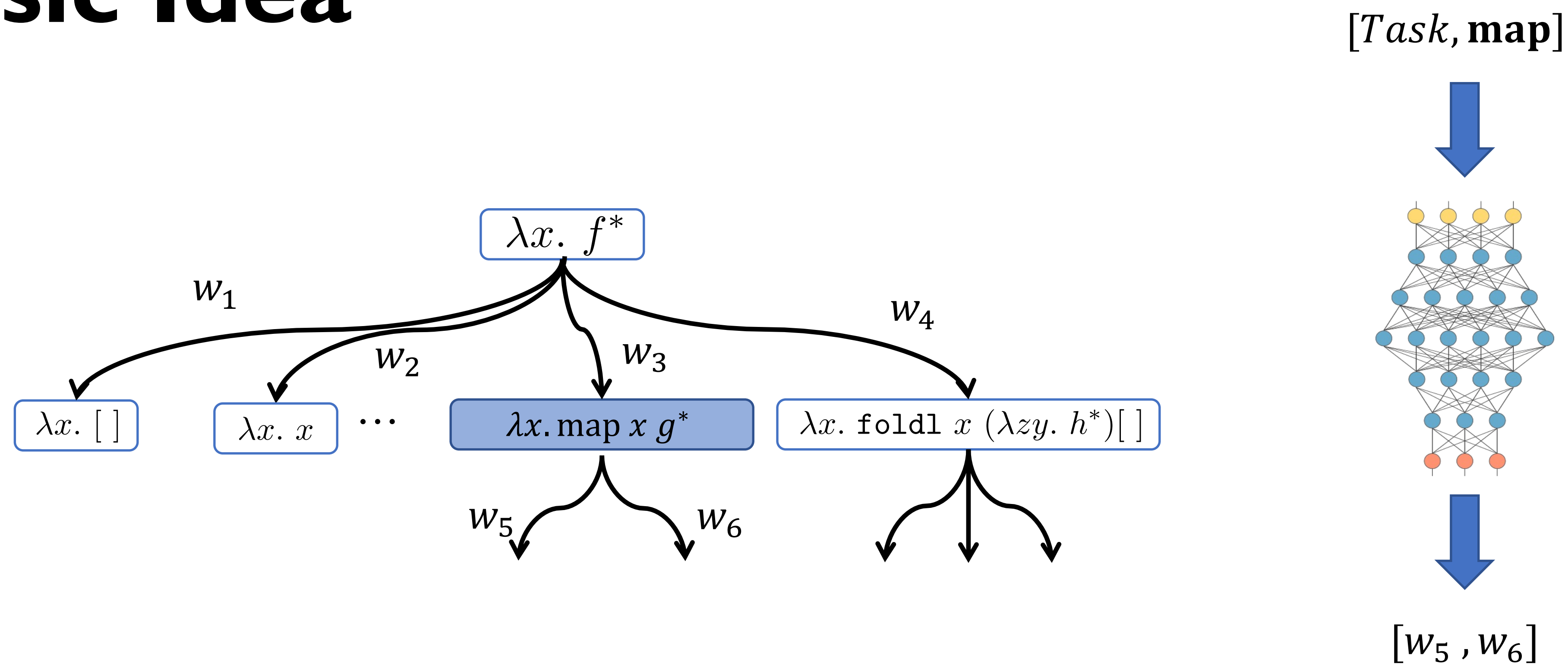
- **Pro:** Can lead to useful pruning
- **Con:** Doesn't engage with the *quantitative* aspect of the problem.

	Task	Number of programs		
		size = 4	size = 5	size = 6
No types	Task 1	8182	110372	1318972
	Task 2	12333	179049	2278113
	Task 3	17834	278318	3727358
	Task 4	24182	422619	6474938
+ Types	Task 1	2	20	44
	Task 2	5	37	67
	Task 3	9	47	158
	Task 4	9	51	175

Source: Valkov et al., 2018

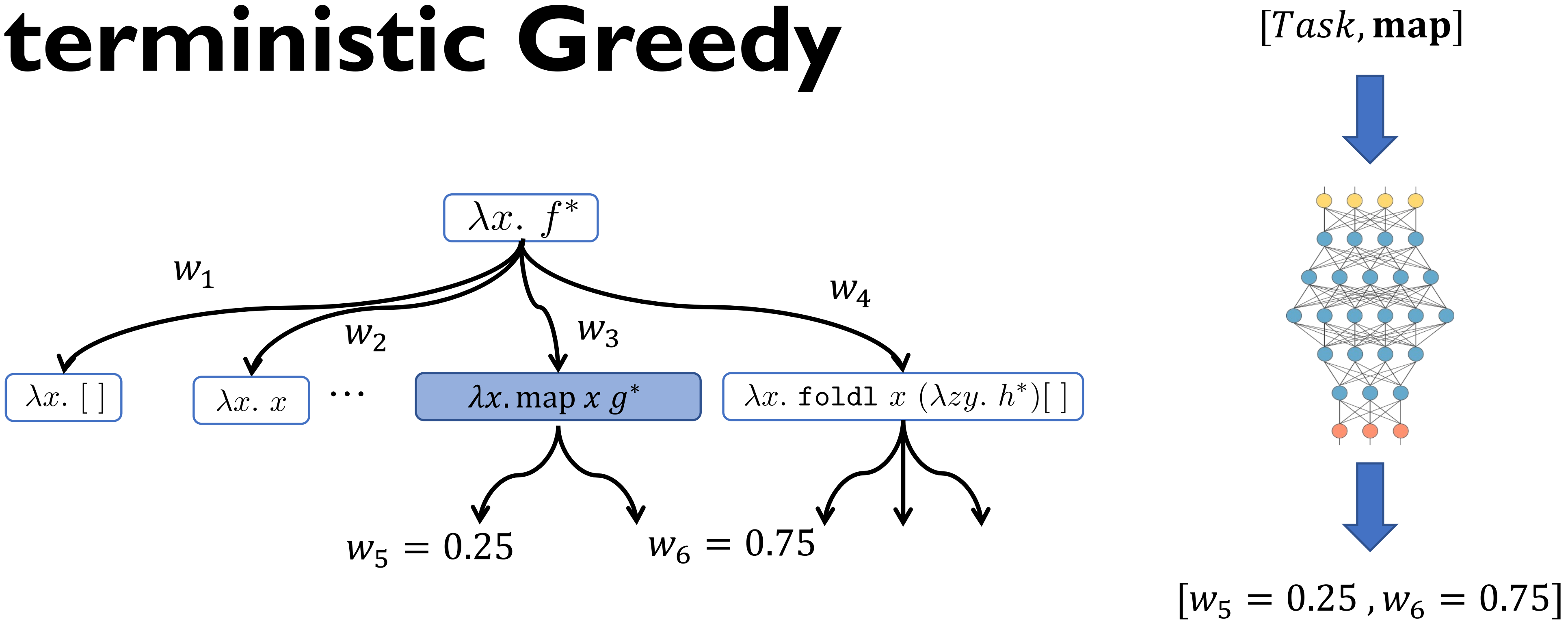
Learning to Search

Basic Idea



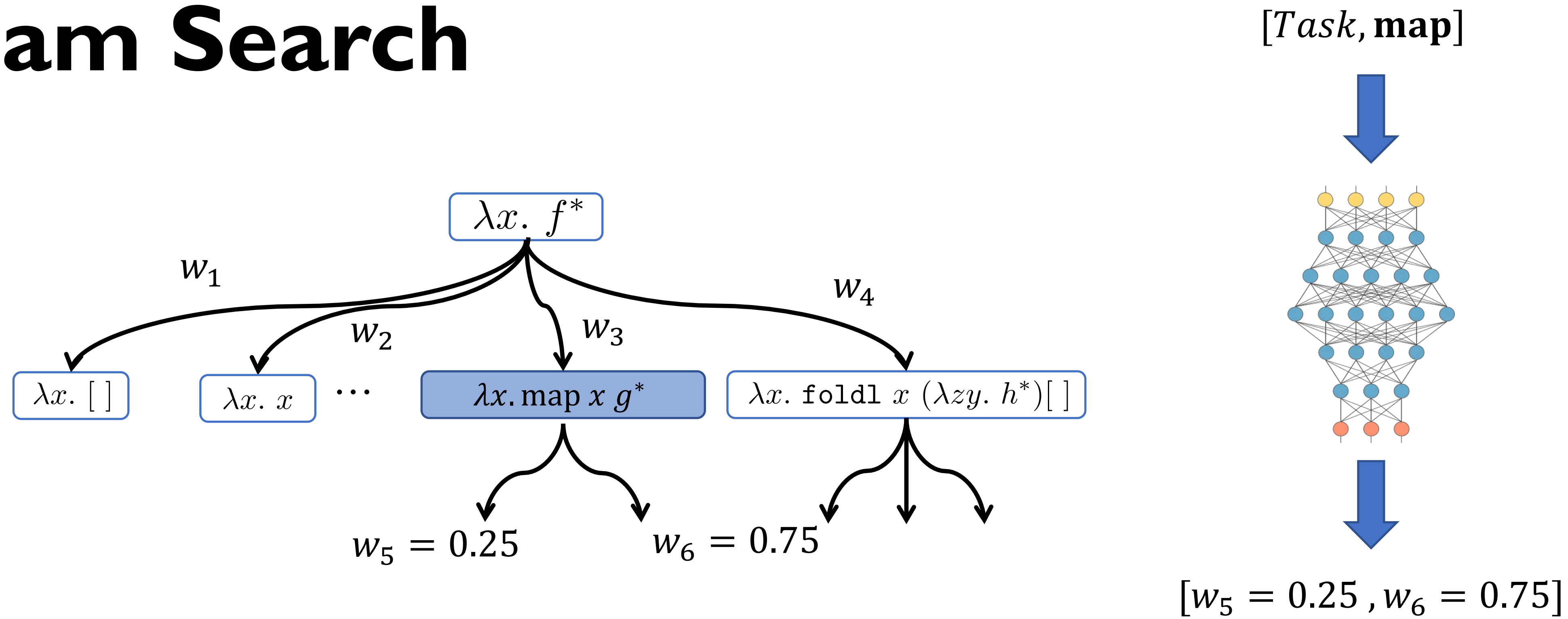
Idea: Learn weights on the search tree from a set of programming tasks.

Simplest Case: Deterministic Greedy



Runtime: greedily follow NN's most preferred predictions

Next Step: Beam Search



Runtime: keep track of top-K most likely sequences

- (e.g., top-K greedy)

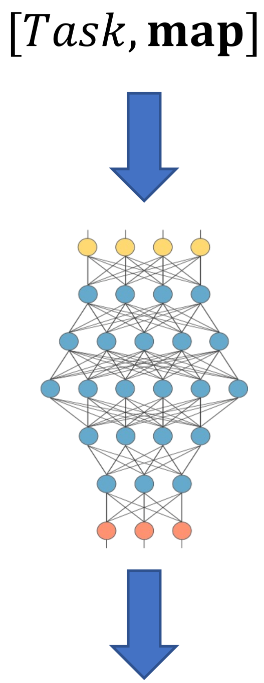
Devlin et al. ***RobustFill: Neural Program Learning Under Noisy I/O.*** ICML 2017.

Why Would This Work?

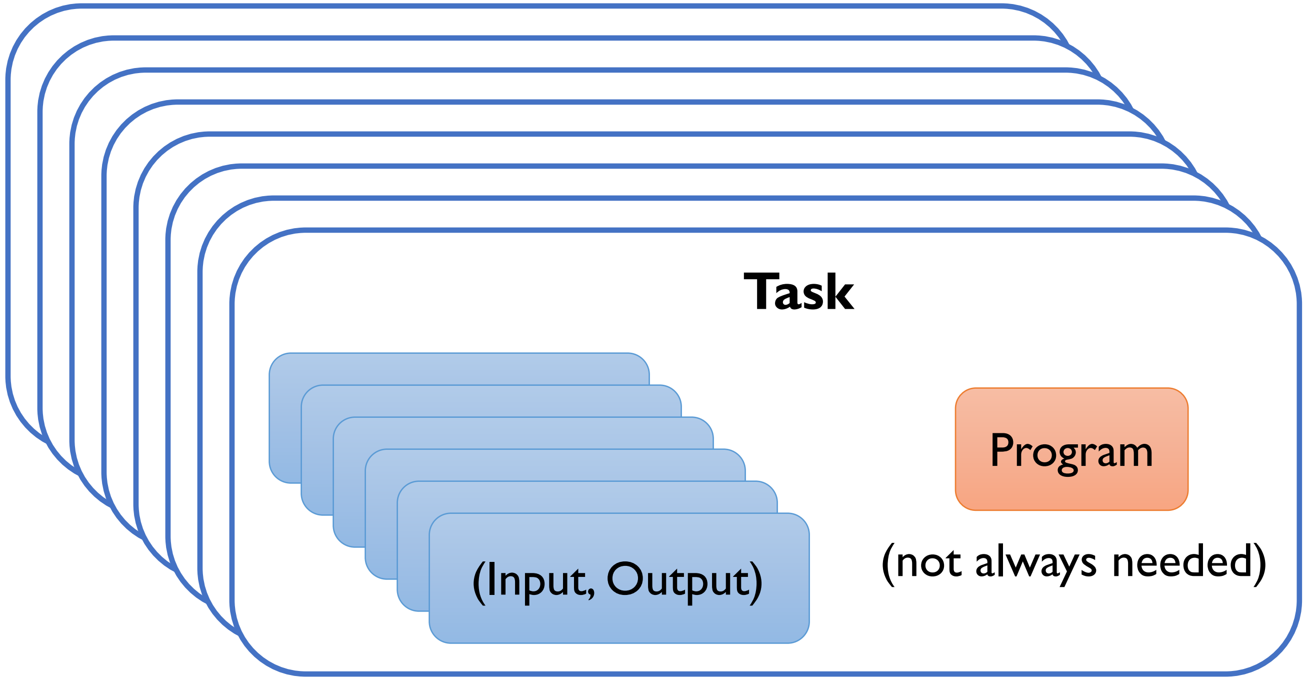
- NN is trained on many related synthesis tasks
  - (unlike Neural Relaxations)

- NN has learned what makes a good completion

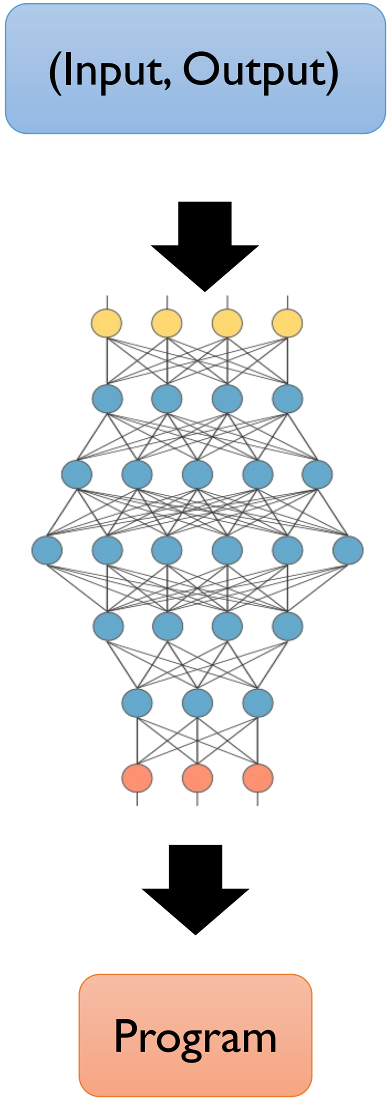
- Thus, can greedily follow NN's predictions
  - (similar to Language Models where NN can complete a prompt)



Learning Setup



E.g., programs and inputs are generated randomly



(grow the program one step at a time)

Devlin et al. ***RobustFill: Neural Program Learning Under Noisy I/O.*** ICML 2017.

Sequence Prediction

Why do children hate the big brown bear?

The big brown bear scares the children with its roar

Q&A

I like artificial intelligence

我喜欢人工智能

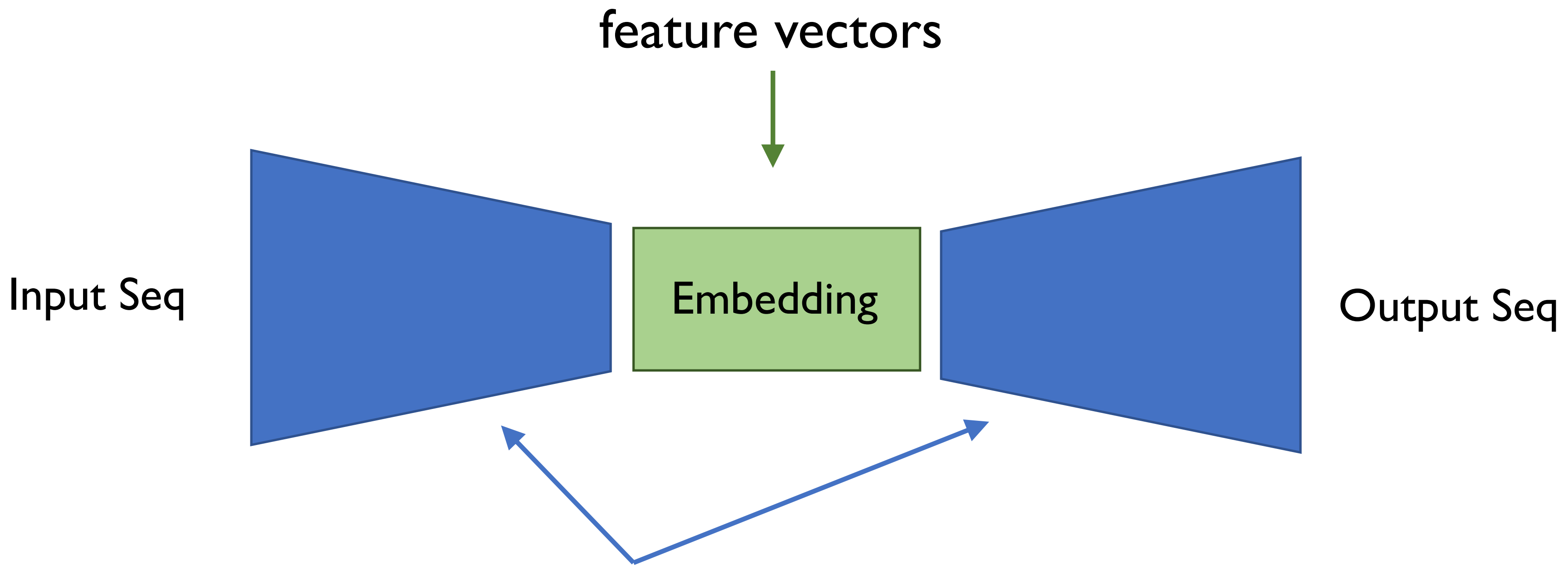
Translation

GPT-3 is a deep neural network that uses the attention mechanism to predict the next word in a sentence.

It is trained on a corpus of over 1 billion words, and can generate text at character level accuracy. GPT-3's architecture consists of two main components…

Completion

Neural Architectures for Sequence Prediction

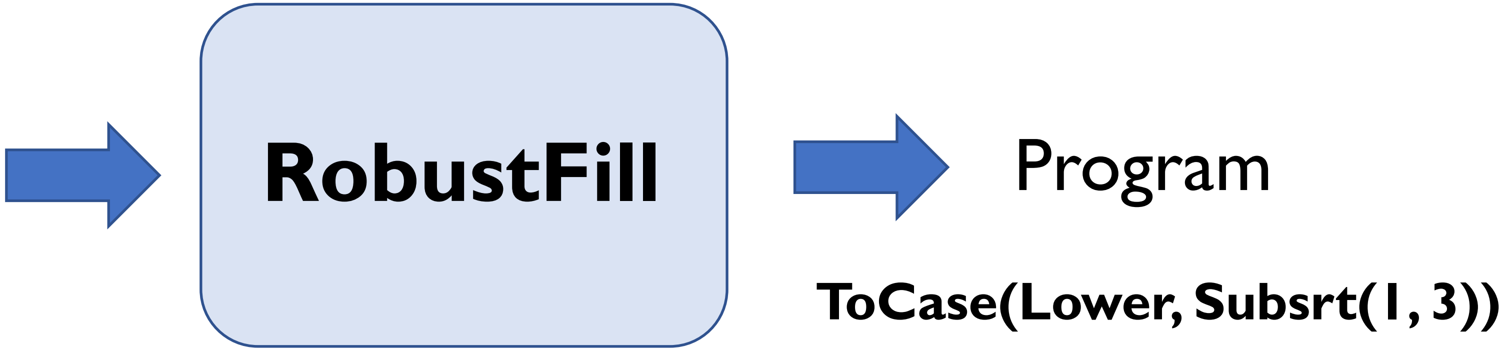


Common Architectures:

- Recurrent NNs
- Transformers (Attention)

(Can handle variable length inputs/outputs)

Input	Output
January	jan
February	feb
March	mar



Output program generated token by token

Devlin et al. ***RobustFill: Neural Program Learning Under Noisy I/O.*** ICML 2017.

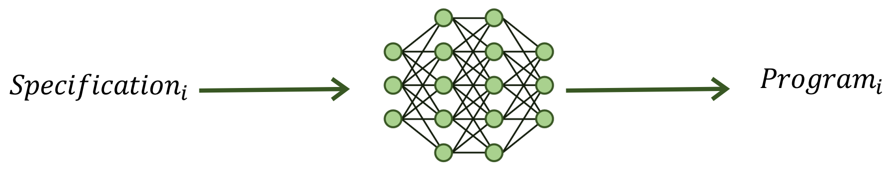
Program

Model prediction: Replace_Space_Comma(GetSpan(Proper, 1, Start, Proper, 4, End) | Const(.) | GetToken_Proper_-1 | EOS

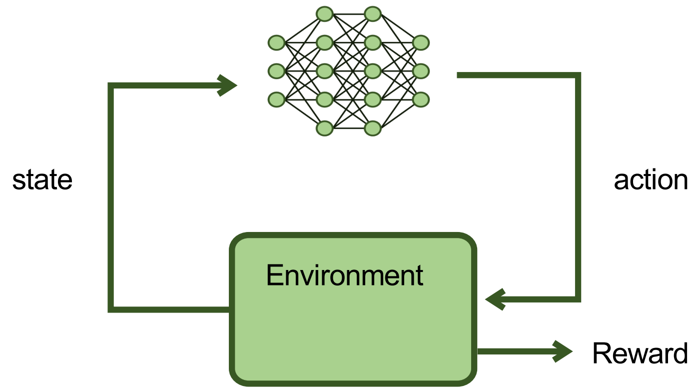
Input	True Output	Program's Output
Jacob Ethan James Alexander Michael	Jacob,Ethan,James,Alexander.- Michael	Jacob,Ethan,James,Alexander.- Michael
Elijah Daniel Aiden Matthew Lucas	Elijah,Daniel,Aiden,Matthew.- Lucas	Elijah,Daniel,Aiden,Matthew.- Lucas
Jackson Oliver Jayden Chris Kevin	Jackson,Oliver,Jayden,Chris.- Kevin	Jackson,Oliver,Jayden,Chris.- Kevin
Earth Fire Wind Water Sun	Earth,Fire,Wind,Water.Sun	Earth,Fire,Wind,Water.Sun
Tom Mickey Minnie Donald Daffy	Tom,Mickey,Minnie,Donald.Daffy	Tom,Mickey,Minnie,Donald.Daffy
Jacob Mickey Minnie Donald Daffy	Jacob,Mickey,Minnie,Donald.- Daffy	Jacob,Mickey,Minnie,Donald.- Daffy
Gabriel Ethan James Alexander Michael	Gabriel,Ethan,James,Alexander- .Michael	Gabriel,Ethan,James,Alexander.- Michael
Rahul Daniel Aiden Matthew Lucas	Rahul,Daniel,Aiden,Matthew.- Lucas	Rahul,Daniel,Aiden,Matthew.- Lucas
Steph Oliver Jayden Chris Kevin	Steph,Oliver,Jayden,Chris.Kevin	Steph,Oliver,Jayden,Chris.Kevin
Pluto Fire Wind Water Sun	Pluto,Fire,Wind,Water.Sun	Pluto,Fire,Wind,Water.Sun

Imitation vs. Reinforcement Learning

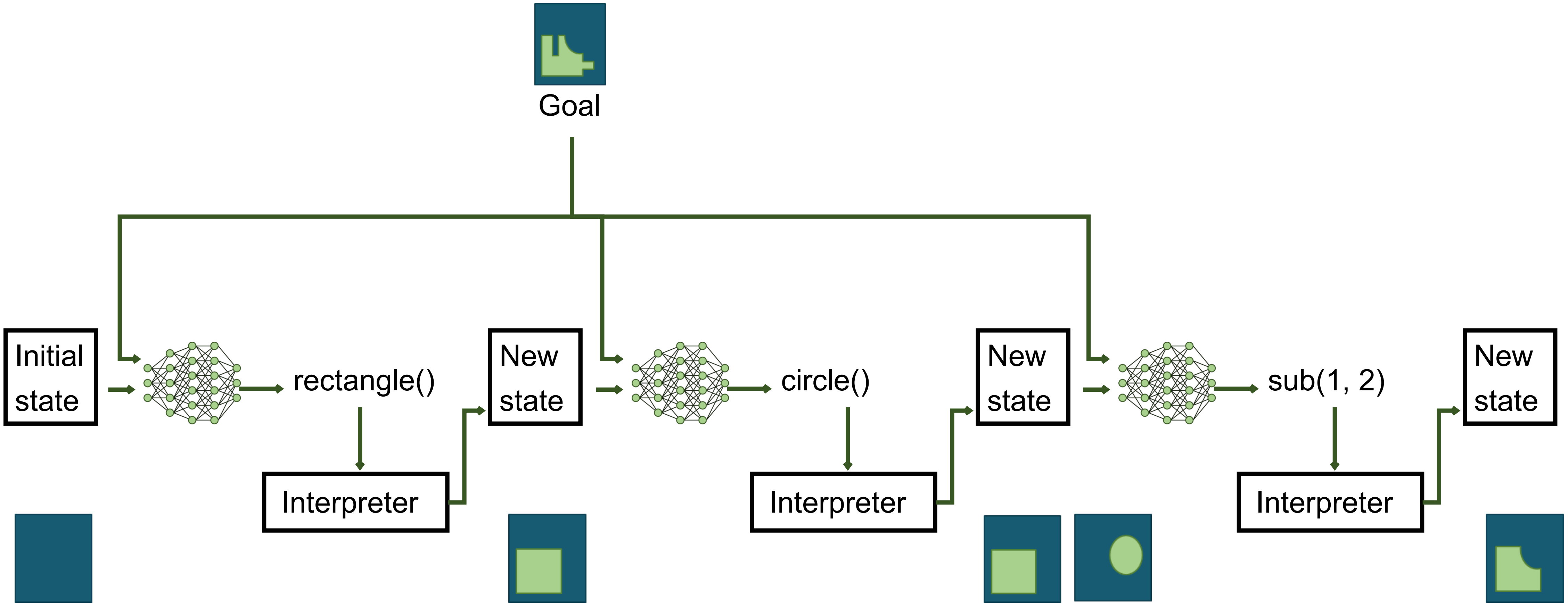
Imitation learning



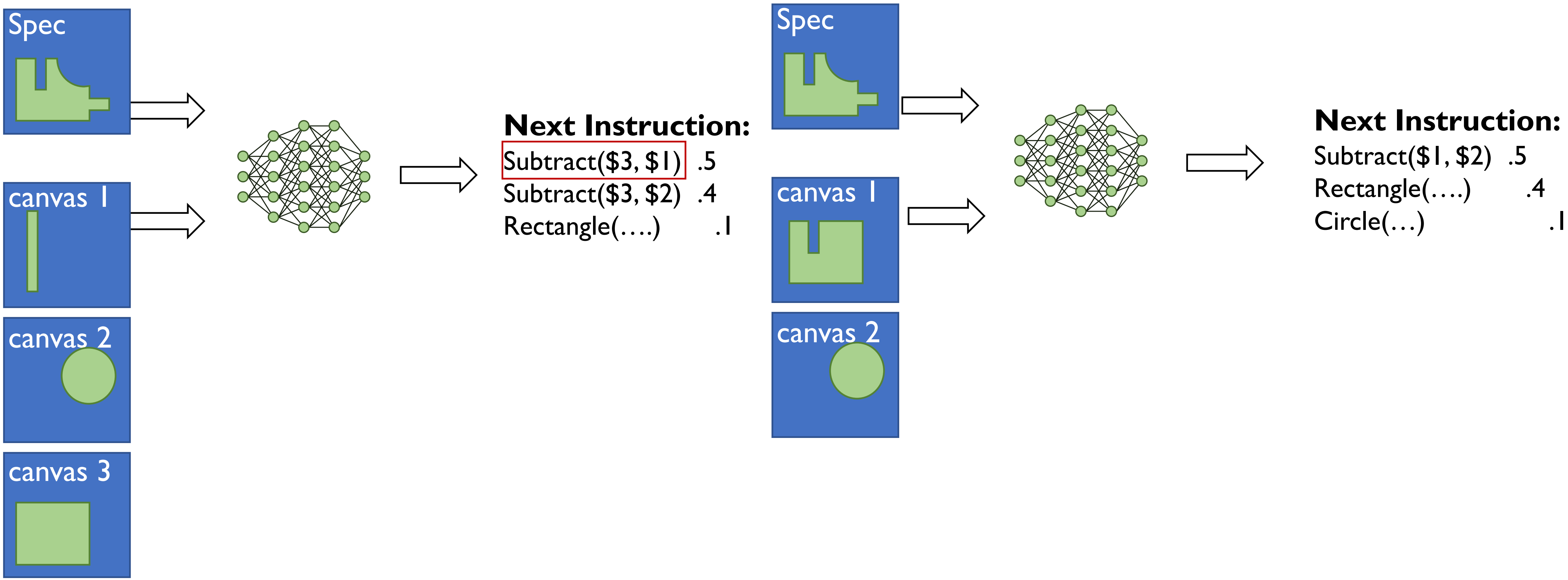
Reinforcement learning



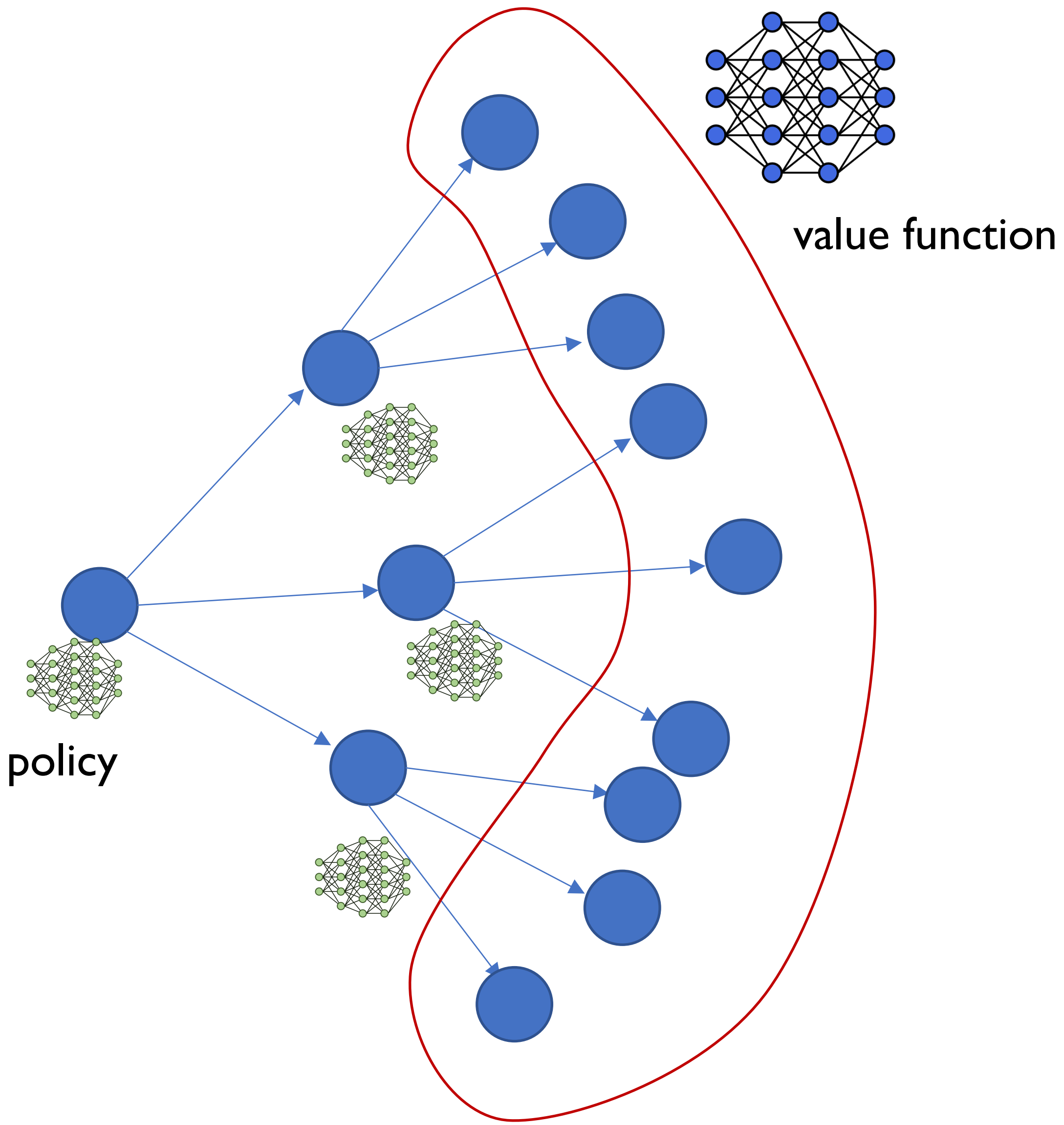
Synthesis with REPL



Learning to synthesize incrementally I



Learning to synthesize incrementally 2



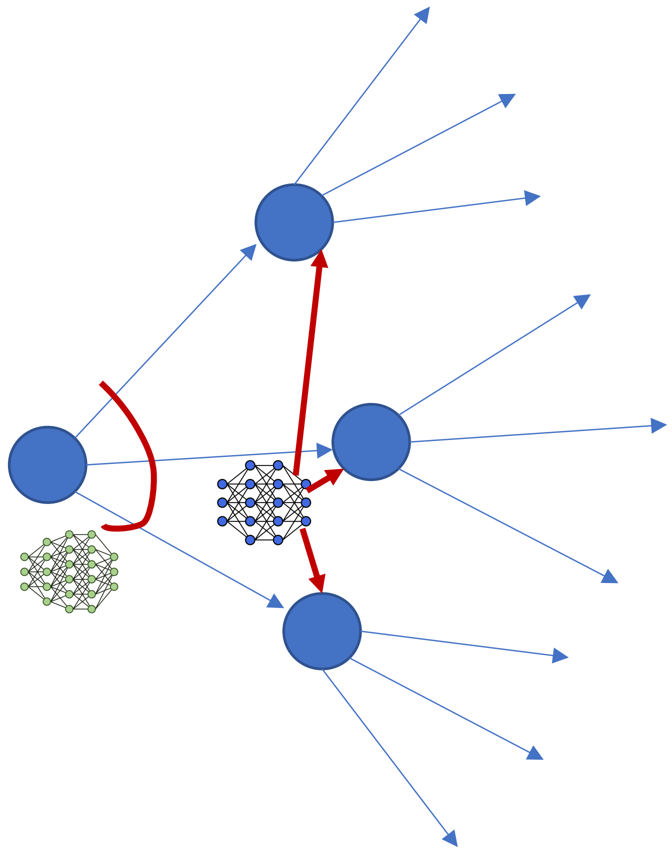
Estimating the "Cost to Go"

- P^* = partial program (non-terminal nodes)
- $\mathbb{C}(P^*)$ = completions of P^* (reachable terminal nodes)

$$\text{Heuristic Estimate: } d(P^*) \approx \overbrace{\min_{P \in \mathbb{C}(P^*)} \left[\underbrace{\Delta s(P, P^*)}_{\text{Additional Structure Cost}} + \underbrace{\min_{\theta} \mathrm{Loss}(\alpha_P, \theta_P)}_{\text{Training Loss}} \right]}^{\text{“Cost to Go”}}$$

- If $d(P^*)$ is a lower bound it becomes an "admissible heuristic"

Guiding program search

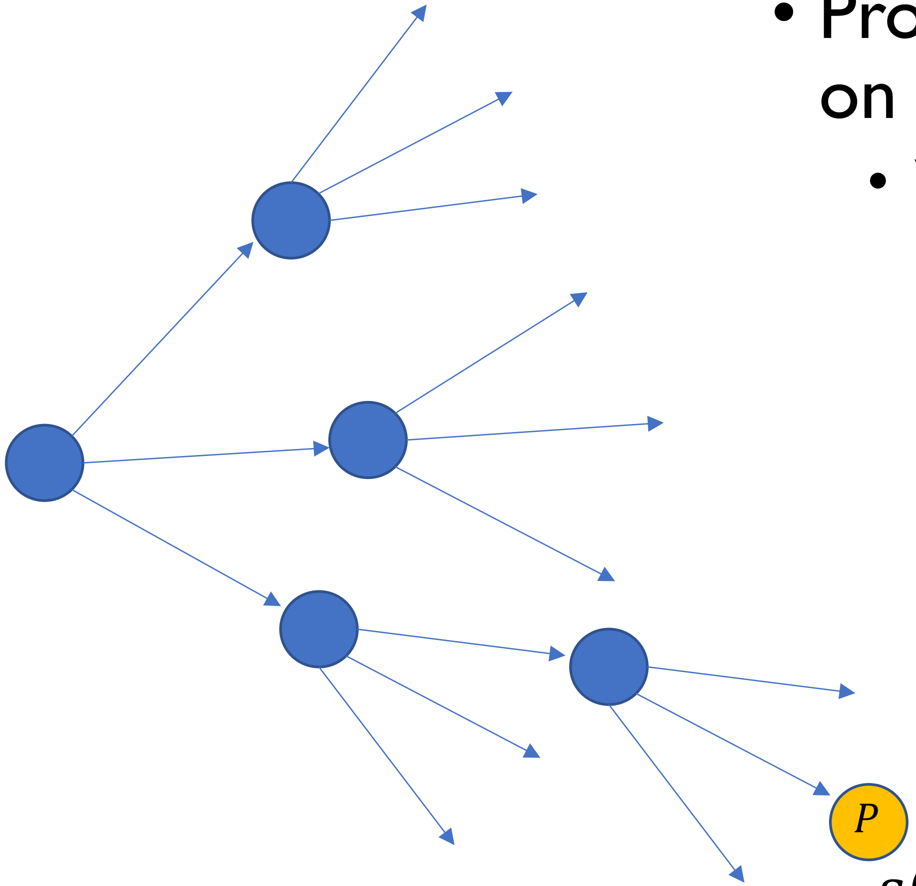


So far

- Neural network policy to guide search
- Value function scores individual states

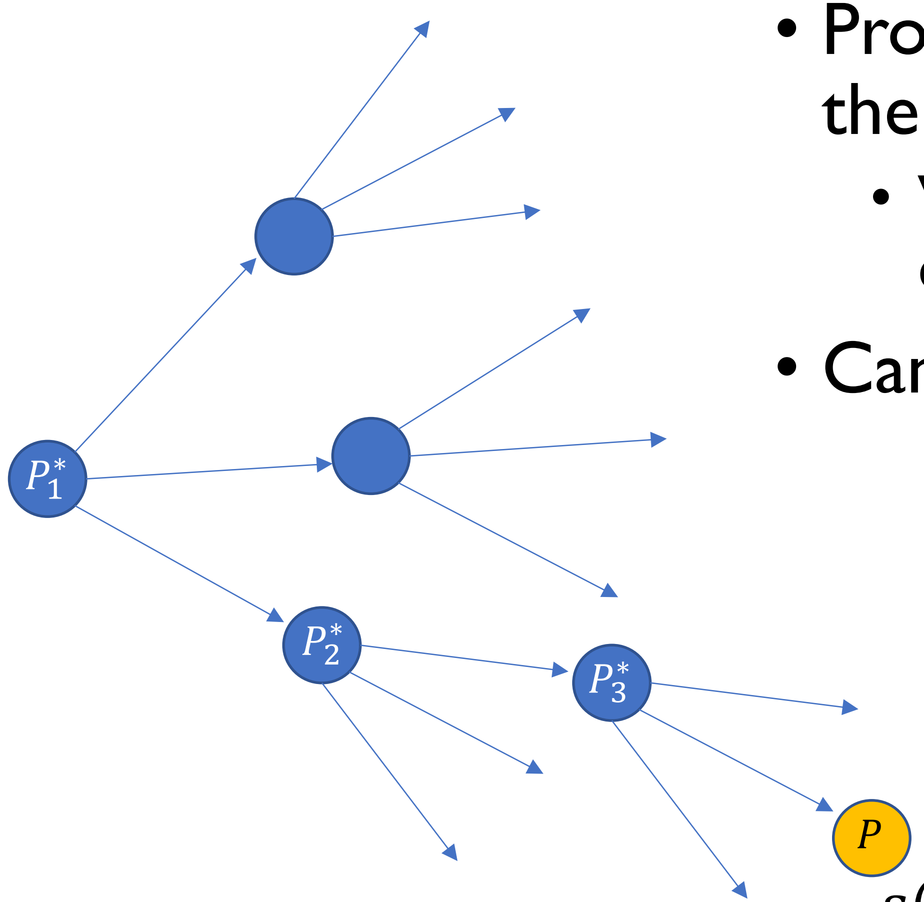
Guiding program search

- Problem: You only get ground truth on the leaves of the search tree
  - Value for an intermediate node is only an estimate



$$s(P) + \min_{\theta} \mathrm{Loss}(\alpha_P, \theta_P)$$

Guiding program search



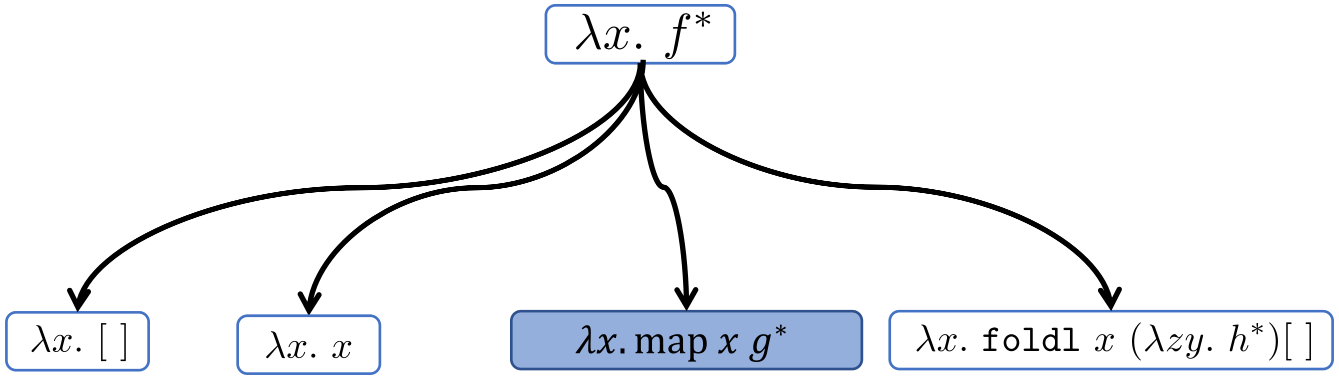
$$s(P) + \min_{\theta} \mathrm{Loss}(\alpha_P, \theta_P)$$

- Problem: You only get ground truth on the leaves of the search tree
  - Value for an intermediate node is only an estimate
- Can we get a better estimate with DL?

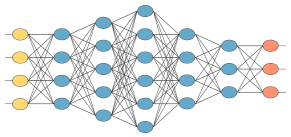
Learning with Neural Heuristics

Shah et al. ***Learning Differentiable Programs with Neural Admissible Heuristics.*** NeurIPS 2020.

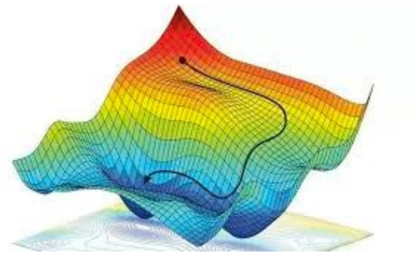
Guiding Search with Neural Relaxations



g^*



Fill hole with NN



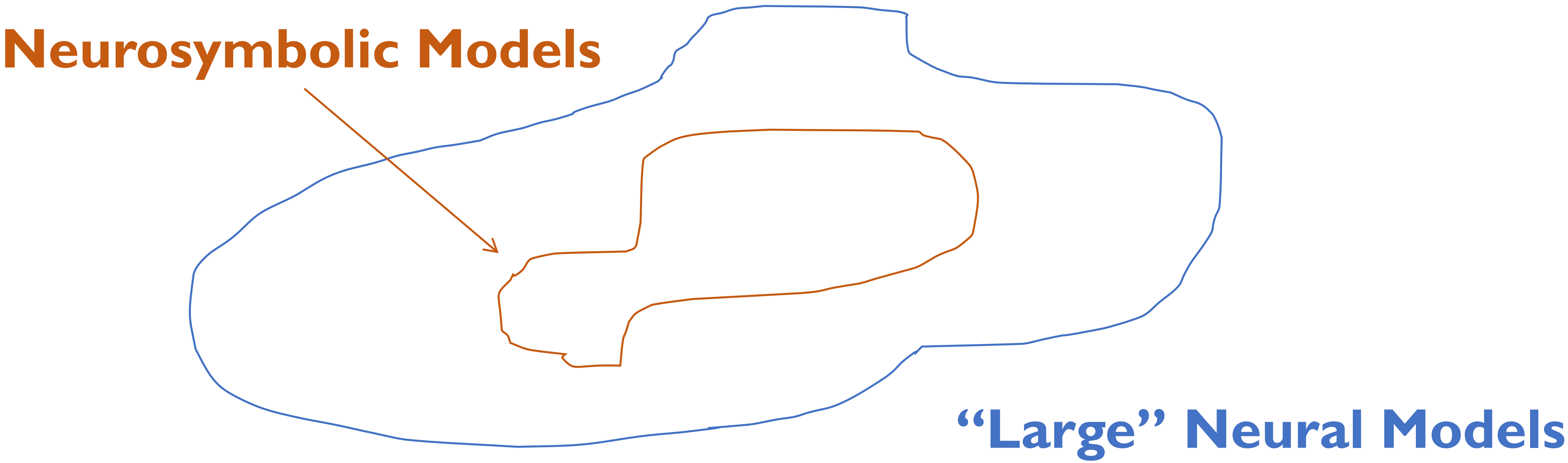
Train parameters

Use training loss to guide search

If a large neural network cannot fit this hole, then a neurosymbolic completion also cannot

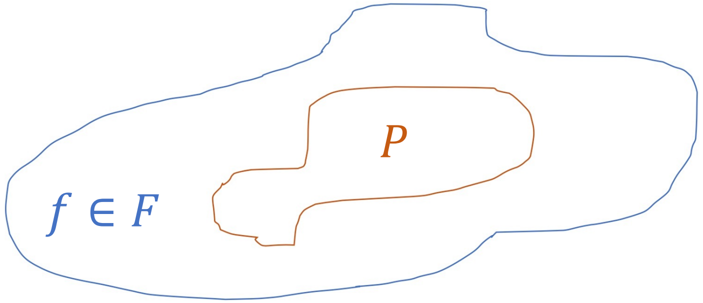
Learning Differentiable Programs with Admissible Neural Heuristics, Ameesh Shah*, Eric Zhan*, et al., NeurIPS 2020

Motivating Observation/Assumption:
Functional Representational Power



“Neural Relaxation” Every neurosymbolic model can be (approximately) represented by some “large” neural model.

Implication
(abstract form)



Slack due to approximation error or training ability

Large Neural

$$\forall P, \exists f \in F \text{ s.t. } d(f) \leq d(P) + \epsilon$$

Neurosymbolic

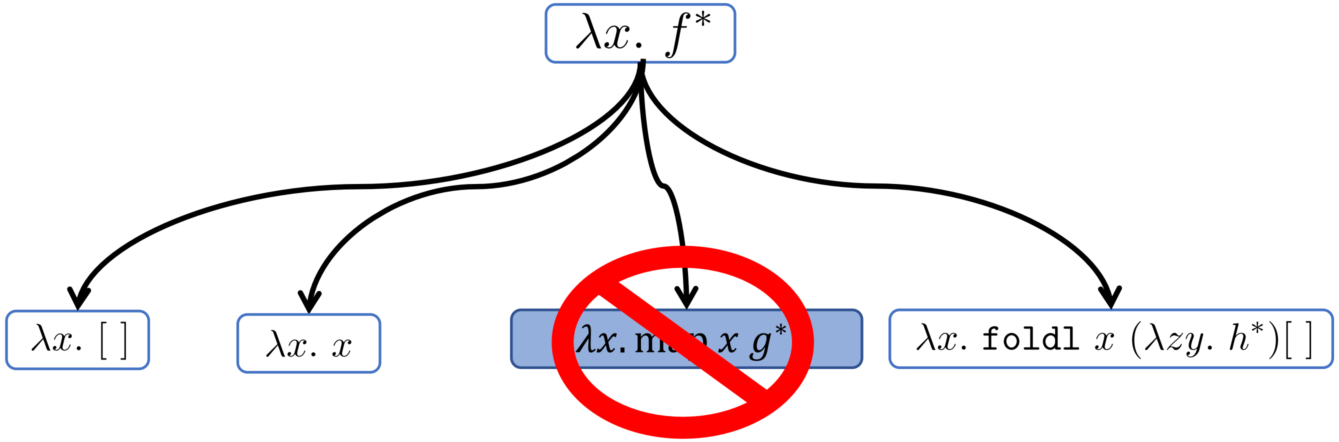
Any Cost Function

We can train an admissible heuristic!

"Neural Relaxation" Every neurosymbolic model can be (approximately) represented by some "large" neural model.

Informed Search (e.g., A*)

- Use $d(P^*)$ to prune the search



Can Prune This Branch!

Suppose:

Structural Cost

Training Loss

$$s(\lambda x. map\ x\ g^*) + d(\lambda x. map\ x\ g^*) > s(\lambda x.\ x) + \mathrm{Loss}(\lambda x.\ x)$$

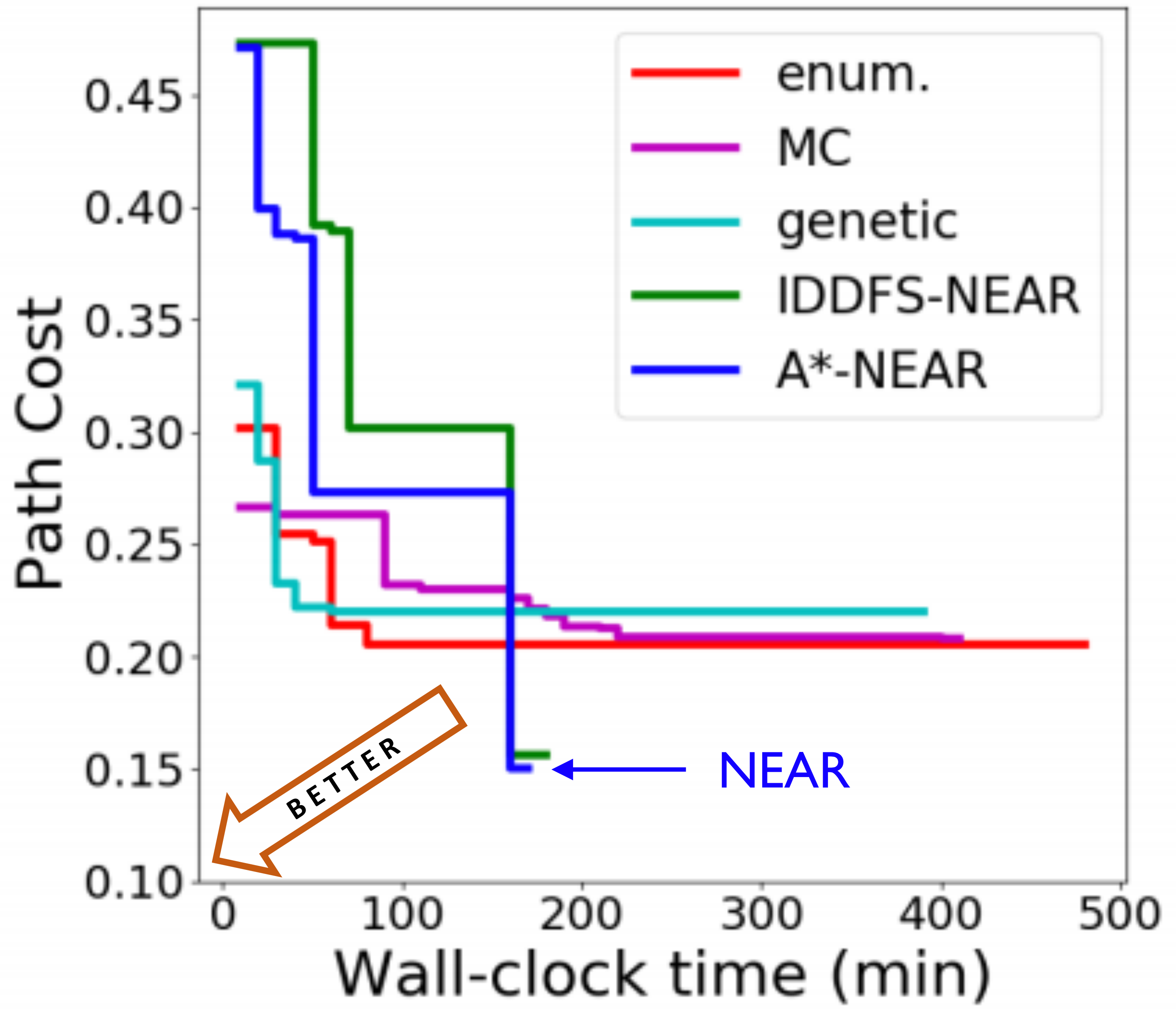
“Cost to Go” Heuristic

A* Search

- Priority queue of current leaf nodes:
  - Sorted by $s(P^*) + d(P^*)$
- Pop off top program P^*
  - If P^* is complete, terminate
  - Else, expand P^*, add child nodes to priority queue

- **Guarantee:** if $d(P^*)$ is $\underbrace{\text{admissible}}^{\text{Lower bounds "Cost to Go"}}$, A* will return optimal P
  - Tighter $d(P^*)$ prunes more aggressively
  - Uninformed $d(P^*)$ (e.g., always 0) => uninformed search

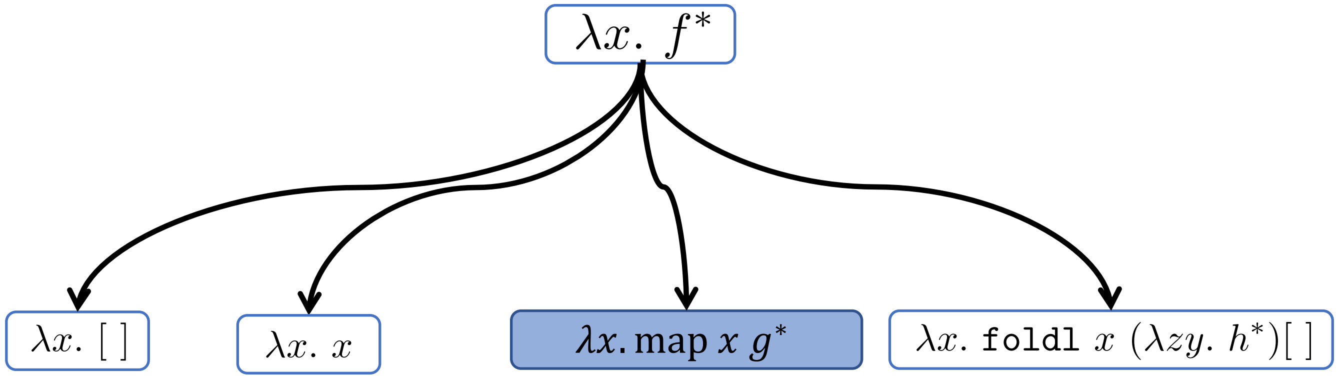
NEAR: Results



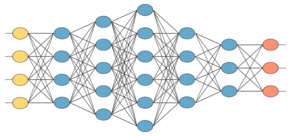
Order of magnitude speedup!

Learning Differentiable Programs with Admissible Neural Heuristics, Ameesh Shah*, Eric Zhan*, et al., NeurIPS 2020

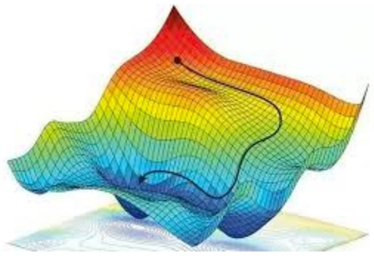
NEAR: Neural Admissible Relaxations



g^*



Fill hole with NN



Train parameters

Use training loss as admissible heuristic

If a large neural network cannot fit this hole, then a neurosymbolic completion also cannot

Learning Differentiable Programs with Admissible Neural Heuristics, Ameesh Shah*, Eric Zhan*, et al., NeurIPS 2020

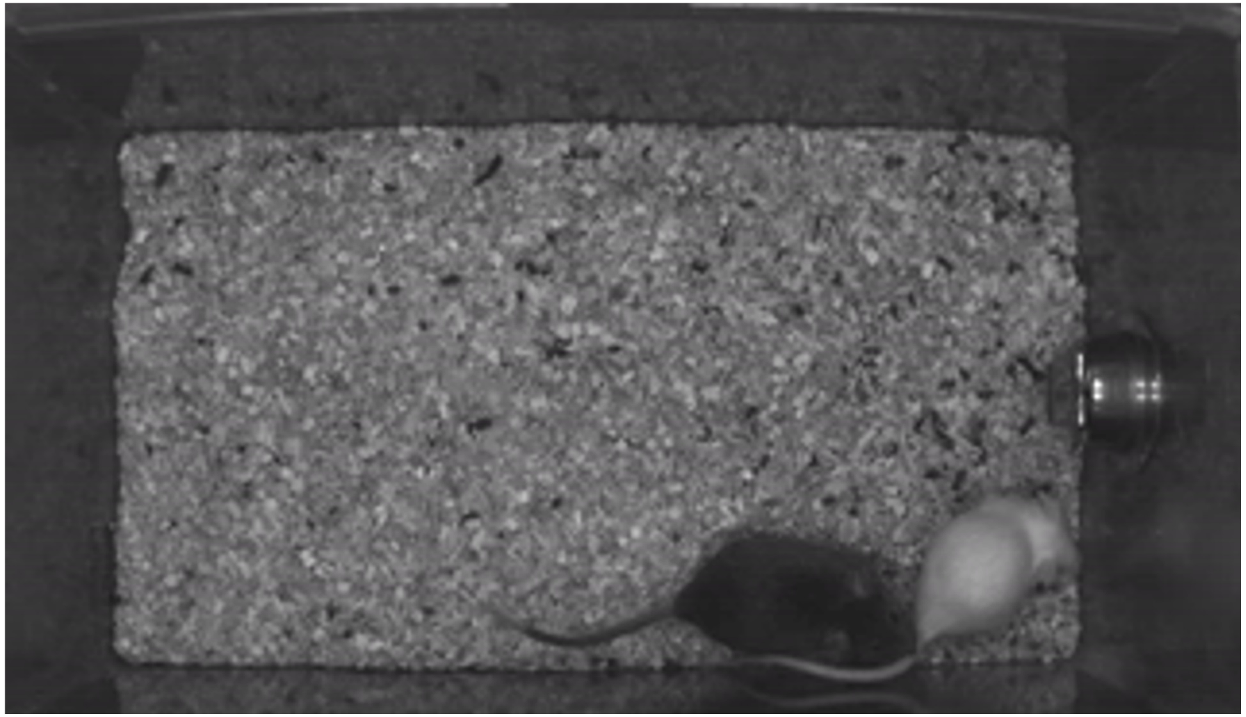
Summary

- Today, we saw two strategies for learning neurosymbolic programs
  - Type-directed enumeration, informed search via admissible neural heuristics

- Next: Deep Dive Continued
  - Code for enumeration & NEAR

Behavior Quantification

How to categorize behavior at each frame?



Code: Use neurosymbolic programming to learn relationship between pose and behavior

CalMS21 Dataset: https://arxiv.org/pdf/2104.02710.pdf

Code Structure: Enumeration

- Running Enumeration
  - Base DSL
  - Morlet Filter DSL
  - Neurosymbolic DSL
- Visualize Runtime vs. Classification Performance
- Implement Temporal Filter
- Open-Ended Exploration

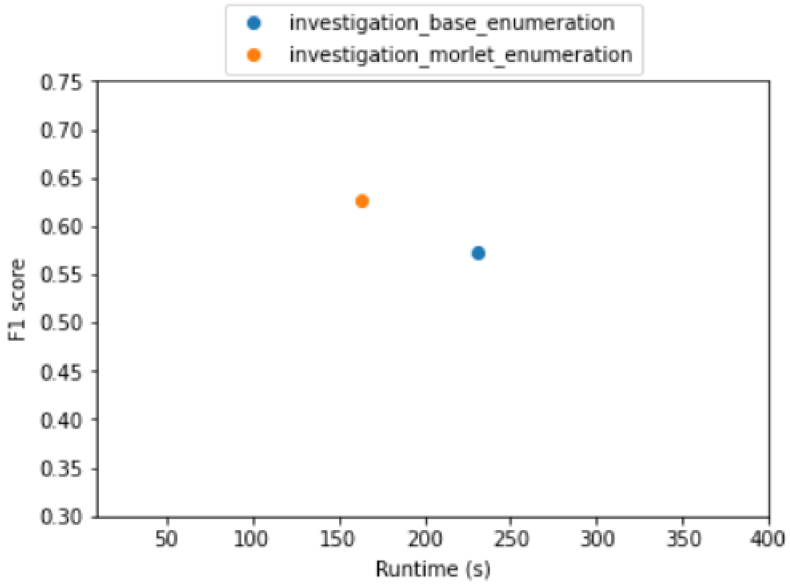
```
!yes | python train.py \
--algorithm enumeration \
--exp_name investigation_base \
--trial 1 \
--seed 1 \
--dsl_str "default" \
--train_data "data/calms21_task1/train_data.npy" \
--test_data "data/calms21_task1/test_data.npy" \
--valid_data "data/calms21_task1/val_data.npy" \
--train_labels "data/calms21_task1/train_investigation_labels.npy" \
--test_labels "data/calms21_task1/test_investigation_labels.npy" \
--valid_labels "data/calms21_task1/val_investigation_labels.npy" \
--input_type "list" \
--output_type "atom" \
--input_size 18 \
--output_size 1 \
--num_labels 1 \
--lossfxn "bcelogits" \
--learning_rate 0.0001 \
--symbolic_epochs 12 \
--max_num_programs 25 \
--class_weights "2.0"
```

```
Evaluating program Start(MorletFilterOp(OverlapBboxesSelect())) on TEST SET
F1 score achieved is 0.6272
Additional performance parameters: {'hamming_accuracy': 0.74, 'all_f1s': array([0.80040942, 0.62715105])}
```

Code Structure: Enumeration

- Running Enumeration
  - Base DSL
  - Morlet Filter DSL
  - Neurosymbolic DSL
- Visualize Runtime vs. Classification Performance
- Implement Temporal Filter
- Open-Ended Exploration

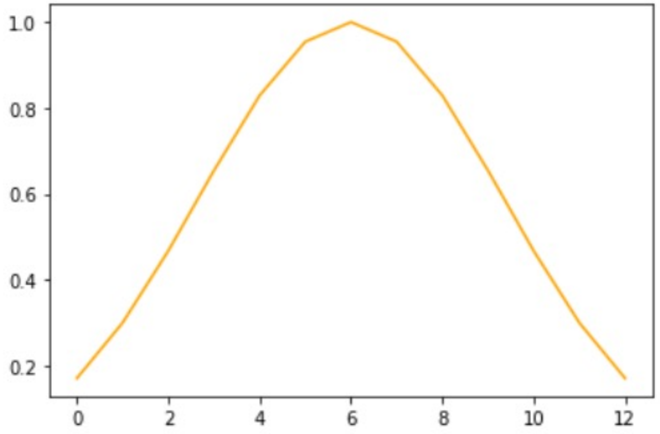
```
# Directory names to plot inside near_code/results
run_names_to_plot = ['investigation_base_enumeration_sd_1_001',
                     'investigation_morlet_enumeration_sd_1_001']

runtime, f1 = parse_runtime_f1_from_logs(run_names_to_plot)

plot_runtime_f1(runtime, f1, run_names_to_plot)
```



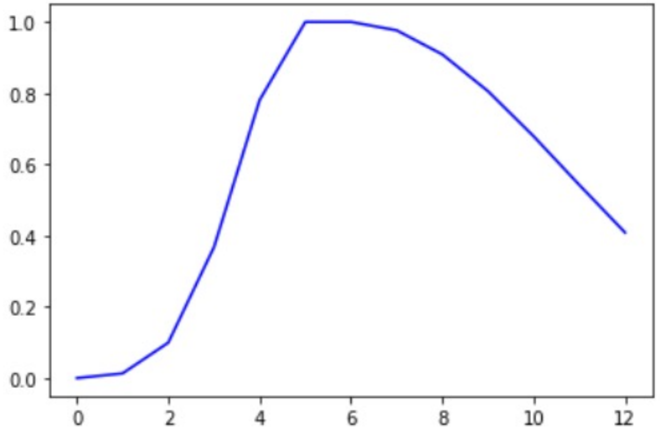
Code Structure: Enumeration

- Running Enumeration
  - Base DSL
  - Morlet Filter DSL
  - Neurosymbolic DSL
- Visualize Runtime vs. Classification Performance
- Implement Temporal Filter
- Open-Ended Exploration

Symmetric Filter (provided)



Asymmetric Filter (implement)



Code Structure: NEAR

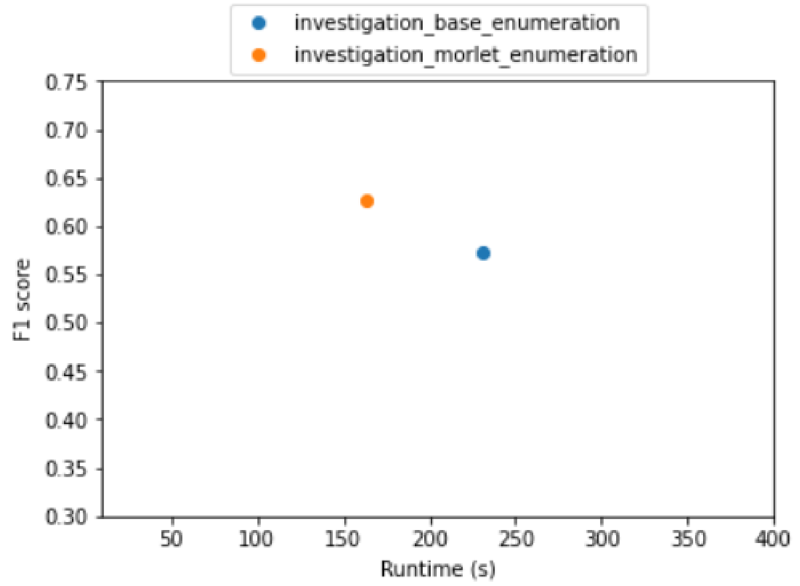
- Running NEAR
  - Base DSL
  - Morlet Filter DSL
- Visualize Runtime vs. Classification Performance
- Open-Ended Exploration:
  - Modifying Heuristic Architecture
  - IDDFS Search
  - Test on Other Behaviors

```
!yes| python train.py \
--algorithm astar-near \
--exp_name investigation_base \
--trial 1 \
--seed 1 \
--dsl_str "default" \
--train_data "data/calms21_task1/train_data.npy" \
--test_data "data/calms21_task1/test_data.npy" \
--valid_data "data/calms21_task1/val_data.npy" \
--train_labels "data/calms21_task1/train_investigation_labels.npy" \
--test_labels "data/calms21_task1/test_investigation_labels.npy" \
--valid_labels "data/calms21_task1/val_investigation_labels.npy" \
--input_type "list" \
--output_type "atom" \
--input_size 18 \
--output_size 1 \
--num_labels 1 \
--lossfxn "bcelogits" \
--frontier_capacity 8 \
--max_num_children 10 \
--max_depth 5 \
--max_num_units 32 \
--min_num_units 16 \
--learning_rate 0.0001 \
--neural_epochs 4 \
--symbolic_epochs 12 \
--class_weights "2.0"
```

Code Structure: NEAR

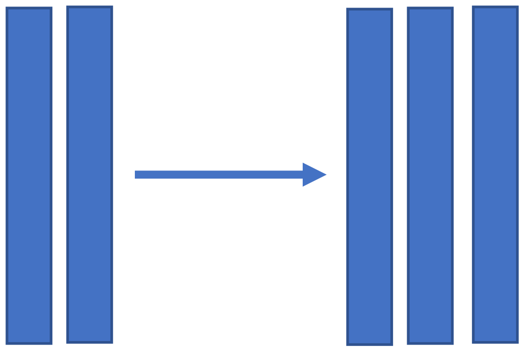
- Running NEAR
  - Base DSL
  - Morlet Filter DSL
- Visualize Runtime vs. Classification Performance
- Open-Ended Exploration:
  - Modifying Heuristic Architecture
  - IDDFS Search
  - Test on Other Behaviors

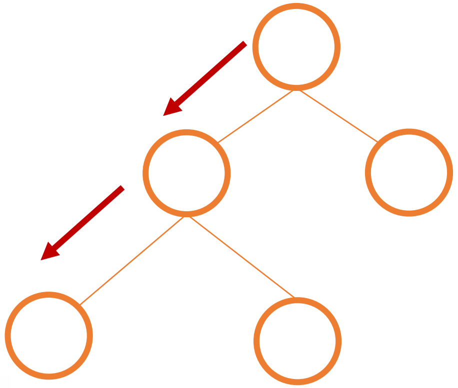
```
# Directory names to plot inside near_code/results
run_names_to_plot = ['investigation_base_enumeration_sd_1_001',
                     'investigation_morlet_enumeration_sd_1_001']

runtime, f1 = parse_runtime_f1_from_logs(run_names_to_plot)

plot_runtime_f1(runtime, f1, run_names_to_plot)
```



Code Structure: NEAR

- Running NEAR
  - Base DSL
  - Morlet Filter DSL
- Visualize Runtime vs. Classification Performance
- Open-Ended Exploration
  - Modifying Heuristic Architecture
  - Different Search Algorithms
  - Test on Other Behaviors





Inside code_and_data…

dsl.ipynb:
contains DSLs

Table of contents

- **Neural Functions**
  - Generic Library Functions
  - Behavior Library Functions
    - Mice Feature Subsets
    - Morlet Filter
      - Test Morlet Filter
  - Starting DSL Dictionary
  - DSL with Morlet Filter
  - DSL with Neural Module
  - DSL with Asymmetric Morlet Filter

near.ipynb:
contains search algorithms

Table of contents

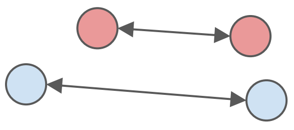
- **NEAR Code**
  - Utilities
    - Data Processing
    - Logging
    - Training
    - Evaluation
  - Program Graph
  - Search Algorithm
    - ASTAR
    - IDDFS
    - Enumeration

Potential Areas to Explore

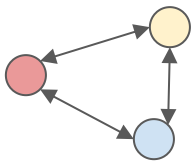
Effect of search hyperparameters

```
!yes| python train.py \
--algorithm astar-near \
--exp_name investigation_base \
--trial 1 \
--seed 1 \
--dsl_str "default" \
--train_data "data/calms21_task1/train_data.npy" \
--test_data "data/calms21_task1/test_data.npy" \
--valid_data "data/calms21_task1/val_data.npy" \
--train_labels "data/calms21_task1/train_investigation_labels.npy" \
--test_labels "data/calms21_task1/test_investigation_labels.npy" \
--valid_labels "data/calms21_task1/val_investigation_labels.npy" \
--input_type "list" \
--output_type "atom" \
--input_size 18 \
--output_size 1 \
--num_labels 1 \
--lossfxn "bcelogits" \
--frontier_capacity 8 \
--max_num_children 10 \
--max_depth 5 \
--max_num_units 32 \
--min_num_units 16 \
--learning_rate 0.0001 \
--neural_epochs 4 \
--symbolic_epochs 12 \
--class_weights "2.0"
```

dsl.ipynb: modify DSLs

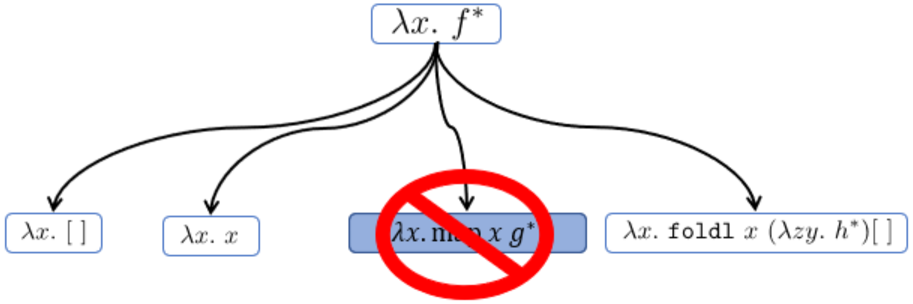


Speed



Distance

near.ipynb: modify search algorithms



Code Walk-Through

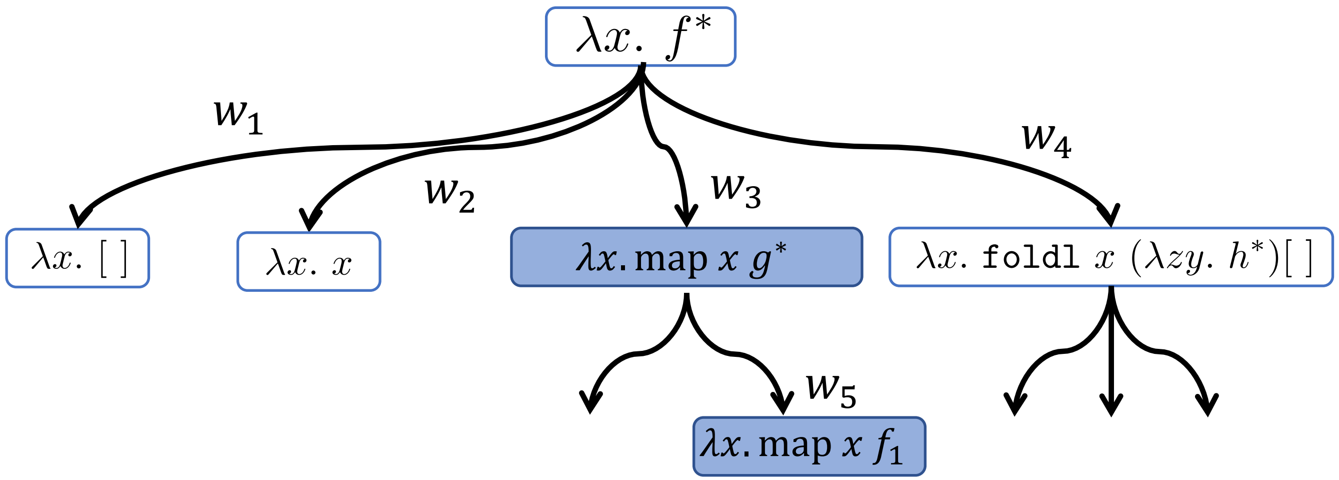
bit.ly/neurosym_tutorial

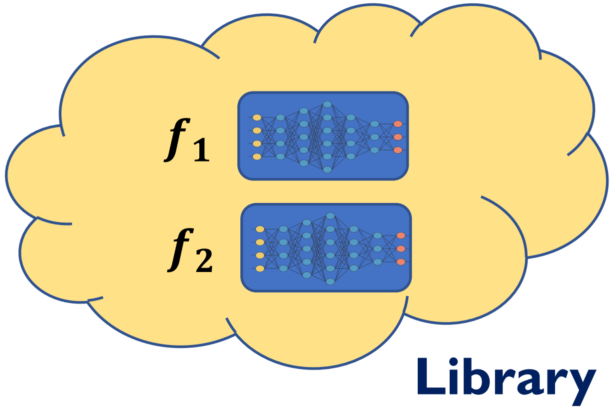
Outline of Tutorial

1. What is Neurosymbolic Programming?
2. Deep Dive: Neurosymbolic Programming for Science
3. Algorithmic Techniques
4. Deep Dive (continued)
5. Algorithmic Techniques (continued)
6. Conclusion

Algorithmic Techniques (continued)

Recall: Searching over program structures





How to search over combinatorial space?

Recall: Informed Search

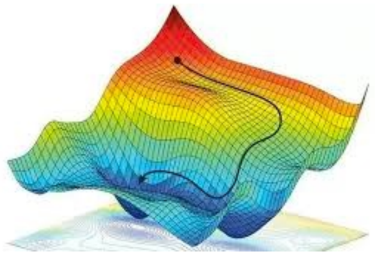
via Neural Relaxation Admissible Heuristic

$\lambda x.\ f^*$

$\lambda x.\ [\,]$ | $\lambda x.\ x$ | $\lambda x.\, \mathrm{map}\ x\ g^*$ | $\lambda x.\ \texttt{foldl}\ x\ (\lambda zy.\ h^*)[\,]$

g^* g^*

Fill hole with NN



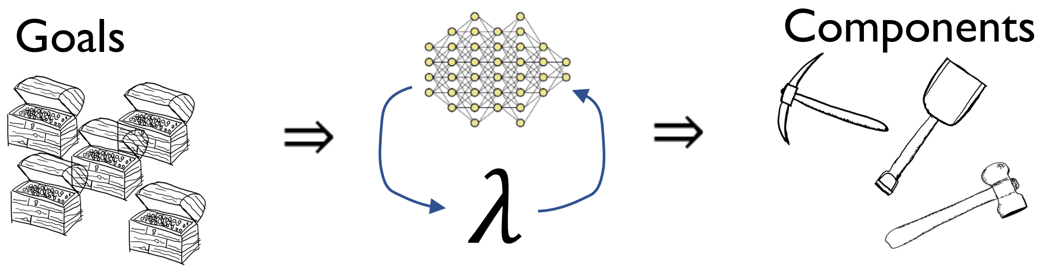
Train parameters

Use training loss as admissible heuristic

If a large neural network cannot fit this hole, then a neurosymbolic completion also cannot

Learning Differentiable Programs with Admissible Neural Heuristics, Ameesh Shah*, Eric Zhan*, et al., NeurIPS 2020

Library Learning



(Based on slides by Kevin Ellis and the work in [Ellis et al. 2021])

Library Learning

Initial Primitives

⋮

`map`

`fold`

`if`

`cons`

`>`

⋮

Sample Problem: `sort list`

[9 2 7 1] → [1 2 7 9]

[3 8 9 4 2] → [2 3 4 8 9]

[6 2 2 3 8 5] → [2 2 3 5 6 8]

...

Ellis, Morales, Sable-Meyer, Solar-Lezama, Tenenbaum. NeurIPS 2018.

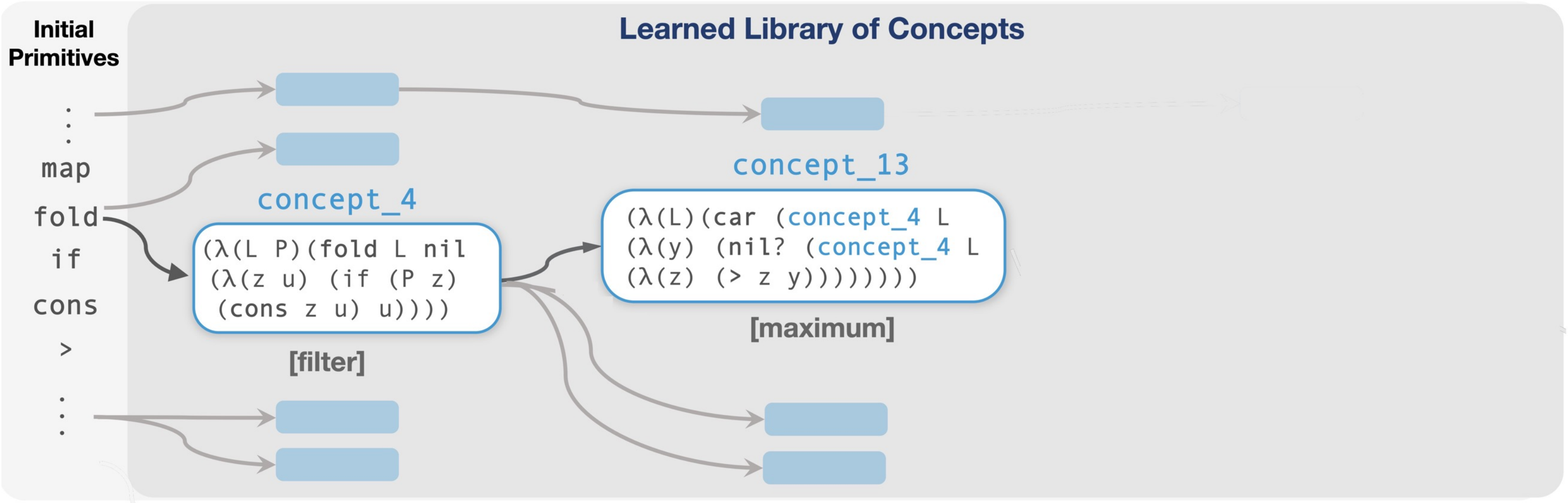
Ellis, Wong, Nye, ..., Solar-Lezama, Tenenbaum. 2020.

Library Learning

Sample Problem: `sort list`

[9 2 7 1] → [1 2 7 9]
[3 8 9 4 2]→ [2 3 4 8 9]
[6 2 2 3 8 5]→ [2 2 3 5 6 8]
...

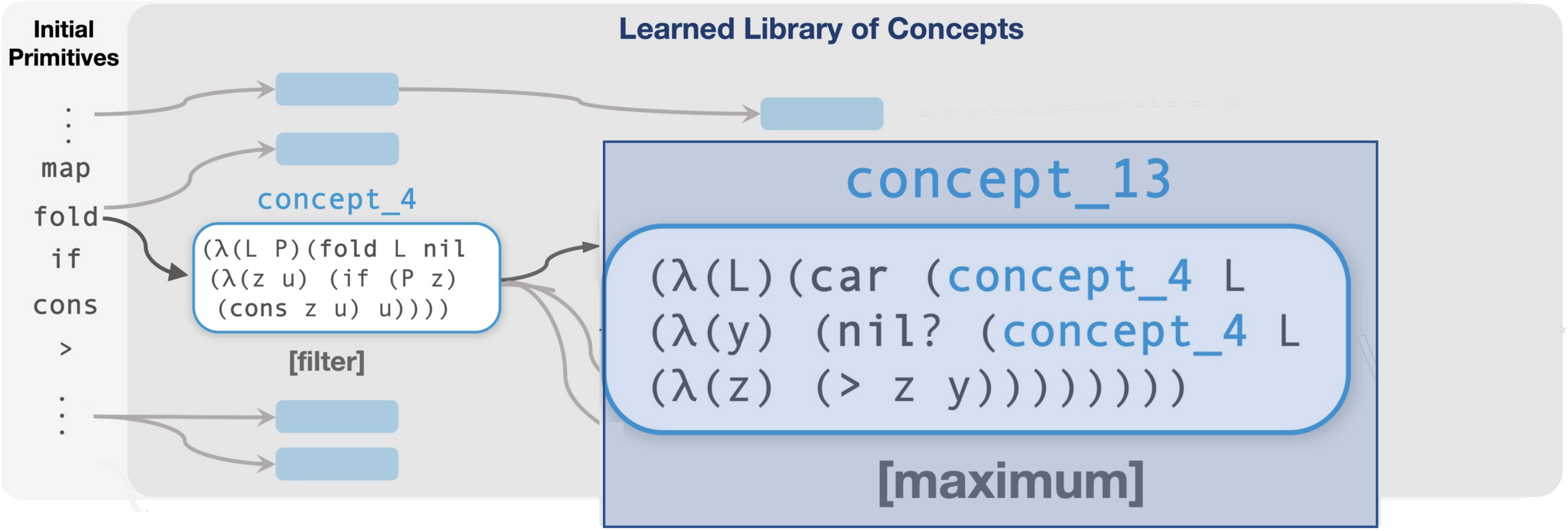
Library Learning



Sample Problem: `sort list`

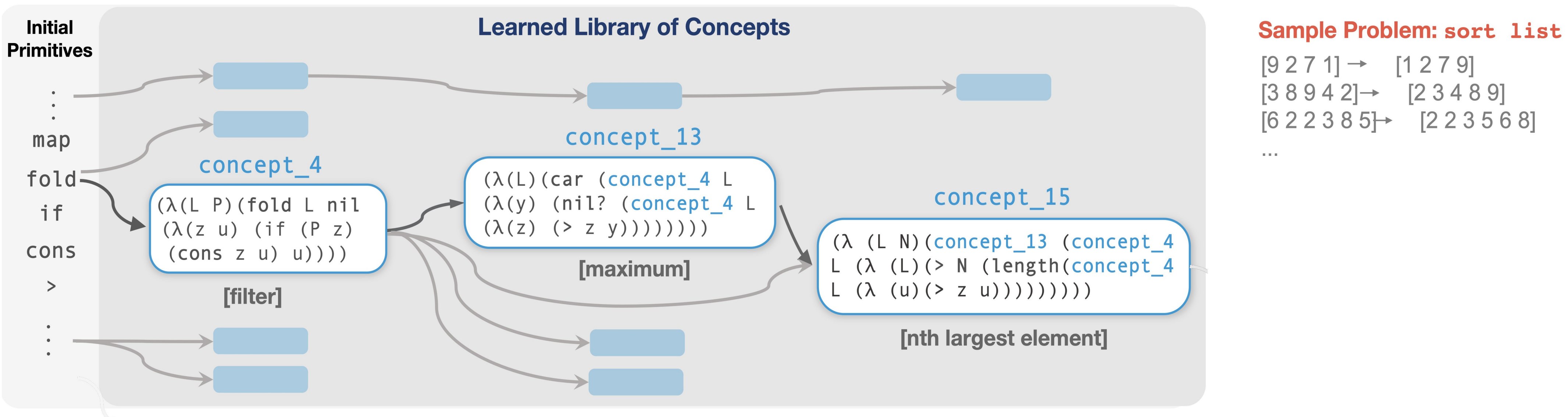
[9 2 7 1] → [1 2 7 9]
[3 8 9 4 2] → [2 3 4 8 9]
[6 2 2 3 8 5] → [2 2 3 5 6 8]
...

Library Learning



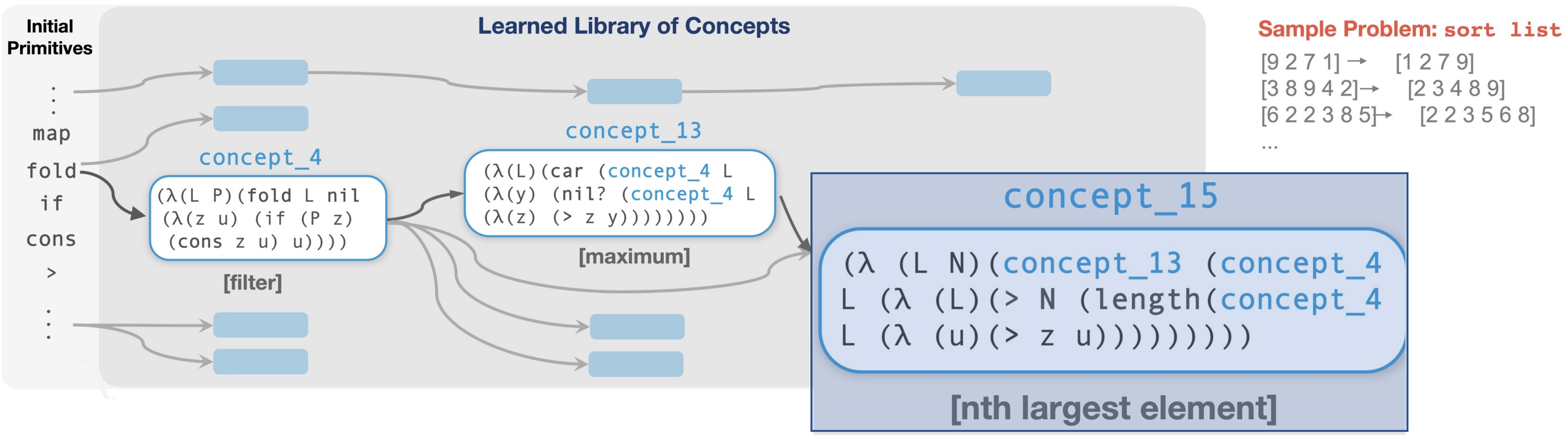
Sample Problem: `sort list`

[9 2 7 1] → [1 2 7 9]
[3 8 9 4 2] → [2 3 4 8 9]
[6 2 2 3 8 5] → [2 2 3 5 6 8]
...

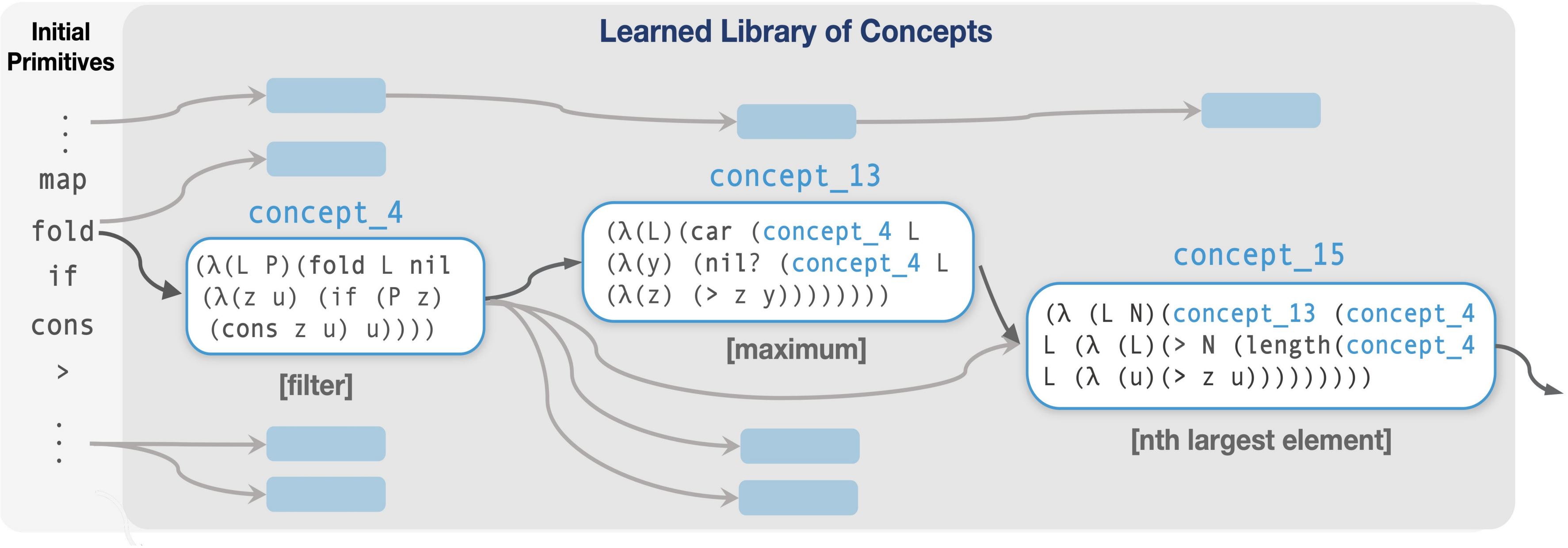


Initial Primitives
map
fold
if
cons
>
Learned Library of Concepts
concept_4
(λ(L P)(fold L nil (λ(z u) (if (P z) (cons z u) u))))
[filter]
concept_13
(λ(L)(car (concept_4 L (λ(y) (nil? (concept_4 L (λ(z) (> z y))))))))
[maximum]
concept_15
(λ (L N)(concept_13 (concept_4 L (λ (L)(> N (length(concept_4 L (λ (u)(> z u)))))))))
[nth largest element]
Sample Problem: sort list
[9 2 7 1] → [1 2 7 9]
[3 8 9 4 2]→ [2 3 4 8 9]
[6 2 2 3 8 5]→ [2 2 3 5 6 8]
...

Library Learning



Library Learning



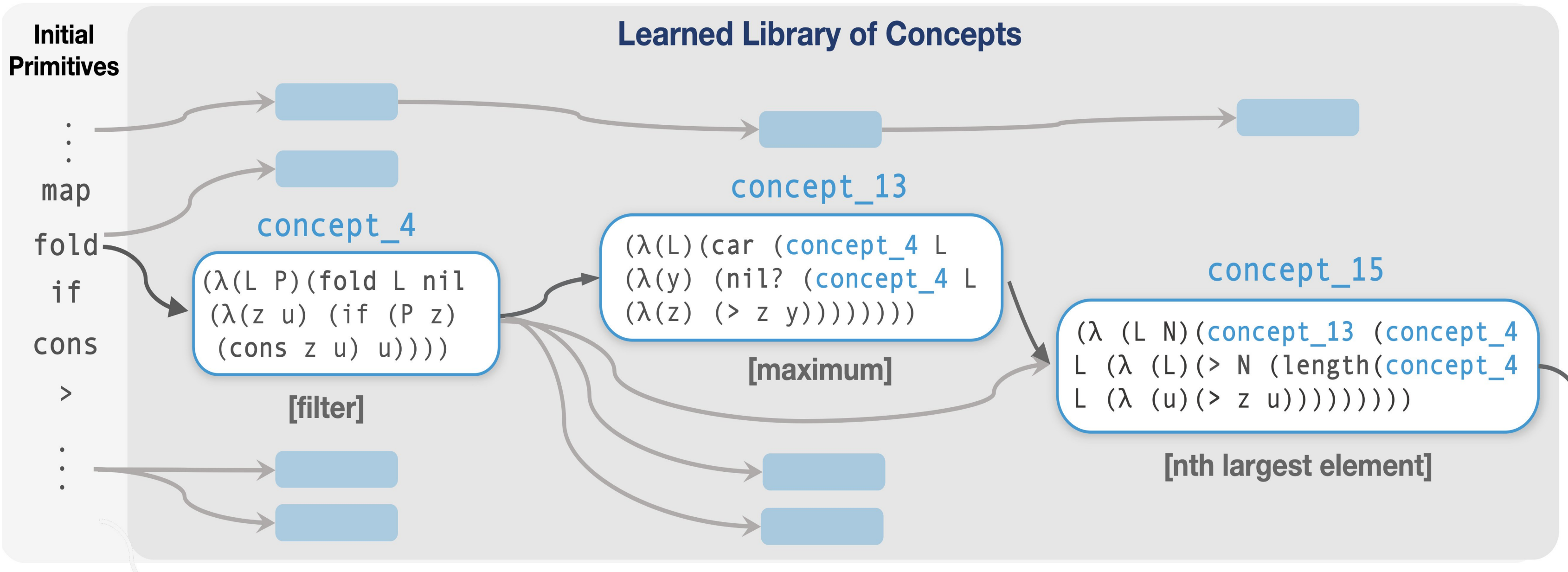
Sample Problem: `sort list`

[9 2 7 1] → [1 2 7 9]
[3 8 9 4 2] → [2 3 4 8 9]
[6 2 2 3 8 5] → [2 2 3 5 6 8]
...

Solution to `sort list` discovered in learned language:

```
(map  (λ (n)
  (concept_15 L (+ 1 n)))
  (range (length L)))
```

Library Learning



Sample Problem: `sort list`

[9 2 7 1] → [1 2 7 9]
[3 8 9 4 2]→ [2 3 4 8 9]
[6 2 2 3 8 5]→ [2 2 3 5 6 8]
...

Solution to `sort list` discovered in learned language:

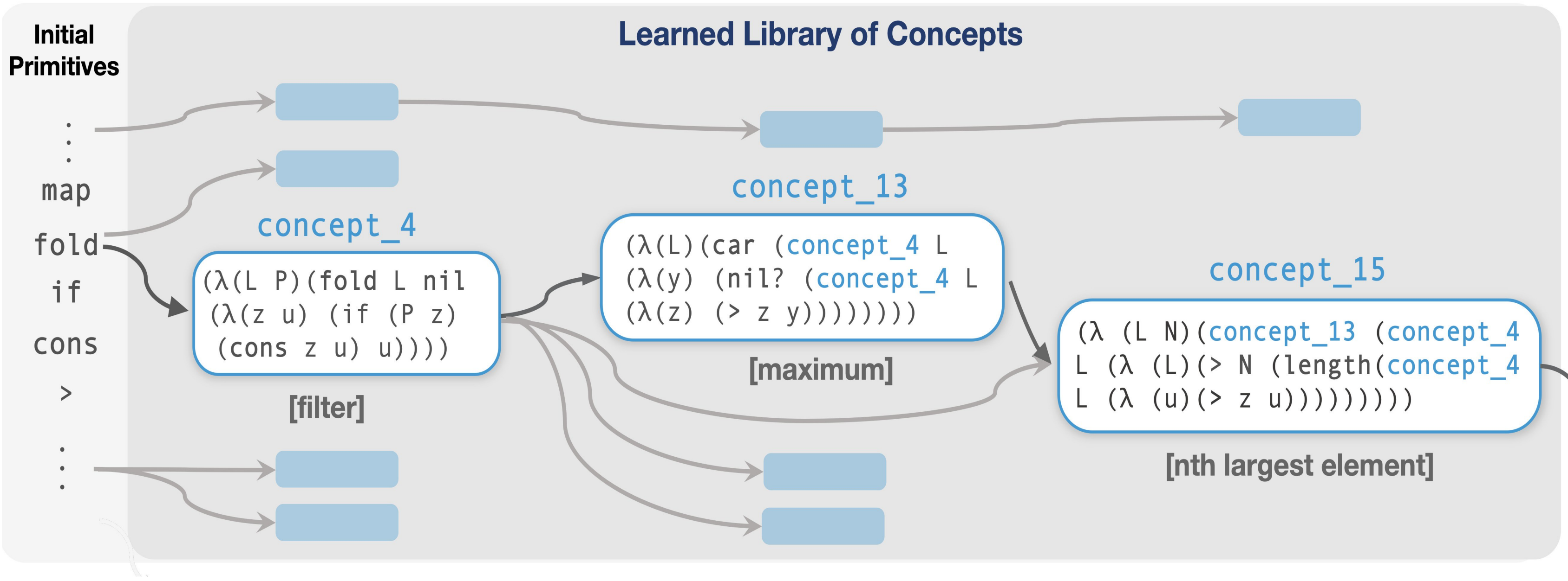
```
(map (λ (n)
  (concept_15 L (+ 1 n)))
  (range (length L)))
```

get Nth largest element, where N is 1, 2, 3, ...

Solution rewritten in initial primitives:

```
(lambda (x) (map (lambda (y) (car (fold (fold x nil (lambda (z u) (if (gt? (+ y 1) (length (fold x
nil (lambda (v w) (if (gt? z v) (cons v w) w))))) (cons z u) u))) nil (lambda (a b) (if (nil?
(fold (fold x nil (lambda (c d) (if (gt? (+ y 1) (length (fold x nil (lambda (e f) (if (gt? c e)
(cons e f) f))))) (cons c d) d))) nil (lambda (g h) (if (gt? g a) (cons g h) h)))) (cons a b)
b))))) (range (length x))))
```

Library Learning



Sample Problem: `sort list`

[9 2 7 1] → [1 2 7 9]
[3 8 9 4 2] → [2 3 4 8 9]
[6 2 2 3 8 5] → [2 2 3 5 6 8]
...

Solution to `sort list` discovered in learned language:

```
(map (λ (n)
  (concept_15 L (+ 1 n)))
  (range (length L)))
```

get Nth largest element, where N is 1, 2, 3, ...

- Induced sort program found in ≤ 10min.
- Brute-force search without learned library would take ≈ 10^{73} years

Dreamcoder

- **Wake:** Solve problems by writing programs
- **Sleep:** Improve library and neural recognition model:
    - **Abstraction sleep:** Improve library
    - **Dream sleep:** Improve neural recognition model

Dreamcoder

List Processing

Sum List

[1 2 3] → 6

[4 6 8 1] → 17

Double

[1 2 3] → [2 4 6]

[4 5 1] → [8 10 2]

Text Editing

Abbreviate

Allen Newell → A.N.

Herb Simon → H.S.

Drop Last Three

shrdlu → shr

shakey → sha

Regexes

Phone numbers

(555) 867-5309

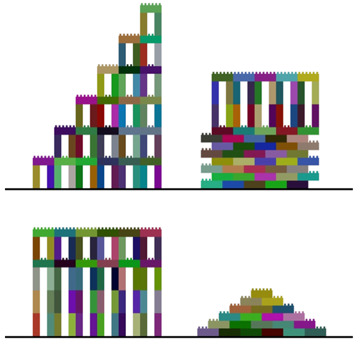
(650) 555-2368

Currency

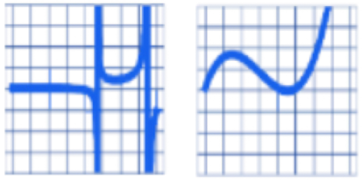
$100.25

$4.50

Block Towers



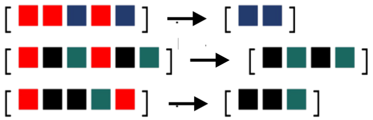
Symbolic Regression



$y = f(x)$

Recursive Programming

Filter Red

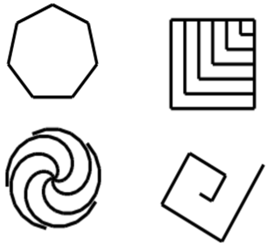


Physical Laws

$$\vec{a} = \frac{1}{m}\sum_i \vec{F}_i$$

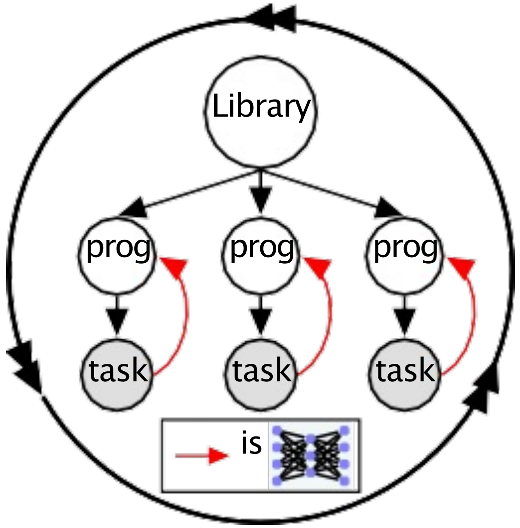
$$\vec{F} \propto \frac{q_1 q_2}{|\vec{r}|^2}\hat{r}$$

LOGO Graphics

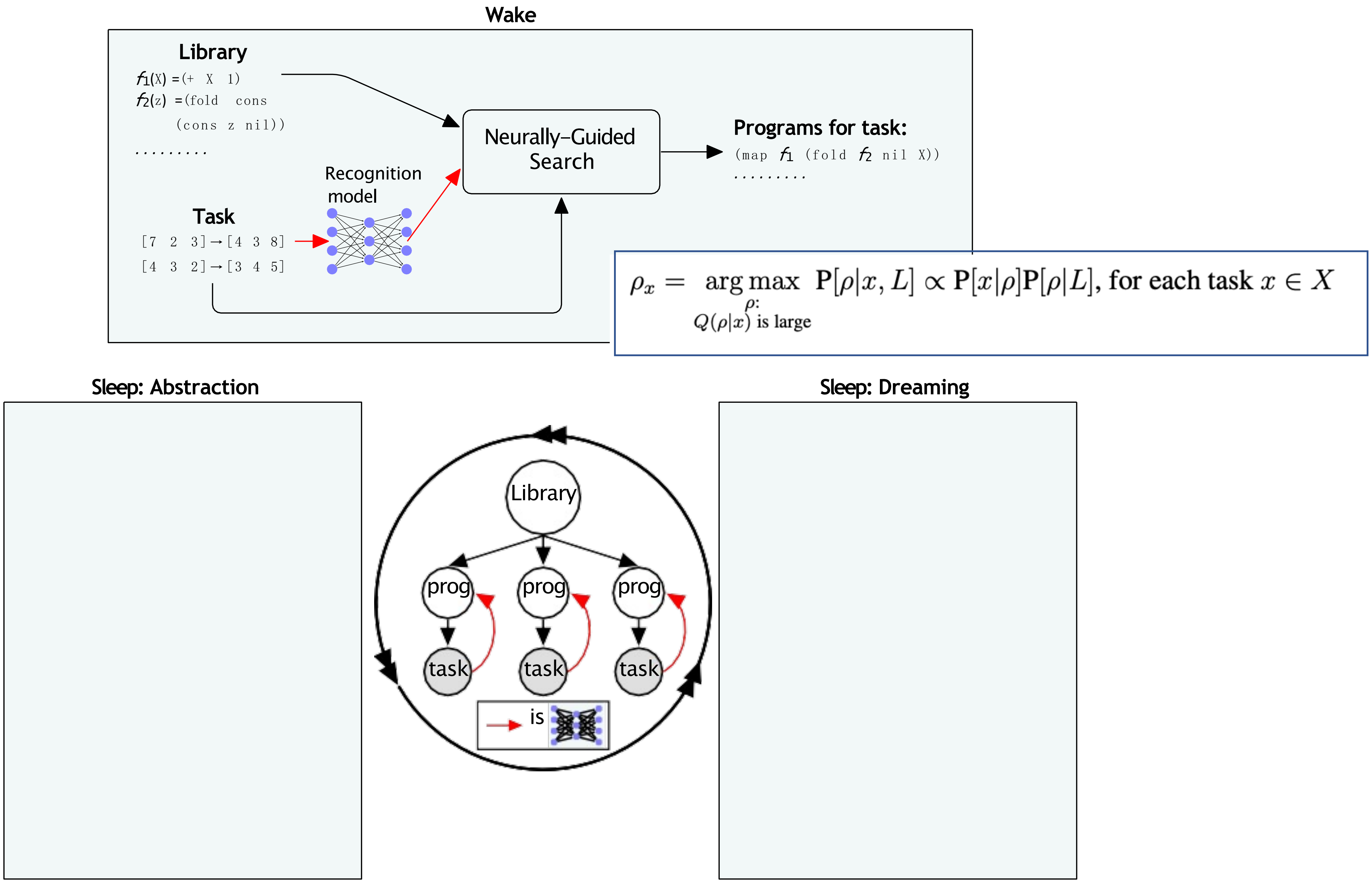


Wake

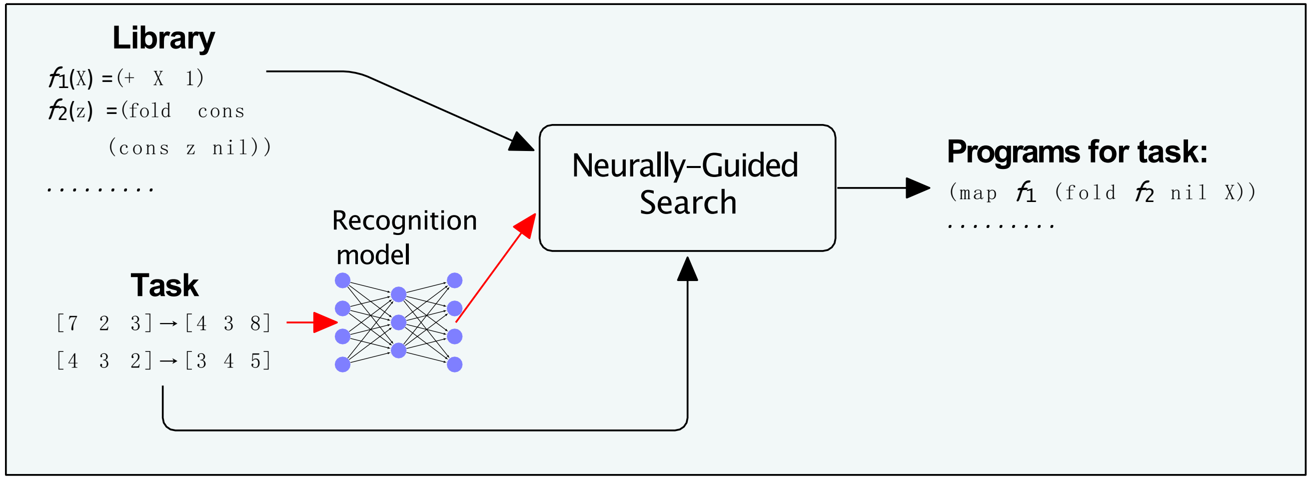
Sleep: Abstraction



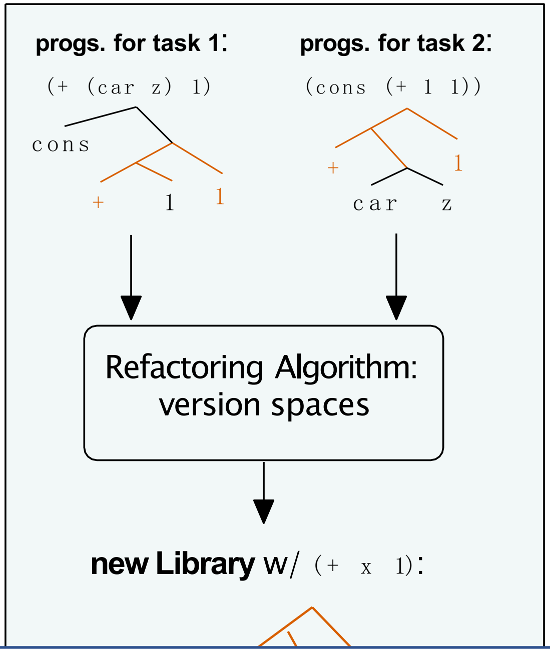
Sleep: Dreaming

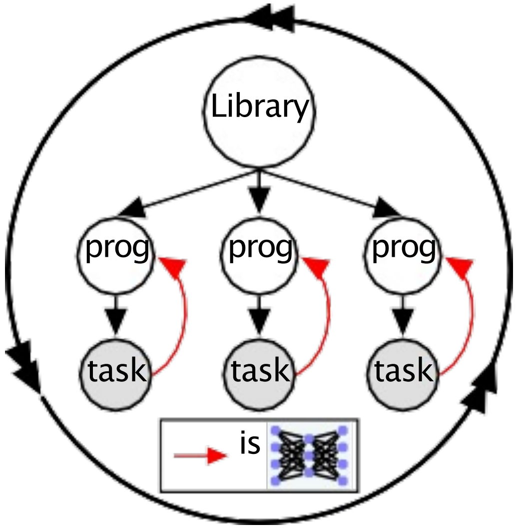

Wake
Library
f1(X) =(+ X 1)
f2(z) =(fold cons (cons z nil))
....
Recognition model
Task
[7 2 3]→[4 3 8]
[4 3 2]→[3 4 5]
Neurally–Guided Search
Programs for task:
(map f1 (fold f2 nil X))
....
$\rho_x = \underset{\substack{\rho: \\ Q(\rho|x) \text{ is large}}}{\arg\max}\ \mathrm{P}[\rho|x, L] \propto \mathrm{P}[x|\rho]\mathrm{P}[\rho|L]$, for each task $x \in X$
Sleep: Abstraction
Sleep: Dreaming
Library
prog
prog
prog
task
task
task
is

Wake

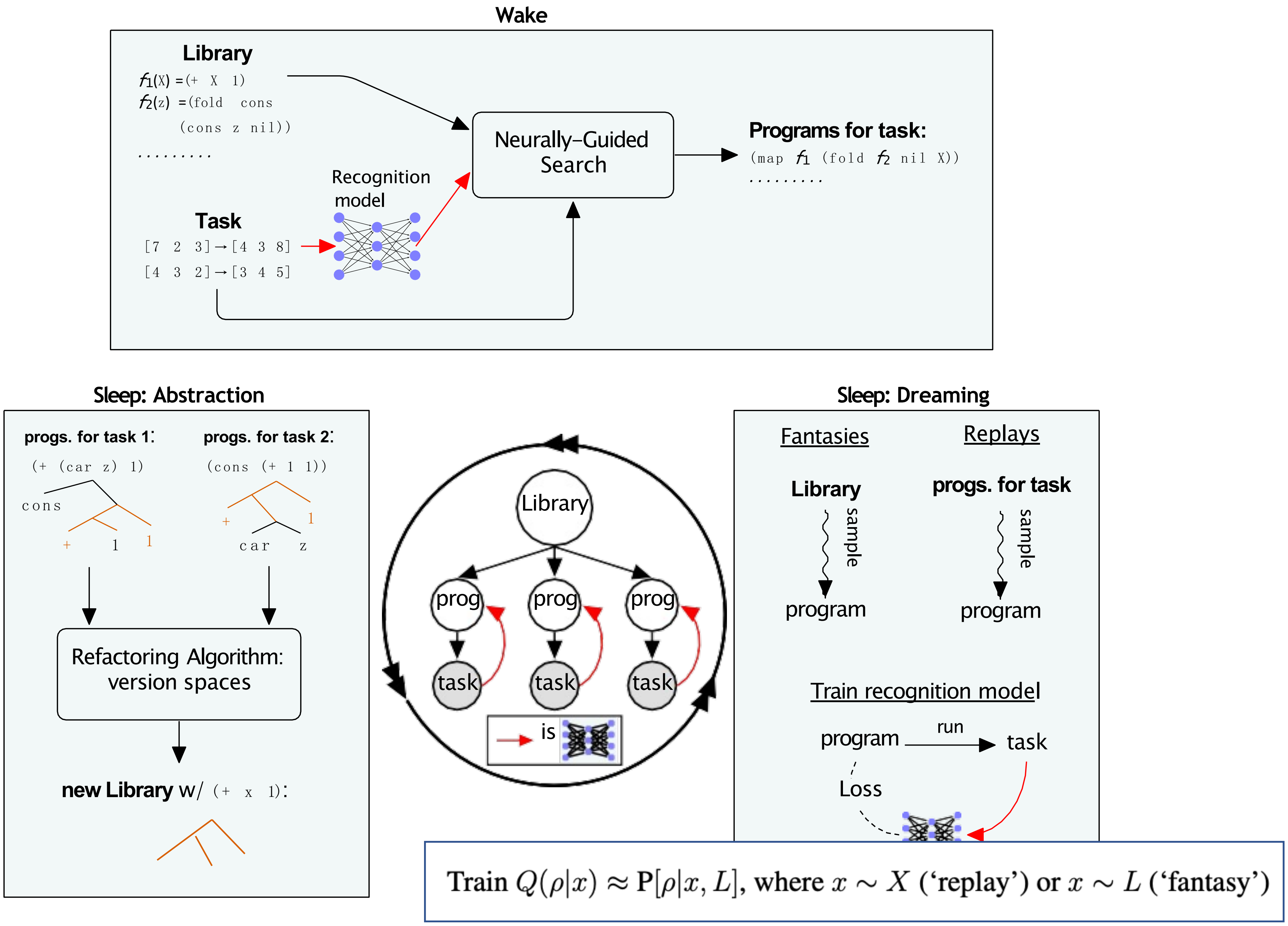


Sleep: Abstraction





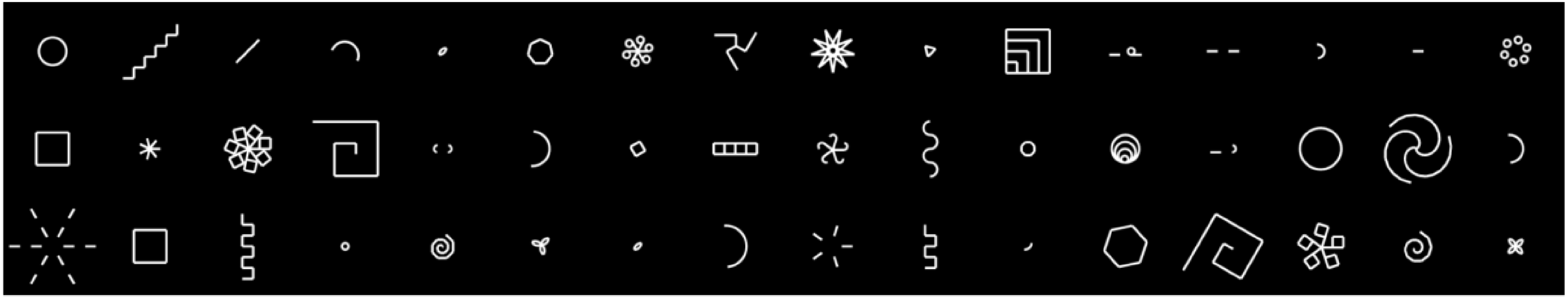
$$L = \arg\max_{L} \mathbf{P}\left[L\right] \prod_{x \in X} \max_{\rho \text{ a refactoring of } \rho_x} \mathbf{P}[x|\rho]\mathbf{P}[\rho|L]$$



Wake
Library
f1(X) = (+ X 1)
f2(z) = (fold cons (cons z nil))
....
Recognition model
Task
[7 2 3] → [4 3 8]
[4 3 2] → [3 4 5]
Neurally–Guided Search
Programs for task:
(map f1 (fold f2 nil X))
....
Sleep: Abstraction
progs. for task 1:
(+ (car z) 1)
cons
+
1
1
progs. for task 2:
(cons (+ 1 1))
+
1
car
z
Refactoring Algorithm: version spaces
new Library w/ (+ x 1):
Library
prog
prog
prog
task
task
task
is
Sleep: Dreaming
Fantasies
Replays
Library
sample
program
progs. for task
sample
program
Train recognition model
program
run
task
Loss
Train Q(ρ|x) ≈ P[ρ|x, L], where x ∼ X ('replay') or x ∼ L ('fantasy')

Example: LOGO Graphics

Input: Corpus of target shapes that we would like to learn how to draw



Input: Basic drawing language

move , for, $*$,+, π, pen-up,…

Learned subroutines

Parametric drawing routines in library	Higher-order drawing routine in library

Semicircle:

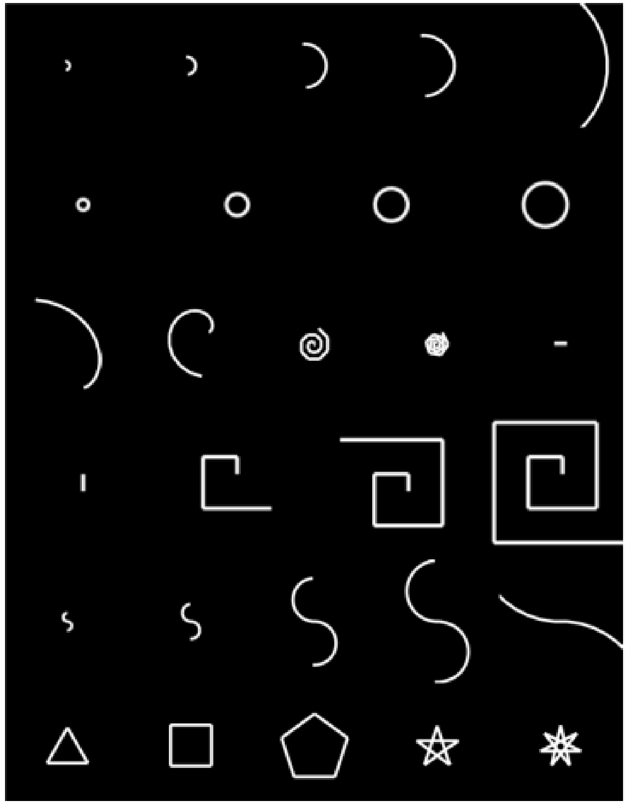
Circles:

Spiral:

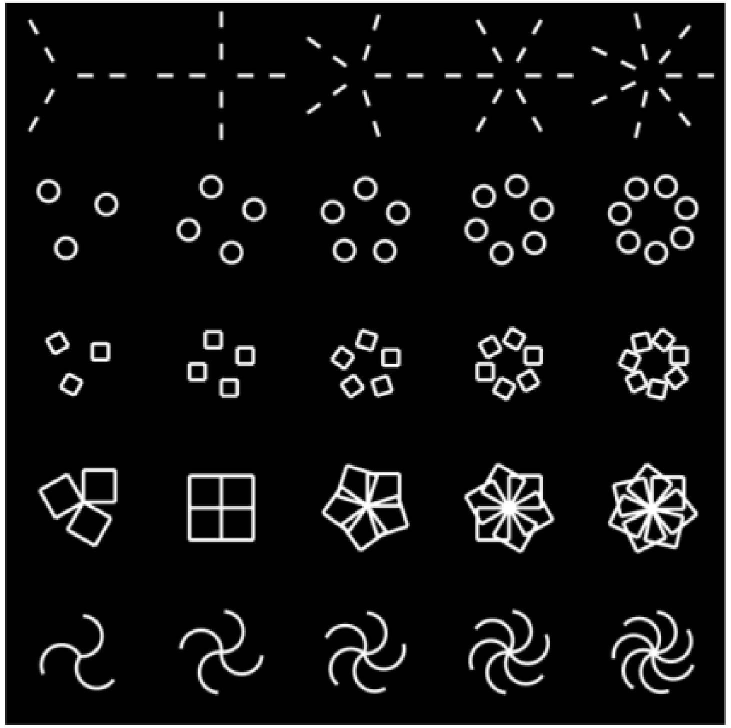
Greek Spiral:

S-Curves:

Polygons & Stars:

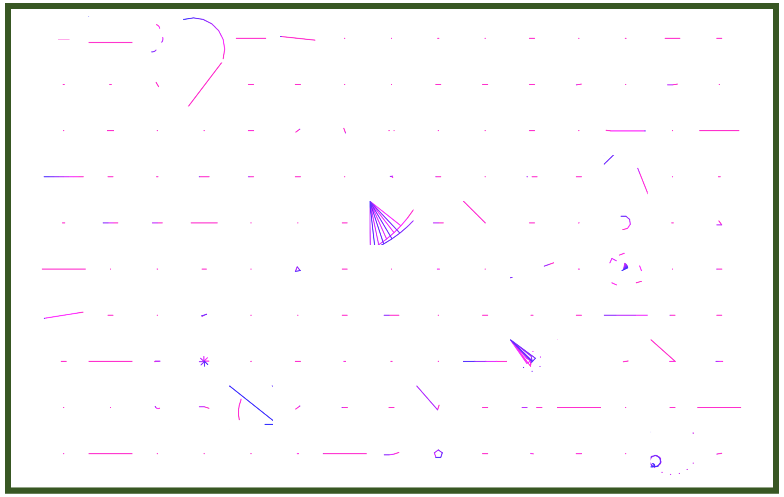


Radial Symmetry:

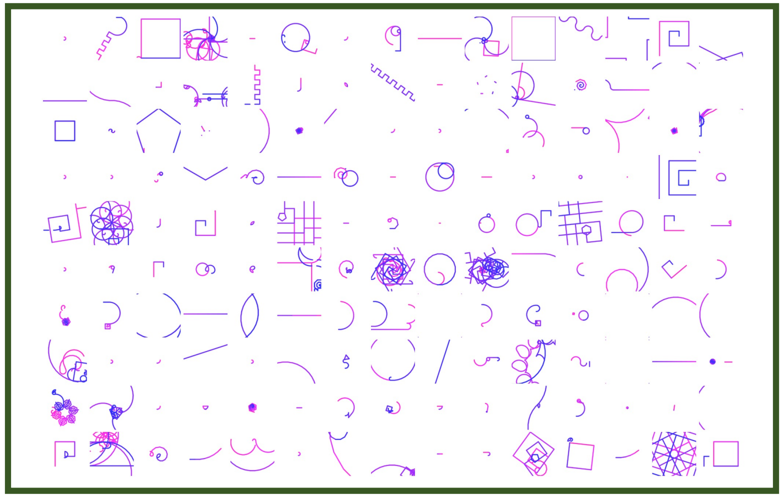


Language helps generation

The model is trained by sampling from the learned language
The language provides an inductive bias for generation

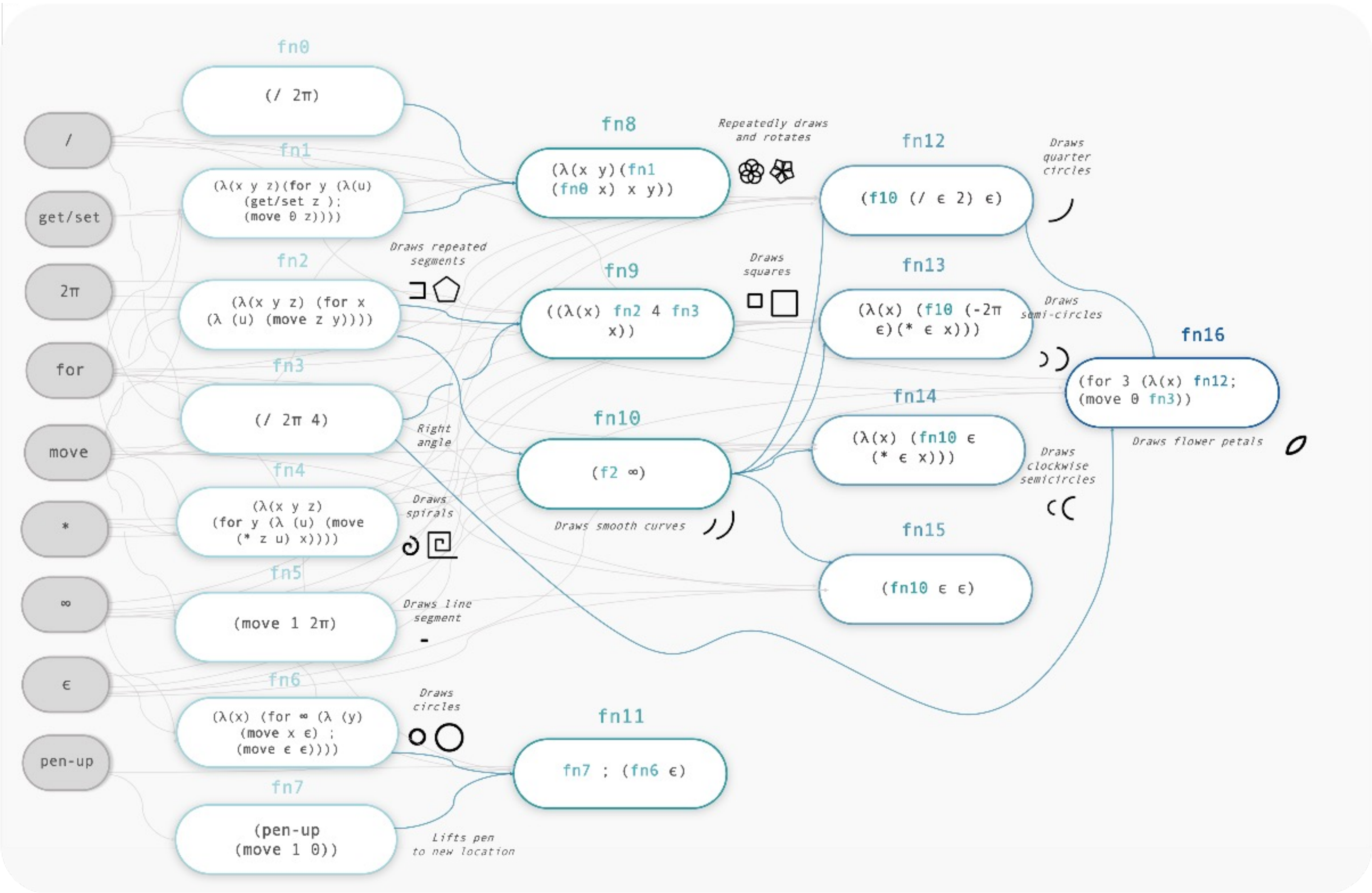


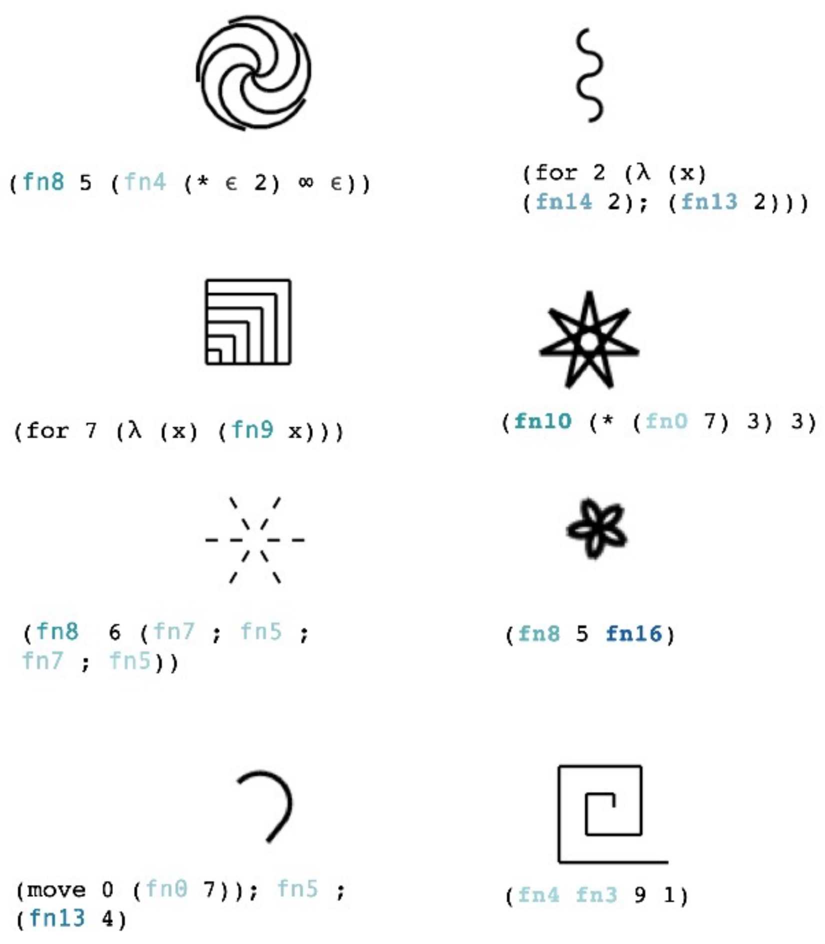
Initially dreams are very unstructured



A richer language leads to more structured dreams

Learning the language

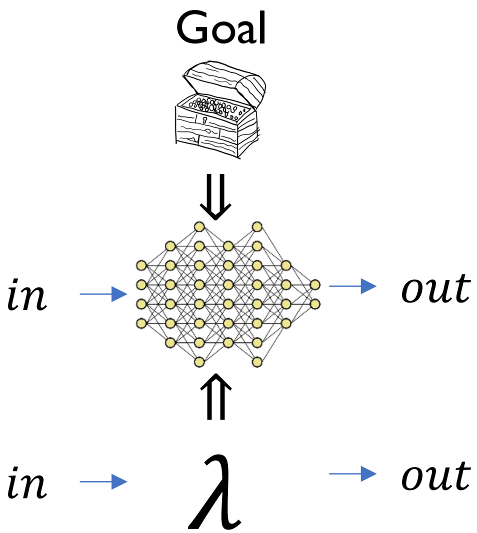


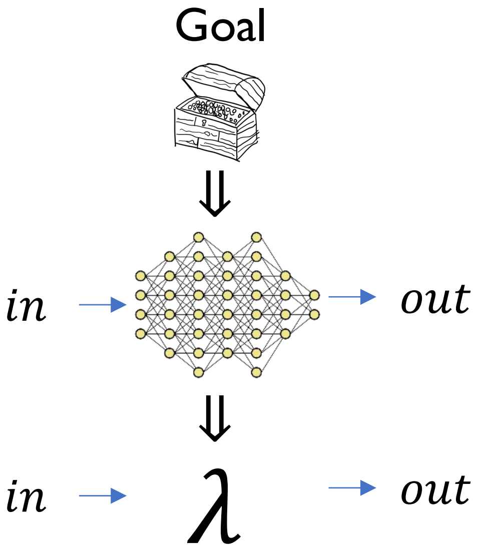


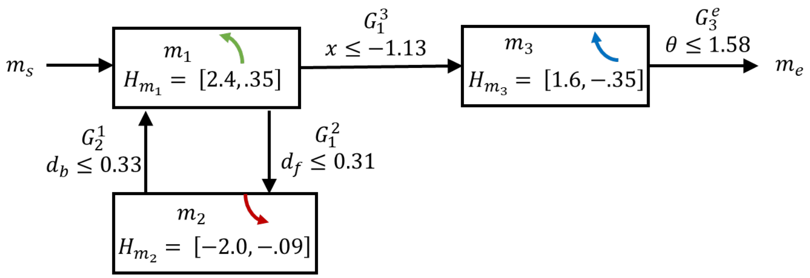
Other Directions

Combining algorithmic building blocks

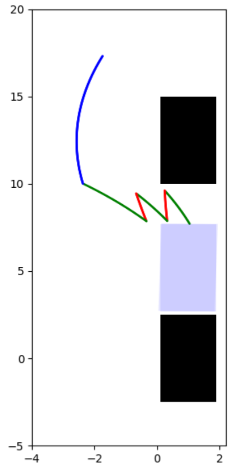
Example:

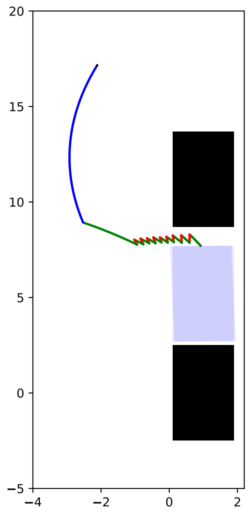




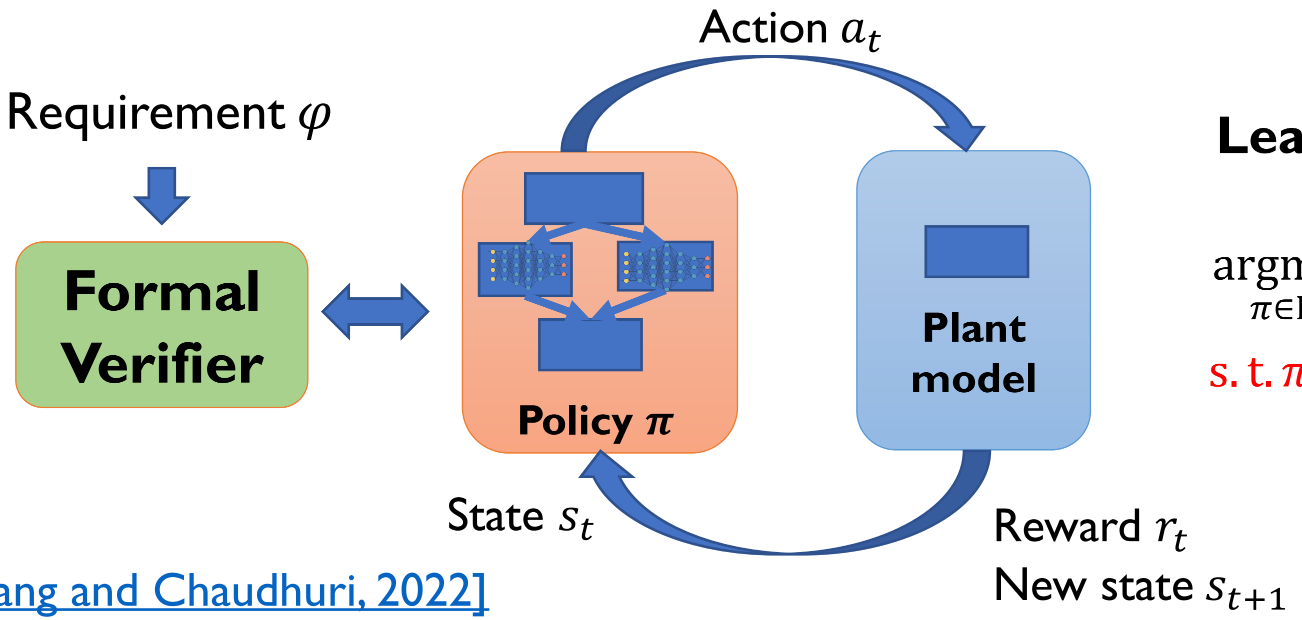


[Inala et. al. ICLR 2020]





Ensuring Correctness

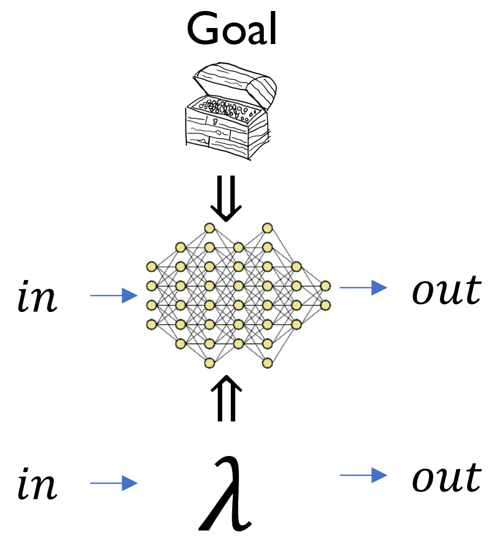


[Yang and Chaudhuri, 2022]

Learning goal:

$$\underset{\pi \in \Pi}{\operatorname{argmax}}\ \mathbb{E}_{\tau \sim \pi}\left[\sum_{t}^{\infty} \gamma^{t}\, r_t\right]$$

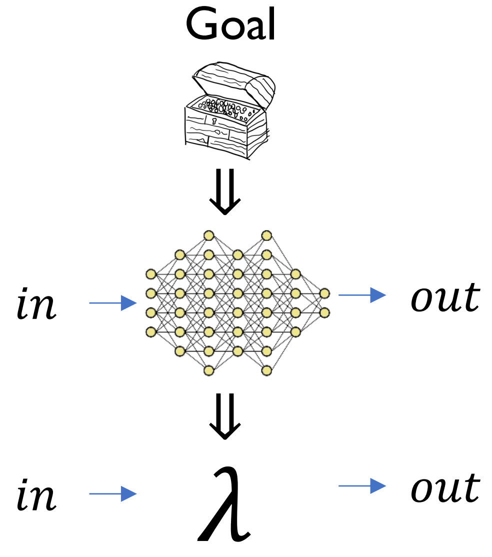
$\mathrm{s.t.}\ \pi \vDash \varphi$



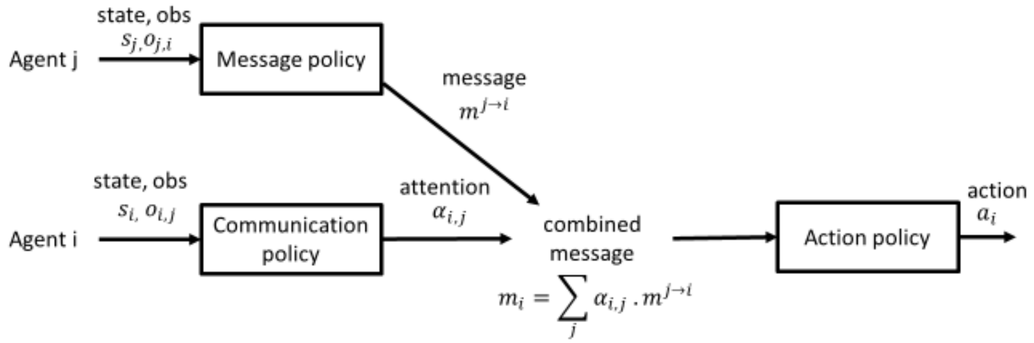
- Differentiable loss quantifying the extent to which the policy satisfies the requirement
- Constructed by calls to a formal verifier from within the learning loop
- Gradients of this loss used to guide learning

Interpretability

[Inala et al. Neurips 2020]

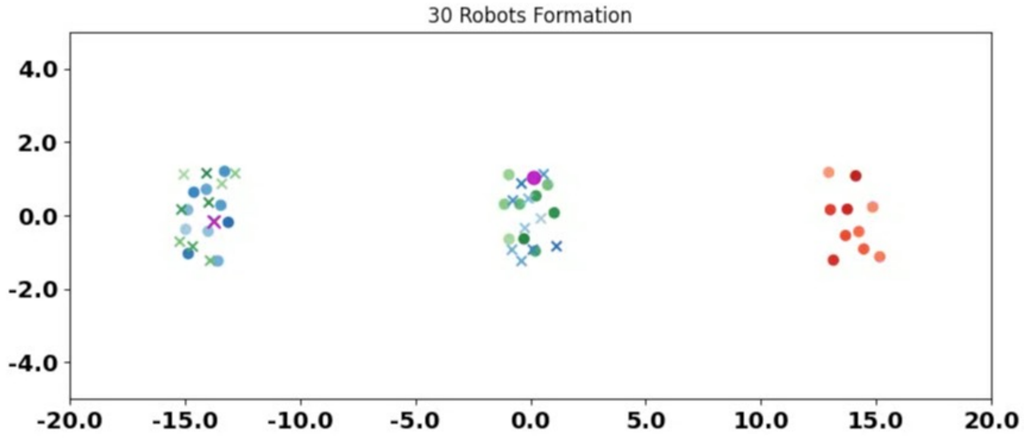


Attention based decentralized policy



Communication constraint: minimize $\underbrace{\max_i \sum_j \mathbf{1}[\alpha_{i,j} > 0]}_{\text{Maximum incoming degree}} + \underbrace{\max_i \sum_j \mathbf{1}[\alpha_{j,i} > 0]}_{\text{Maximum outgoing degree}}$

Rule-based policy



Rule1 :: random (filter (agents in $\boxed{}_{\theta}$))

Neurosymbolic Programs

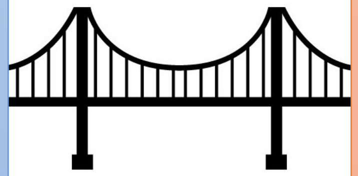
Symbolic Programs

Interpretable

Verifiable

Structured domain knowledge

Data efficient



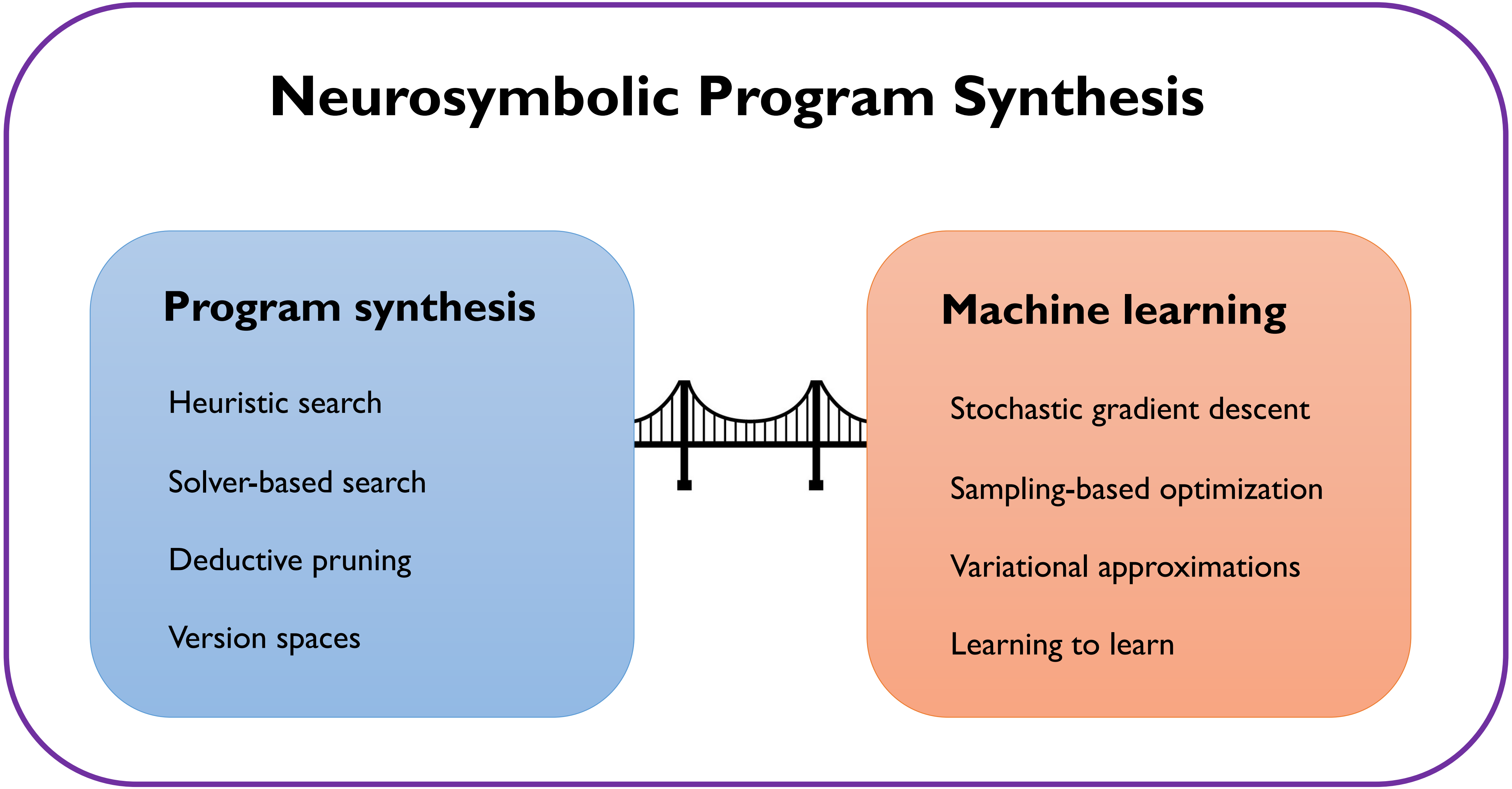
Neural Networks

Scalable algorithms

Flexible

Handles messy data

Easy to get started

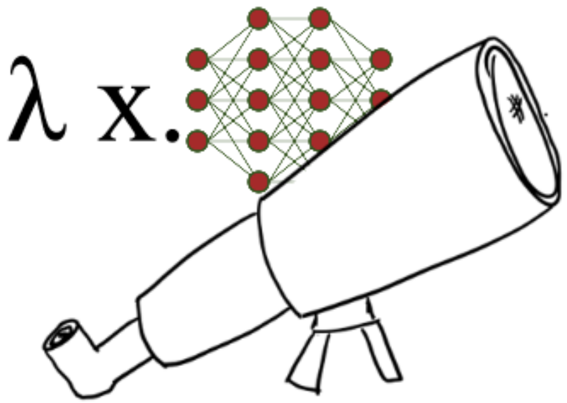

Neurosymbolic Program Synthesis
Program synthesis
Heuristic search
Solver-based search
Deductive pruning
Version spaces
Machine learning
Stochastic gradient descent
Sampling-based optimization
Variational approximations
Learning to learn

Neurosymbolic learning isn't new…

…but it's a good time to push on it!

- Recent progress in symbolic reasoning and deep learning
- New algorithms that can scale
- Demand by domain experts

Understanding the World Through Code



An NSF funded Expeditions in Computing Project

neurosymbolic.org