Machine Learning Guidance for an Automatic Theorem Prover

Martin Suda*

Czech Technical University in Prague, Czech Republic

ML4SP Workshop, Glasgow, August 2025

*Supported by the Czech Science Foundation standard project 24-12759S.

Machine Learning for Solvers

Machine Learning for Solvers

Pick a Heuristic

- many don't-care non-deterministic choice points
- normally governed by hand-crafted heuristics

Machine Learning for Solvers

Pick a Heuristic

- many don't-care non-deterministic choice points
- normally governed by hand-crafted heuristics

Replace it with a neural network

Machine Learning for Solvers

Pick a Heuristic

- many don't-care non-deterministic choice points
- normally governed by hand-crafted heuristics

Replace it with a neural network

- give it access to the relevant context for making decisions

Machine Learning for Solvers

Pick a Heuristic

- many don't-care non-deterministic choice points
- normally governed by hand-crafted heuristics

Replace it with a neural network

- give it access to the relevant context for making decisions
- feature engineering / auto-learning of task-relevant node embeddings (GNNs, RvNNs,...)

Machine Learning for Solvers

Pick a Heuristic

- many don't-care non-deterministic choice points
- normally governed by hand-crafted heuristics

Replace it with a neural network

- give it access to the relevant context for making decisions
- feature engineering / auto-learning of task-relevant node embeddings (GNNs, RvNNs,...)
- faithful representations / easy-to-compute abstractions

Machine Learning for Solvers

Pick a Heuristic

- many don't-care non-deterministic choice points
- normally governed by hand-crafted heuristics

Replace it with a neural network

- give it access to the relevant context for making decisions
- feature engineering / auto-learning of task-relevant node embeddings (GNNs, RvNNs,...)
- faithful representations / easy-to-compute abstractions

Train the network from previous runs

Machine Learning for Solvers

Pick a Heuristic

- many don't-care non-deterministic choice points
- normally governed by hand-crafted heuristics

Replace it with a neural network

- give it access to the relevant context for making decisions
- feature engineering / auto-learning of task-relevant node embeddings (GNNs, RvNNs,...)
- faithful representations / easy-to-compute abstractions

Train the network from previous runs

- What lead to a success? / What was a reason for failure?

Machine Learning for Solvers

Pick a Heuristic

- many don't-care non-deterministic choice points
- normally governed by hand-crafted heuristics

Replace it with a neural network

- give it access to the relevant context for making decisions
- feature engineering / auto-learning of task-relevant node embeddings (GNNs, RvNNs,...)
- faithful representations / easy-to-compute abstractions

Train the network from previous runs

- What lead to a success? / What was a reason for failure?
- The proxy (ML) task vs the true target

Machine-Learning-Boosted Automated Theorem Proving

ATP technology: saturation-based

- state of the art (cf. CASC)
- E, iProver, SPASS, Vampire, ...

Machine-Learning-Boosted Automated Theorem Proving

ATP technology: saturation-based

- state of the art (cf. CASC)
- E, iProver, SPASS, Vampire, ...

Heuristic to boost: clause selection

- arguably the most important choice point
- "selecting just the proof clauses" intuition

Machine-Learning-Boosted Automated Theorem Proving

ATP technology: saturation-based

- state of the art (cf. CASC)
- E, iProver, SPASS, Vampire, ...

Heuristic to boost: clause selection

- arguably the most important choice point
- "selecting just the proof clauses" intuition

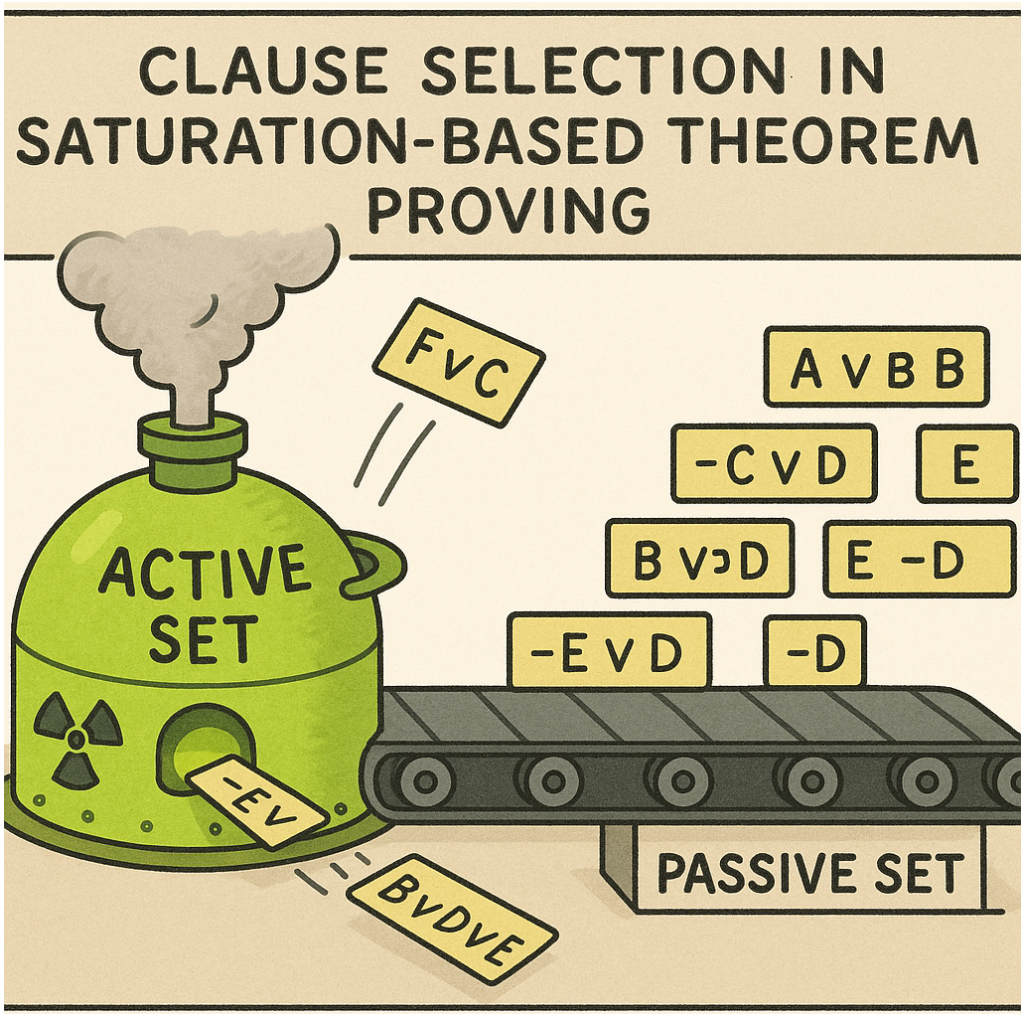
Three main contributions:

- a RL-inspired learning operator
- a new neural architecture (GNN + RvNNs + MLP)
- 20 % performance boost of Vampire under neural guidance

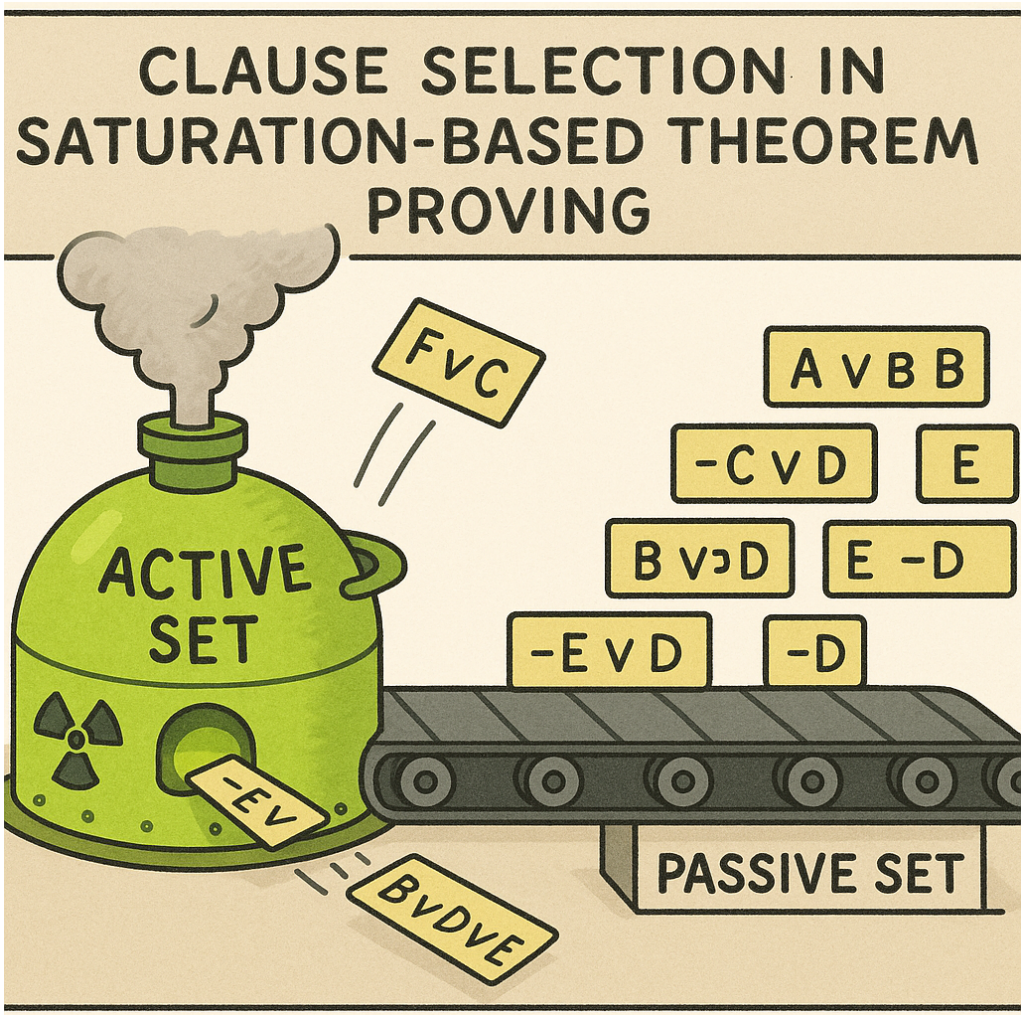
Outline

Outline

Saturation-based Theorem Proving

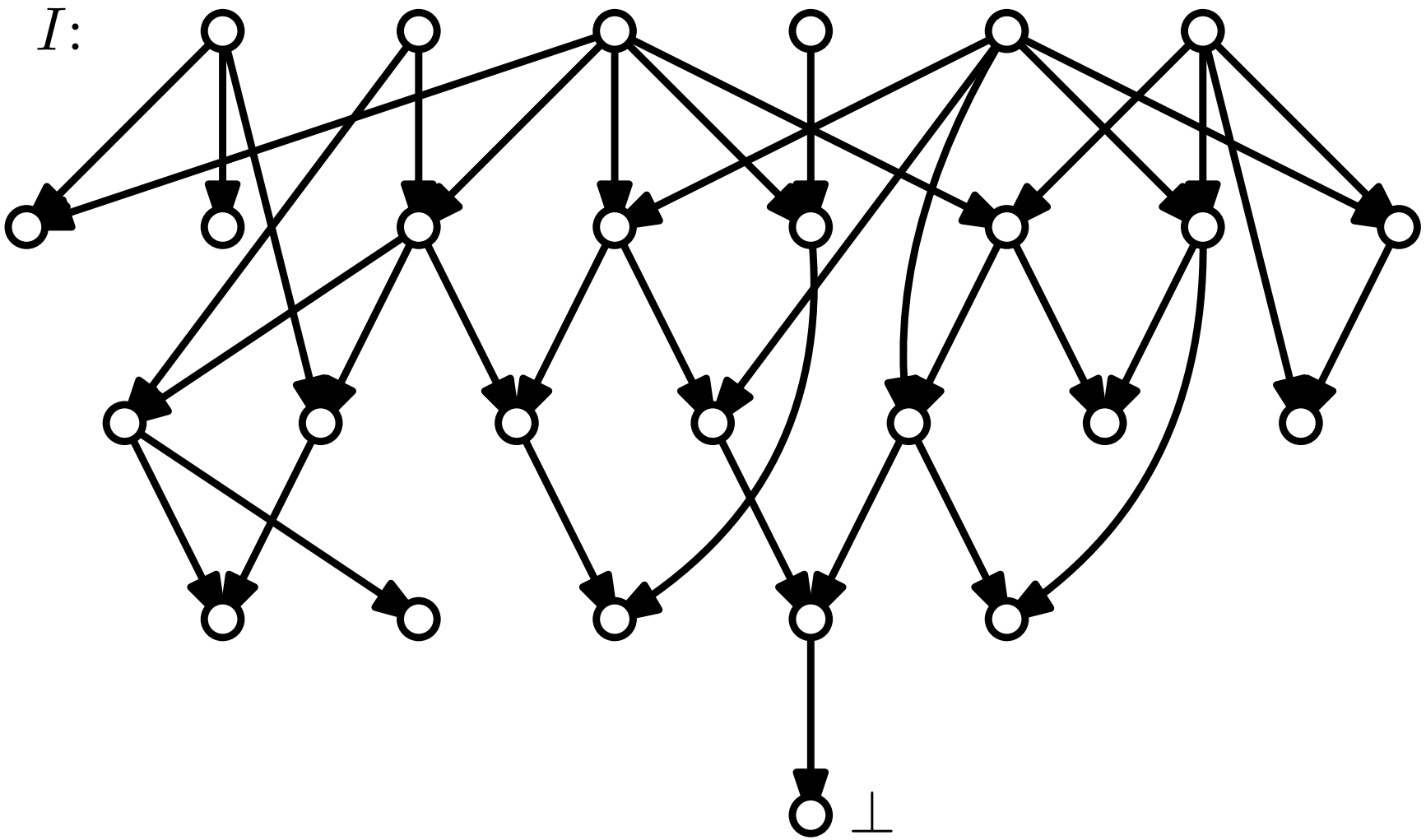


Saturation-based Theorem Proving

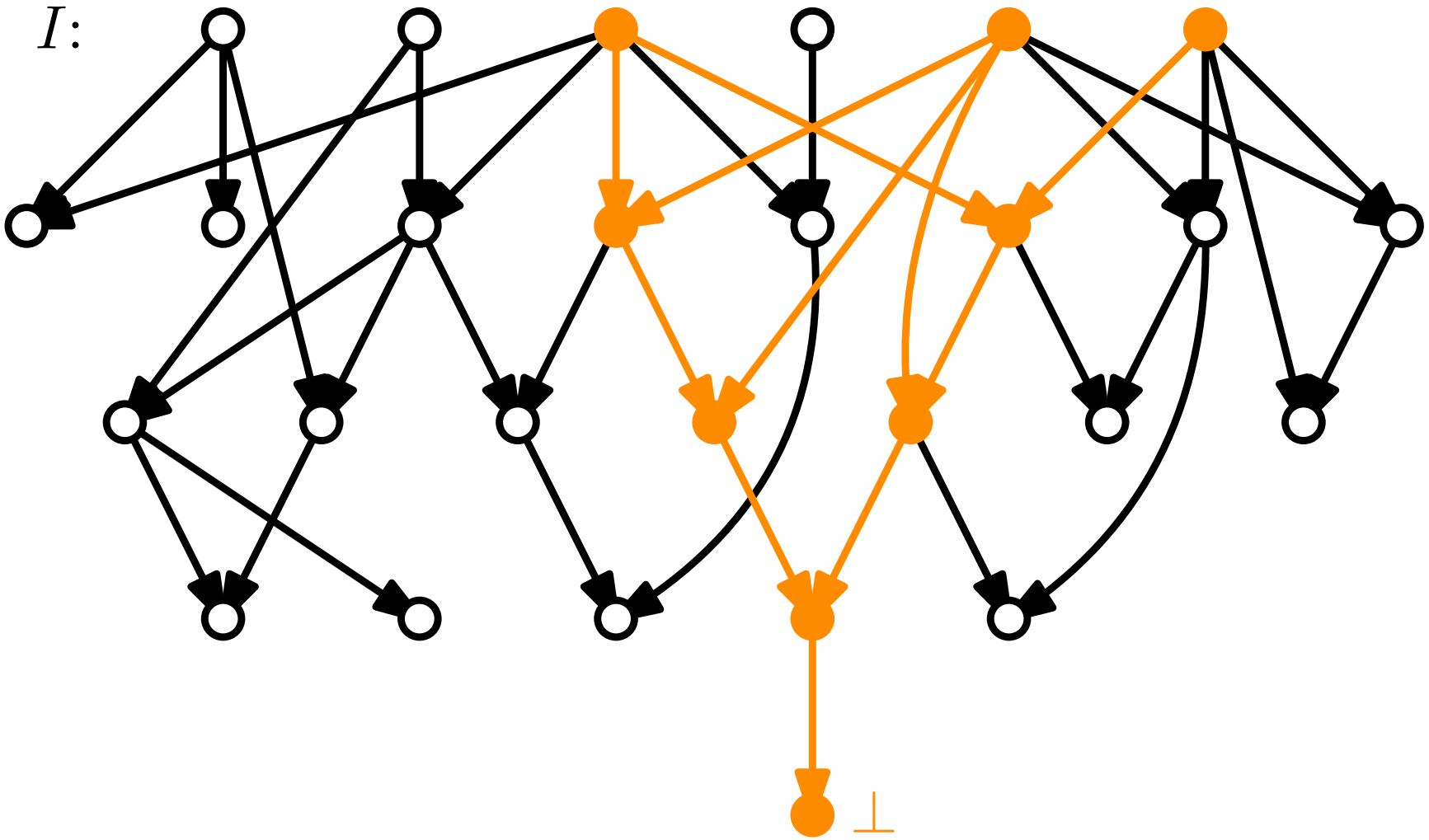


At a typical successful end: $|Passive| \gg |Active| \gg |Proof|$

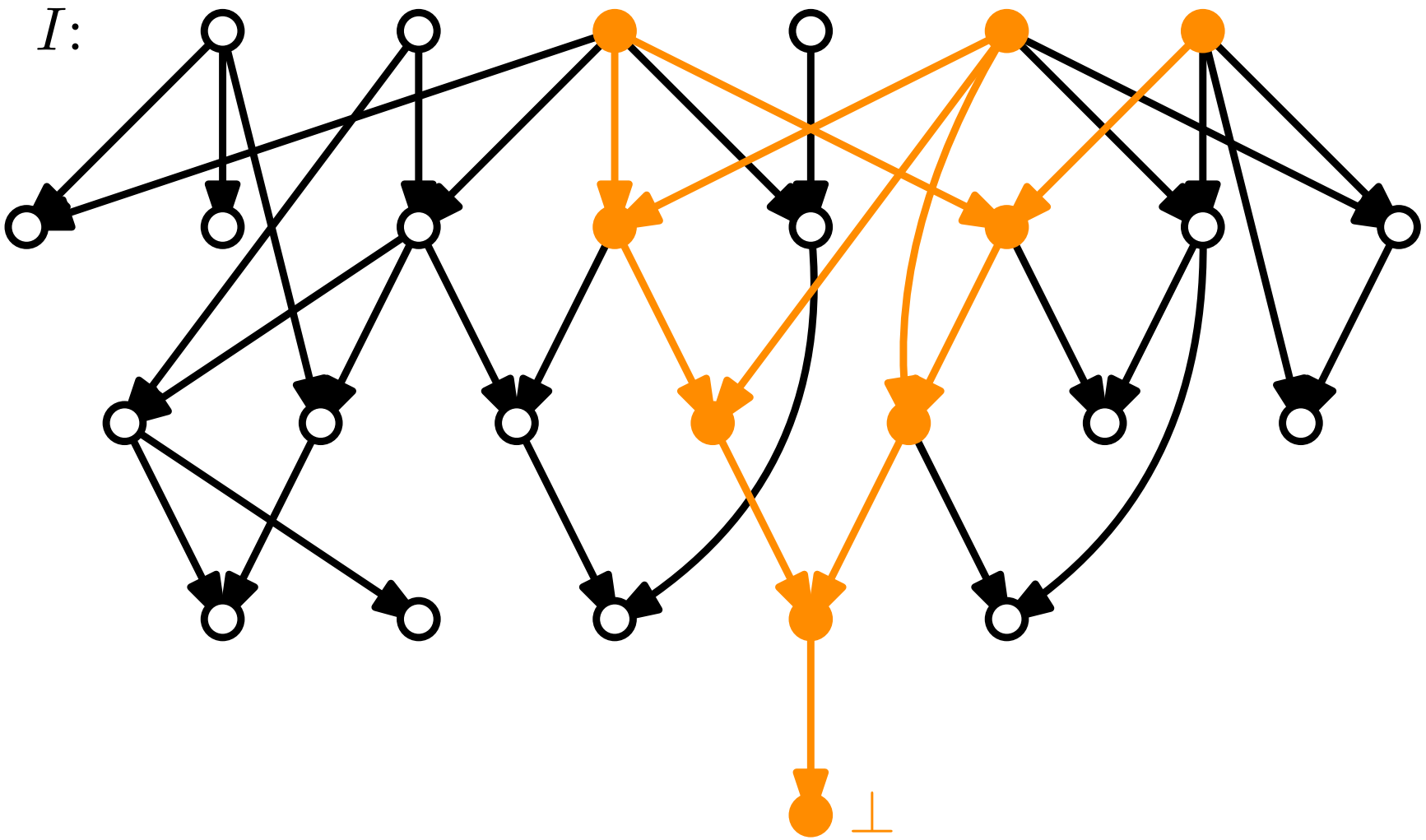
The Proof Is Often Just A Tiny Part



The Proof Is Often Just A Tiny Part



The Proof Is Often Just A Tiny Part



How close can we actually hope get to the perfect clause selection?

How is clause selection traditionally done?

Take simple clause evaluation criteria:

- age: prefer clauses that were generated long time ago
- weight: prefer clauses with fewer symbols

How is clause selection traditionally done?

Take simple clause evaluation criteria:

- age: prefer clauses that were generated long time ago
- weight: prefer clauses with fewer symbols

Combine them into a single scheme:

- have a priority queue ordering *Passive* for each criterion
- alternate between selecting from the queues using a fixed ratio

How is clause selection traditionally done?

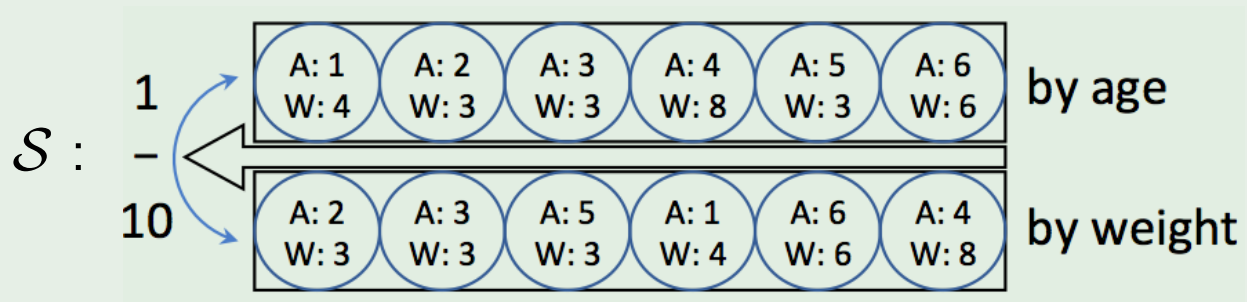
Take simple clause evaluation criteria:

- age: prefer clauses that were generated long time ago
- weight: prefer clauses with fewer symbols

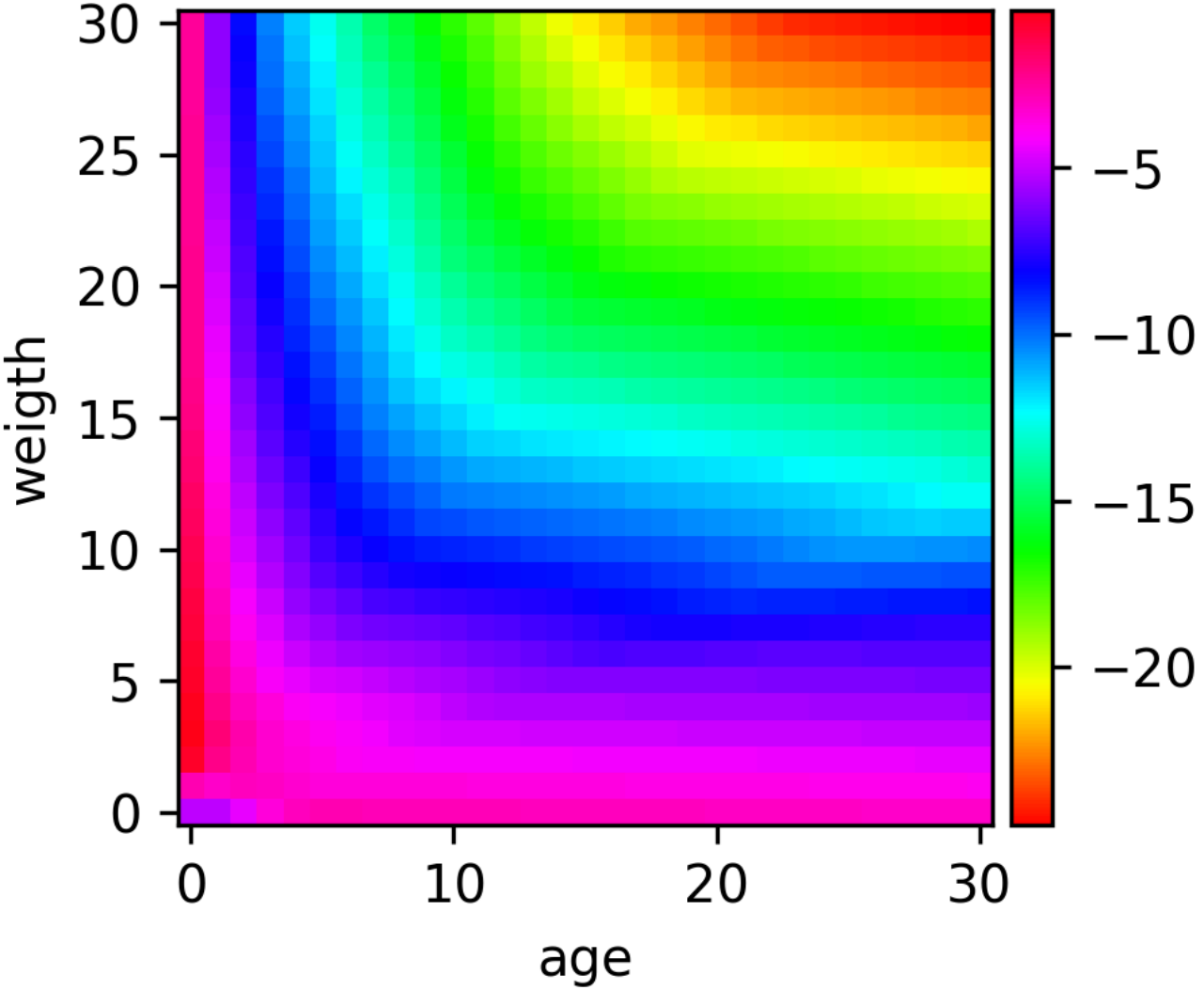
Combine them into a single scheme:

- have a priority queue ordering *Passive* for each criterion
- alternate between selecting from the queues using a fixed ratio

Example (Organizing *Passive* via two priority queues)



Sneak Peek: What Do NNs Think of Age and Weigth?



Outline

Why Reinforcement Learning?

Why Reinforcement Learning?

Inspired by the great successes:

- ATARI games (DQN)
  V. Mnih et al. Playing ATARI with deep reinforcement learning. CoRR, 2013.
- Board games (AlphaZero)
  D. Silver et al. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. Science, 2018.
- . . .
- "I wan't to try it on my pet problem too!"

Why Reinforcement Learning?

Inspired by the great successes:

- ATARI games (DQN)
  V. Mnih et al. Playing ATARI with deep reinforcement learning. CoRR, 2013.
- Board games (AlphaZero)
  D. Silver et al. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. Science, 2018.
- ...
- "I wan't to try it on my pet problem too!"

What's really unique about RL?

- It programs itself (sometimes even optimally, in the limit)
- It could discover fundamentally novel tricks and hacks!

Key Reinforcement Learning Concepts

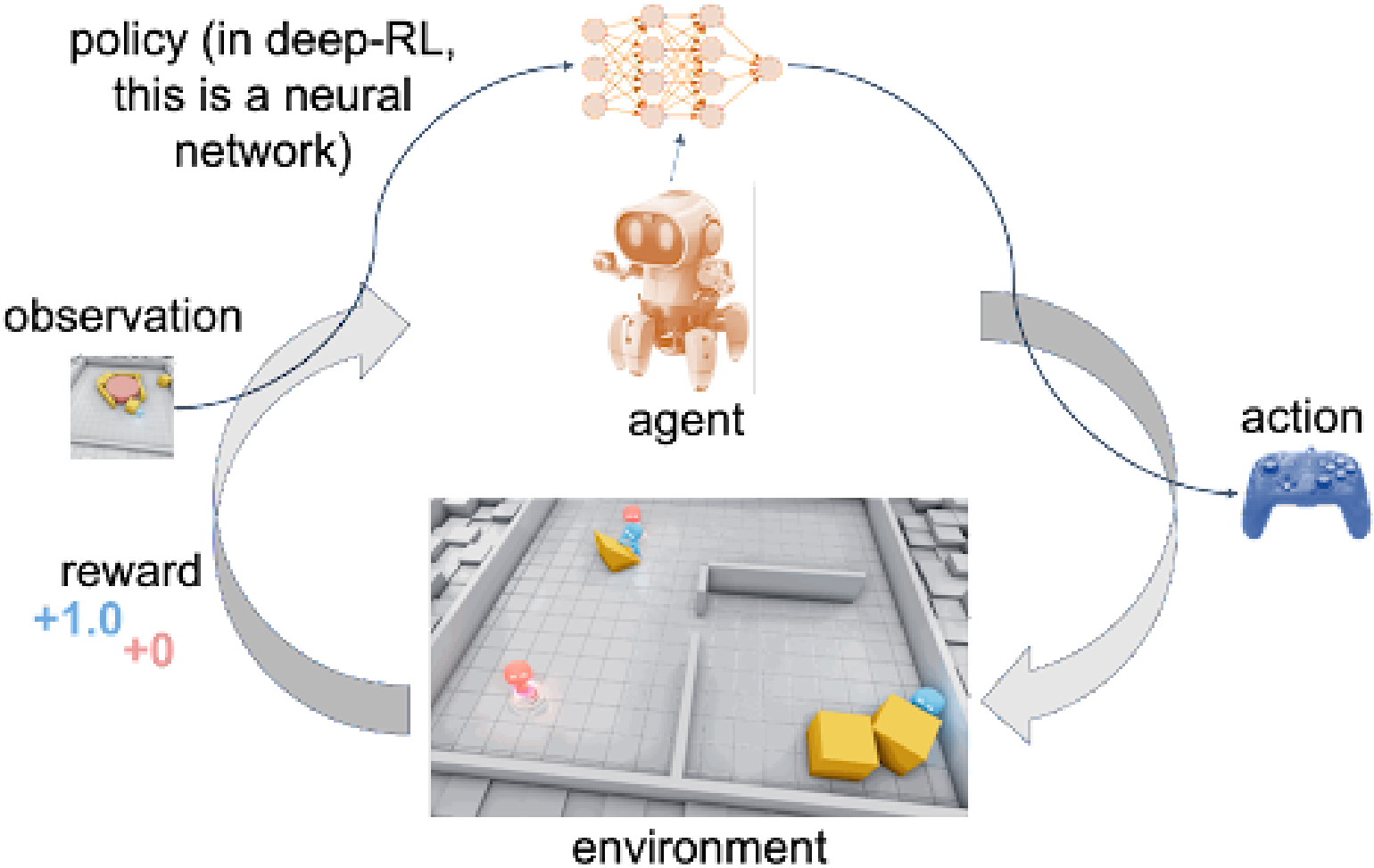


*Illustration from anyscale.com.

Saturation as an Reinforcement-Learning Environment

Agent

- the clause selection heuristic

Saturation as an Reinforcement-Learning Environment

Agent

- the clause selection heuristic

Action

- the next clause to select from the current passive set

Saturation as an Reinforcement-Learning Environment

Agent

- the clause selection heuristic

Action

- the next clause to select from the current passive set

State / Observation

- static - the conjecture we are trying to prove
- evolving - the internal state of the prover at particular moment

Saturation as an Reinforcement-Learning Environment

Agent

- the clause selection heuristic

Action

- the next clause to select from the current passive set

State / Observation

- static - the conjecture we are trying to prove
- evolving - the internal state of the prover at particular moment

Reward

- Score 1 point for solving a problem (within the time limit) ???

Saturation as an Reinforcement-Learning Environment

Agent

- the clause selection heuristic

Action

- the next clause to select from the current passive set

State / Observation

- static - the conjecture we are trying to prove
- evolving - the internal state of the prover at particular moment

Reward

- Score 1 point for solving a problem (within the time limit) ???

➥ TRAIL [Crouse et al.'21], [McKeown'23], [Shminke'23], ...

Design Decisions

Guiding Principle

The new design accommodates the old heuristic as an attainable point in the space of possible solutions.

Design Decisions

Guiding Principle

The new design accommodates the old heuristic as an attainable point in the space of possible solutions.

State / Observation

- the evolving state of an ATP is a large amorphous blob
- there is no state in the SoTA clause-selection heuristics
- let's discard state too $\Rightarrow$ assumption of state-less environment

Design Decisions

Guiding Principle

The new design accommodates the old heuristic as an attainable point in the space of possible solutions.

State / Observation

- the evolving state of an ATP is a large amorphous blob
- there is no state in the SoTA clause-selection heuristics
- let's discard state too $\Rightarrow$ assumption of state-less environment

Reward

- refusing the play the honest, super-sparse reward game
- like in ENIGMA: a proof clause is a good clause

Towards the RL-Inspired Learning Operator

A trace of a successful proof attempt on problem P is a tuple

$$T = (P, \mathcal{C}, \mathcal{C}^+, \{\mathcal{P}_i\}_{i \in I_T}).$$

Towards the RL-Inspired Learning Operator

A trace of a successful proof attempt on problem P is a tuple

$$T = (P, \mathcal{C}, \mathcal{C}^+, \{\mathcal{P}_i\}_{i \in I_T}).$$

Learning operator (for clause selection)

- input: neural network N_θ (learnable params θ), set of traces $\mathcal{T}$
- output: updated parameters θ',
  such that $N_{\theta'}$ is better at solving problems like those from $\mathcal{T}$

Towards the RL-Inspired Learning Operator

A trace of a successful proof attempt on problem P is a tuple

$$T = (P, \mathcal{C}, \mathcal{C}^+, \{\mathcal{P}_i\}_{i \in I_T}).$$

Learning operator (for clause selection)

- input: neural network $N_{\boldsymbol{\theta}}$ (learnable params $\boldsymbol{\theta}$), set of traces $\mathcal{T}$
- output: updated parameters $\boldsymbol{\theta}'$,
  such that $N_{\boldsymbol{\theta}'}$ is better at solving problems like those from $\mathcal{T}$

Logits and Policy

Assuming $N_{\boldsymbol{\theta}}$ produces a score $N_{\boldsymbol{\theta}}(C) = l_C$ for each clause C, then

Towards the RL-Inspired Learning Operator

A trace of a successful proof attempt on problem P is a tuple

$$T = (P, \mathcal{C}, \mathcal{C}^+, \{\mathcal{P}_i\}_{i \in I_T}).$$

Learning operator (for clause selection)

- input: neural network $N_{\boldsymbol{\theta}}$ (learnable params $\boldsymbol{\theta}$), set of traces $\mathcal{T}$
- output: updated parameters $\boldsymbol{\theta}'$,
  such that $N_{\boldsymbol{\theta}'}$ is better at solving problems like those from $\mathcal{T}$

Logits and Policy

Assuming $N_{\boldsymbol{\theta}}$ produces a score $N_{\boldsymbol{\theta}}(C) = l_C$ for each clause C, then

$$\pi_{C,\boldsymbol{\theta}} = \mathrm{softmax}_C\left(\{l_D\}_{D \in \mathcal{P}}\right) = \frac{e^{l_C}}{\sum_{D \in \mathcal{P}} e^{l_D}}$$

is the (stochastic) clause selection policy defined by $N_{\boldsymbol{\theta}}$

The RL-Inspired Operator

Policy Gradient Theorem [Williams'92]

To improve a policy in terms of the expected return we update

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha r_C \nabla_{\boldsymbol{\theta}} \log \pi_{C,\boldsymbol{\theta}},$$

where r_C is the return / reward at the corresponding step.

The RL-Inspired Operator

Policy Gradient Theorem [Williams'92]

To improve a policy in terms of the expected return we update

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha r_C \nabla_{\boldsymbol{\theta}} \log \pi_{C,\boldsymbol{\theta}},$$

where r_C is the return / reward at the corresponding step.

Our Operator:
Each moment in time i is an independent opportunity to improve, with

$$\delta_i^T = \mathrm{mean}_{C \in \mathcal{P}_i^+} \nabla_{\boldsymbol{\theta}} \log \pi_{C,\boldsymbol{\theta}},$$

for a trace $T = (P, \mathcal{C}, \mathcal{C}^+, \{\mathcal{P}_i\}_{i \in I_T})$ and $\mathcal{P}_i^+ = \mathcal{P}_i \cap \mathcal{C}^+$. Then

$$\delta^T = \mathrm{mean}_{i \in I_T} \delta_i^T \text{ and } \delta = \mathrm{mean}_{T \in \mathcal{T}} \delta^T.$$

Outline

Neural Clause Evaluation

Aim for a balance between expressivity and speed of inference!

Neural Clause Evaluation

Aim for a balance between expressivity and speed of inference!

One-off GNN Invocation:

- Graph Neural Networks
- name-invariant formula representations
- relatively expensive; the more context the better
- here: only apply to the input CNF (i.e., only one GNN call)

Neural Clause Evaluation

Aim for a balance between expressivity and speed of inference!

One-off GNN Invocation:

- Graph Neural Networks
- name-invariant formula representations
- relatively expensive; the more context the better
- here: only apply to the input CNF (i.e., only one GNN call)

Generalizing Age and Weight with RvNNs:

- Recursive Neural Networks
- g-age: grow along the clause derivation tree
- g-weight: grow along the clause syntax tree
- share substructures (dag) and cache results

Neural Clause Evaluation

Aim for a balance between expressivity and speed of inference!

One-off GNN Invocation:

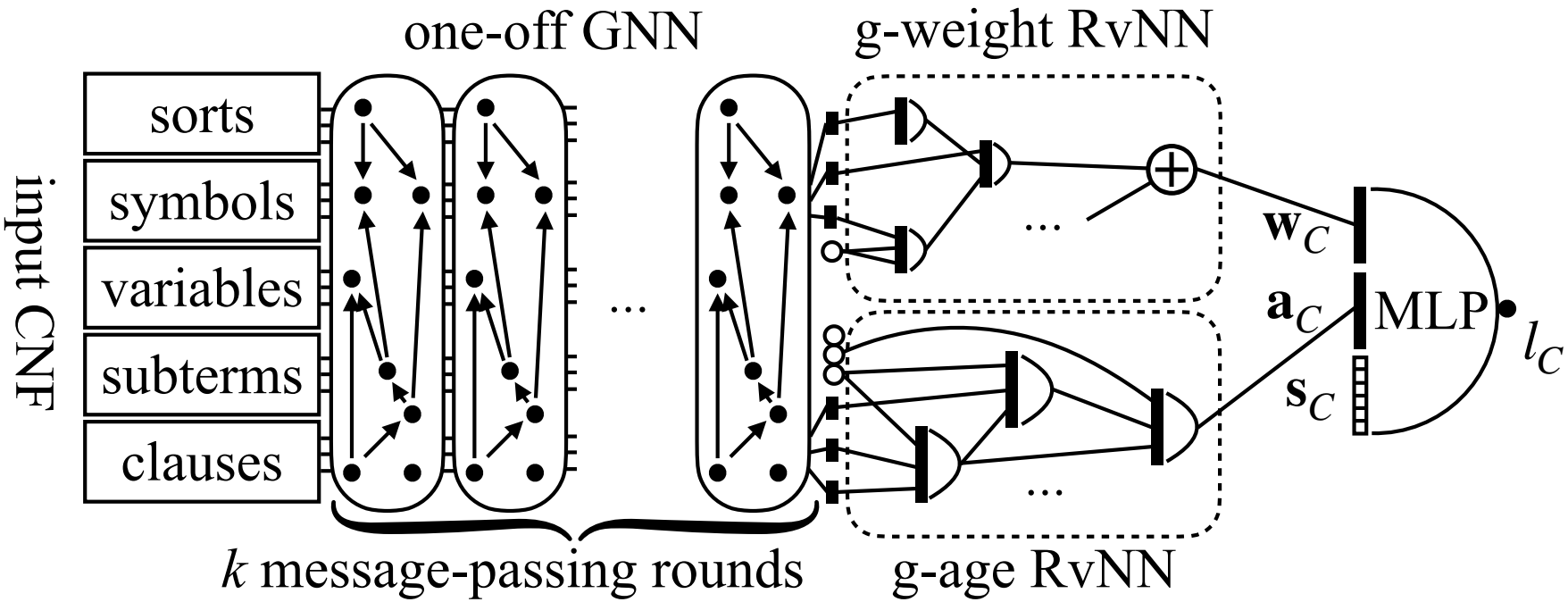
- Graph Neural Networks
- name-invariant formula representations
- relatively expensive; the more context the better
- here: only apply to the input CNF (i.e., only one GNN call)

Generalizing Age and Weight with RvNNs:

- Recursive Neural Networks
- g-age: grow along the clause derivation tree
- g-weight: grow along the clause syntax tree
- share substructures (dag) and cache results

Simple Hand-Crafted Features on Top!

Architecture Diagram



Note on Efficient RvNNs

Strive to compute as much as possible in one bulk!

Note on Efficient RvNNs

Strive to compute as much as possible in one bulk!

```
def gage_insert(cl_num:int,                                          1
                inf_rule:int,                                        2
                parents:list[int]):                                  3
  level = max(base_level,                                            4
              1 + max(height[p] for p in parents))                   5
  height[cl_num] = level                                             6
  index = level - base_level                                         7
  if len(todo_layers) == index:                                      8
    todo_layers.append([])                                           9
  todo_layers[index].append((cl_num, inf_rule, parents))             10
```

Note on Efficient RvNNs

Strive to compute as much as possible in one bulk!

```
def gage_insert(cl_num:int,                                          1
                inf_rule:int,                                        2
                parents:list[int]):                                  3
  level = max(base_level,                                            4
              1 + max(height[p] for p in parents))                   5
  height[cl_num] = level                                             6
  index = level - base_level                                         7
  if len(todo_layers) == index:                                      8
    todo_layers.append([])                                           9
  todo_layers[index].append((cl_num, inf_rule, parents))             10
```

I still need to try out how much GPUs could help here . . .

Outline

Implementation

Single Clause Queue:

- ordered by the computed logits $N_{\theta}(C) = l_C$
- Could we also sample?

Implementation

Single Clause Queue:

- ordered by the computed logits $N_{\theta}(C) = l_C$
- Could we also sample?
- Gumbel-max trick:

Implementation

Single Clause Queue:

- ordered by the computed logits $N_{\theta}(C) = l_C$
- Could we also sample?
- Gumbel-max trick: add some Gumbel noise

$$g = -\log(-\log(u)) \text{ for } u \sim \mathrm{Uniform}(0, 1)$$

Implementation

Single Clause Queue:

- ordered by the computed logits $N_{\theta}(C) = l_C$
- Could we also sample?
- Gumbel-max trick: add some Gumbel noise

$$g = -\log(-\log(u)) \text{ for } u \sim \mathrm{Uniform}(0, 1)$$

Delayed Insertion Buffer:

- insertions into passive are lazy
- only evaluate things in buffer when selection is called

Implementation

Single Clause Queue:

- ordered by the computed logits $N_{\theta}(C) = l_C$
- Could we also sample?
- Gumbel-max trick: add some Gumbel noise

$$g = -\log(-\log(u)) \text{ for } u \sim \mathrm{Uniform}(0, 1)$$

Delayed Insertion Buffer:

- insertions into passive are lazy
- only evaluate things in buffer when selection is called

Iterative Improvement Loop:

- run (guided/plain) prover, collect traces, train from traces
- repeat

Implementation

Single Clause Queue:

- ordered by the computed logits $N_{\theta}(C) = l_C$
- Could we also sample?
- Gumbel-max trick: add some Gumbel noise

$$g = -\log(-\log(u)) \text{ for } u \sim \mathrm{Uniform}(0, 1)$$

Delayed Insertion Buffer:

- insertions into passive are lazy
- only evaluate things in buffer when selection is called

Iterative Improvement Loop:

- run (guided/plain) prover, collect traces, train from traces
- repeat
- little trick; despite the RL heritage:
  inner loop trains until validation loss does not improve

Experiments

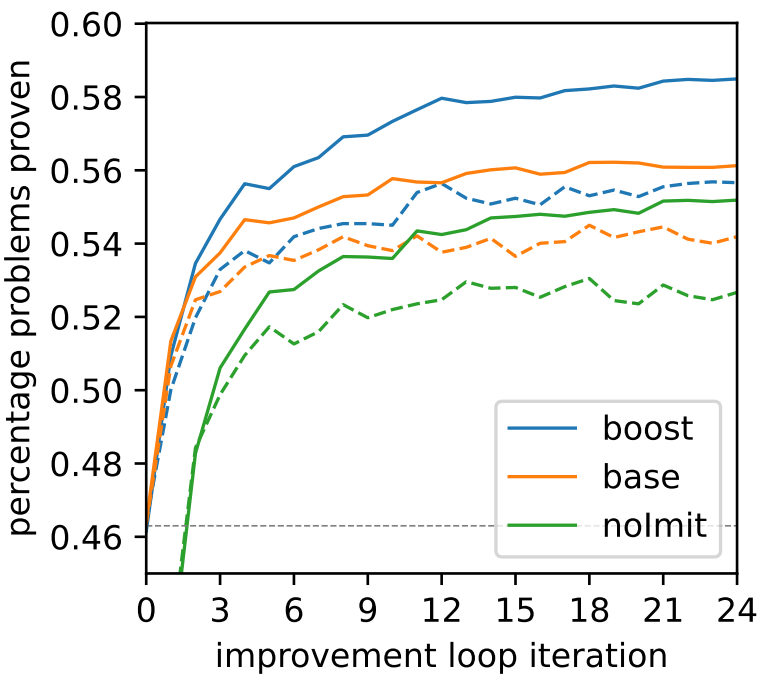
Setup:

- TPTP v9 CNF+FOF, 19 477 problems (train/test split)
- Vampire's default strategy (1:1 age-weight alternation)
- limit of 30 000 Mi (∼10 s) per proof attempt

Experiments

Setup:

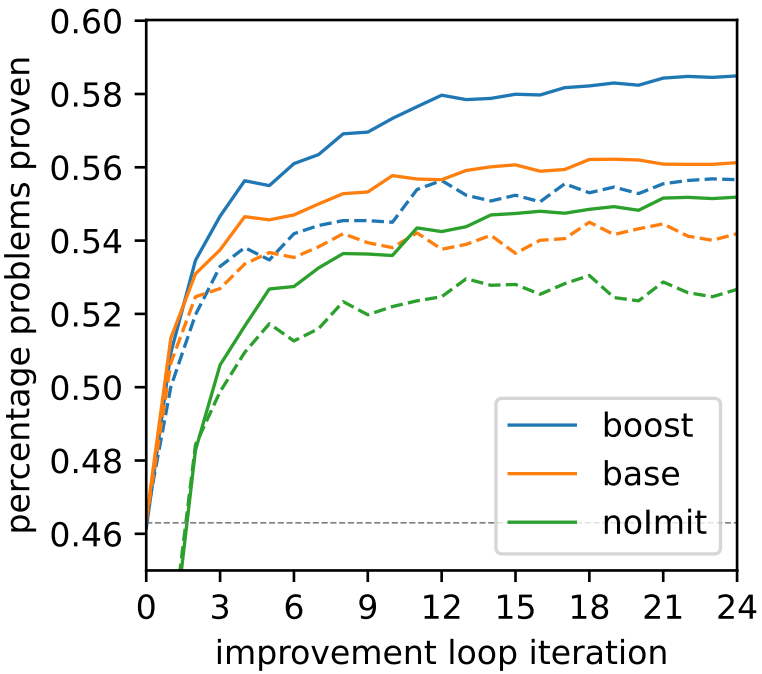
- TPTP v9 CNF+FOF, 19 477 problems (train/test split)
- Vampire's default strategy (1:1 age-weight alternation)
- limit of 30 000 Mi (∼10 s) per proof attempt

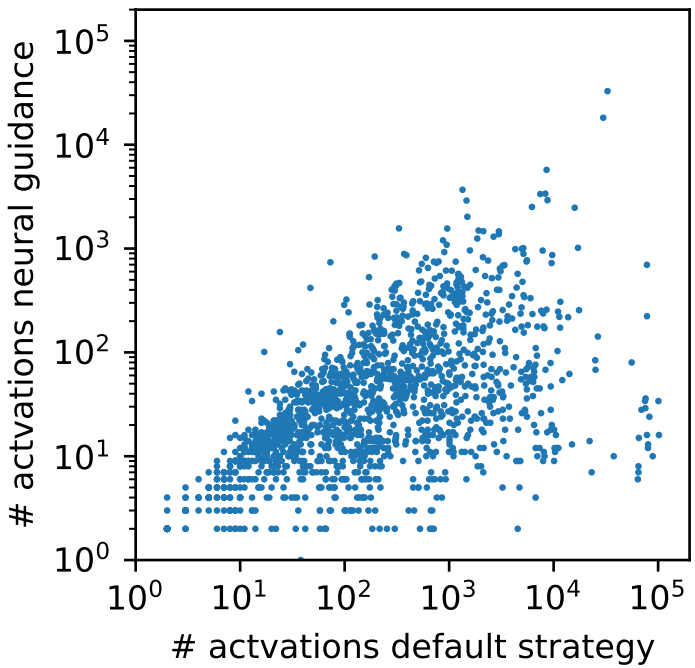


Experiments

Setup:

- TPTP v9 CNF+FOF, 19 477 problems (train/test split)
- Vampire's default strategy (1:1 age-weight alternation)
- limit of 30 000 Mi (∼10 s) per proof attempt





Experiments II

Solving Hard Problems:

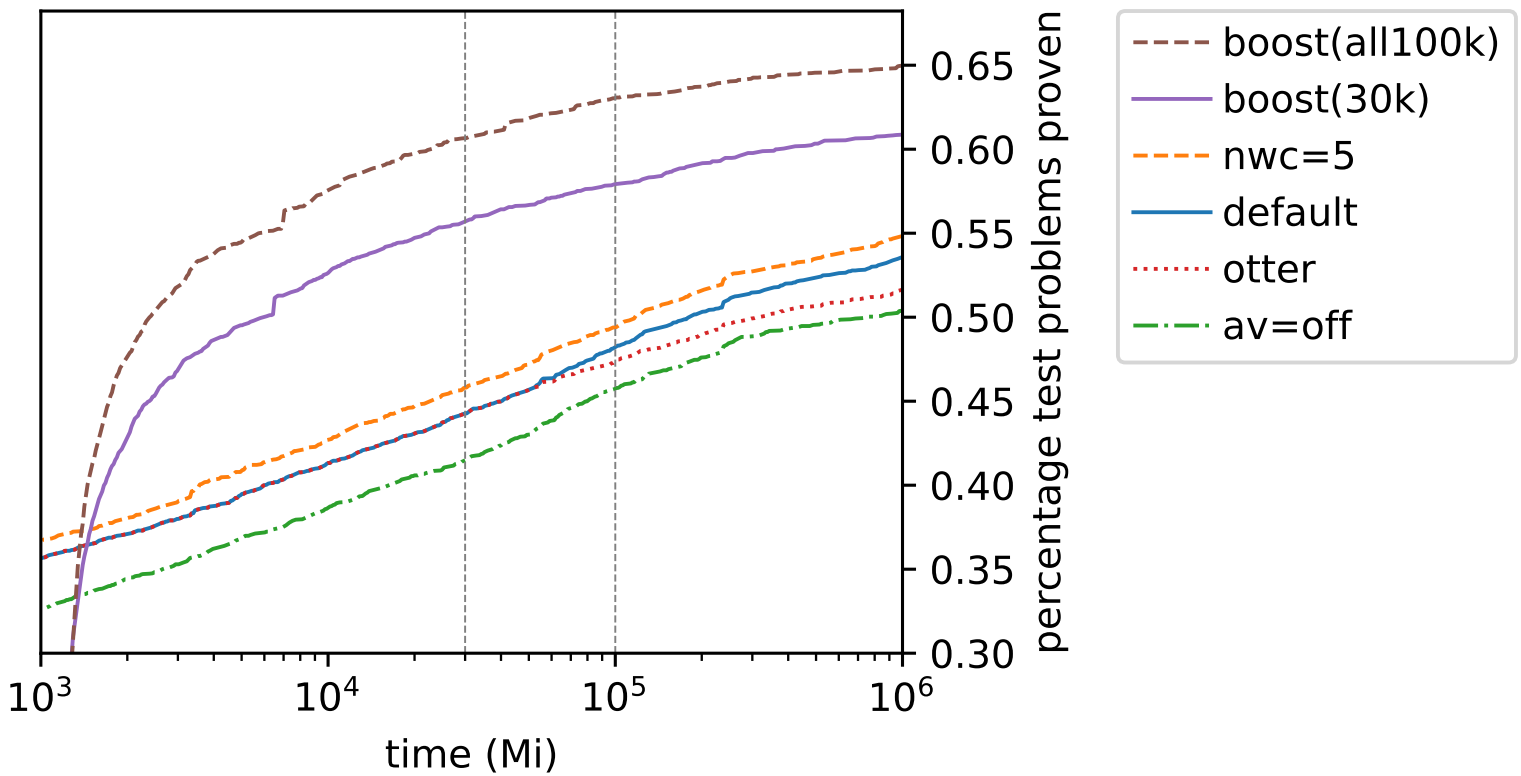
- overfit to TPTP with 100 000 Mi-limited runs
- ran for 12.4 days
- solved 130 rating 1.0 (49 never solved, 8 status UNK)

Experiments II

Solving Hard Problems:

- overfit to TPTP with 100 000 Mi-limited runs
- ran for 12.4 days
- solved 130 rating 1.0 (49 never solved, 8 status UNK)

Put Into Perspective:



Conclusion

Summary:

- new efficient name-invariant neural architecture
- new learning operator inspired by reinforcement learning
- implementation in Vampire
  - 20 % performance boost of the default strategy
  - trained model can solve many very hard (previously unsolved) TPTP problems

Conclusion

Summary:

- new efficient name-invariant neural architecture
- new learning operator inspired by reinforcement learning
- implementation in Vampire
  - 20 % performance boost of the default strategy
  - trained model can solve many very hard (previously unsolved) TPTP problems

Outlook:

- ENIGMA-style vs RL-inspired learning
- other benchmarks than TPTP; e.g. Mizar40; transfer learning
- neural guidance and theorem proving strategies

Conclusion

Summary:

- new efficient name-invariant neural architecture
- new learning operator inspired by reinforcement learning
- implementation in Vampire
  - 20 % performance boost of the default strategy
  - trained model can solve many very hard (previously unsolved) TPTP problems

Outlook:

- ENIGMA-style vs RL-inspired learning
- other benchmarks than TPTP; e.g. Mizar40; transfer learning
- neural guidance and theorem proving strategies

PhD & PostDoc Position Open!

ENIGMA-style

The core idea

Learn to recognize and prefer for selection clauses that look like those that contributed to a proof in past successful runs.

➥ [Schulz00], ENIGMA [Jakubův&Urban17], [Loos et al.'17], ...

ENIGMA-style

The core idea

Learn to recognize and prefer for selection clauses that look like those that contributed to a proof in past successful runs.

➥ [Schulz00], ENIGMA [Jakubův&Urban17], [Loos et al.'17], ...

The "pos/neg"s of E:

E prover can be asked to output, for every clause selected in a run, whether it ended up in the final proof (pos) or not (neg)

ENIGMA-style

The core idea

Learn to recognize and prefer for selection clauses that look like those that contributed to a proof in past successful runs.

➥ [Schulz00], ENIGMA [Jakubův&Urban17], [Loos et al.'17], ...

The "pos/neg"s of E:

E prover can be asked to output, for every clause selected in a run, whether it ended up in the final proof (pos) or not (neg)

Next comes the ML:

- represent those clauses somehow (features, NNs, ...)
- train a binary classifier on the task
- integrate back with the prover:

ENIGMA-style

The core idea

Learn to recognize and prefer for selection clauses that look like those that contributed to a proof in past successful runs.

➥ [Schulz00], ENIGMA [Jakubův&Urban17], [Loos et al.'17], ...

The "pos/neg"s of E:

E prover can be asked to output, for every clause selected in a run, whether it ended up in the final proof (pos) or not (neg)

Next comes the ML:

- represent those clauses somehow (features, NNs, ...)
- train a binary classifier on the task
- integrate back with the prover: "try to do more of the pos"

Possible Ways of Integrating the Learnt Advice

Priority:

- sort by model's Y/N and tiebreak by age
- 

Possible Ways of Integrating the Learnt Advice

Priority:

- sort by model's Y/N and tiebreak by age
- 

Logits:

- even a binary classifier internally uses a real value
- 

Possible Ways of Integrating the Learnt Advice

Priority:

- sort by model's Y/N and tiebreak by age
- 

Logits:

- even a binary classifier internally uses a real value
- 

Combine with the original strategy

$\mathcal{S} \oplus \mathcal{M}^{1,0}$: