# DeepIsaHOL progress report: current machine learning for the Isabelle proof assistant

Jonathan Julián Huerta y Munive
huertjon@cvut.cz
Czech Technical University in Prague
April, 2025

CZECH INSTITUTE OF INFORMATICS ROBOTICS AND CYBERNETICS CTU IN PRAGUE

Funded by the European Union

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union. The European Union cannot be held responsible for them.

# General Takeaway

*Me*: Come here as a user of interactive theorem provers (ITPs)

*DeepIsaHOL Objective:* create tools that serve the Machine Learning (ML) and the ITP community for advancing the state of the art in ML and ITP

Let's collaborate!

***Example***



# Summary

*EuroProofNet Workshop: to discuss the state-of-the-art and future directions in Theorem Proving and Machine Learning*

Discussion points:

- overview of the (interacting) proving experience
- (non-exhaustive) overview of previous ML + ITP integrations (reviewer's requested)
- lessons learned from previous tools (reviewer's requested)
- what has already been done in DeepIsaHOL? (reviewer's requested)
- why Isabelle? (reviewer's requested)
- what is there still to do? (reviewer's requested)

# Overview of the Isabelle proving experience

# An Isabelle apply-style proof

Tactics for backward reasoning from goal to hypothesis
(tactics inputs: terms and premises)

```
lemma
  fixes P :: "('a::finite) ⇒ nat ⇒ bool"
  assumes "∀i. ∃N::nat. ∀n ≥ N. P i n"
  shows "∃N. ∀i. ∀n ≥ N. P i n"
  apply (rule_tac x="Max {Inf {N. ∀n≥N. P i n} |i::'a. i ∈ UNIV}" in exI)
  apply clarify
  subgoal for i n
    using wellorder_Inf_lemma[of "λn. P i n"] assms
      Max_ge[of "{Inf {N. ∀n≥N. P i n} |i::'a. i ∈ UNIV}", OF finite_image]
    apply -
    apply (erule_tac x=i in allE)
    apply (subgoal_tac "∀n≥Inf {N. ∀n≥N. P i n}. P i n")
```

```
proof (prove)
goal (1 subgoal):
 1. ∃N. ∀i n. N ≤ n ⟶ P i n
```

```
proof (prove)
goal (1 subgoal):
 1. ⋀i n. Max {Inf {N. ∀n≥N. P i n} |i. i ∈ UNIV} ≤ n ⟹ P i n
```

```
proof (prove)
goal (1 subgoal):
 1. Max {Inf {N. ∀n≥N. P i n} |i. i ∈ UNIV} ≤ n ⟹ P i n
```

# Hammers

Calls to automated theorem provers (ATPs) to reduce the number of premises needed to automatically prove the current proof obligation

```
lemma
  fixes P :: "('a::finite) ⇒ nat ⇒ bool"
  assumes "∀i. ∃N::nat. ∀n ≥ N. P i n"
  shows "∃N. ∀i. ∀n ≥ N. P i n"
  apply (rule_tac x="Max {Inf {N. ∀n≥N. P i n} |i::'a. i ∈ UNIV}" in exI)
  apply clarify
  subgoal for i n
    using wellorder_Inf_lemma[of "λn. P i n"] assms
      Max_ge[of "{Inf {N. ∀n≥N. P i n} |i::'a. i ∈ UNIV}", OF finite_image]
    apply -
    apply (erule_tac x=i in allE)
    apply (subgoal_tac "∀n≥Inf {N. ∀n≥N. P i n}. P i n")
    sledgehammer
```

```
proof (prove)
goal (1 subgoal):
 1. ∃N. ∀i n. N ≤ n ⟶ P i n
```

```
Sledgehammering...
vampire found a proof...
spass found a proof...
vampire: Try this: apply (metis (mono_tags, lifting) UNIV_I dual_order.trans mem_Collect_eq) (90 ms)
spass: Try this: apply (metis (mono_tags, lifting) iso_tuple_UNIV_I mem_Collect_eq order_trans) (40 ms)
Done
```

external automated theorem provers (ATPs)

trusted prover

premise selection (3 premises selected)

# An Isabelle apply-style proof

Iterate based on feedback and finish the proof

```
lemma
  fixes P :: "('a::finite) ⇒ nat ⇒ bool"
  assumes "∀i. ∃N::nat. ∀n ≥ N. P i n"
  shows "∃N. ∀i. ∀n ≥ N. P i n"
  apply (rule_tac x="Max {Inf {N. ∀n≥N. P i n} |i::'a. i ∈ UNIV}" in exI)
  apply clarify
  subgoal for i n
    using wellorder_Inf_lemma[of "λn. P i n"] assms
      Max_ge[of "{Inf {N. ∀n≥N. P i n} |i::'a. i ∈ UNIV}", OF finite_image]
    apply -
    apply (erule_tac x=i in allE)
    apply (subgoal_tac "∀n≥Inf {N. ∀n≥N. P i n}. P i n")
    apply (metis (mono_tags, lifting) dual_order.trans iso_tuple_UNIV_I mem_Collect_eq)
    apply blast
    done
done
```

```
proof (prove)
goal (1 subgoal):
 1. ∃N. ∀i n. N ≤ n ⟶ P i n
```

```
proof (prove)
goal (1 subgoal):
 1. ∀i n. Max {Inf {N. ∀n≥N. P i n} |i. i ∈ UNIV} ≤ n ⟶ P i n
```

```
proof (prove)
goal (1 subgoal):
 1. ⋀i n. Max {Inf {N. ∀n≥N. P i n} |i. i ∈ UNIV} ≤ n ⟹ P i n
```

```
proof (prove)
goal (1 subgoal):
 1. Max {Inf {N. ∀n≥N. P i n} |i. i ∈ UNIV} ≤ n ⟹ P i n
```

```
proof (prove)
goal:
No subgoals!
```

# An Isabelle Isar-style Proof

- More human readable proofs
- Standard in the Archive of Formal Proofs (AFP)

```
lemma
  fixes P :: "('a::finite) ⇒ nat ⇒ bool"
  assumes "∀i. ∃N::nat. ∀n ≥ N. P i n"
  shows "∃N. ∀i. ∀n ≥ N. P i n"
proof-
  let "?bound i" = "Inf {N . ∀n ≥ N. P i n}"
  let ?N = "Max {?bound i |i. i ∈ UNIV}"
  {fix n::nat and i::'a
    from assms
    obtain M where "∀n ≥ M. P i n"
      \<proof>
    hence obs:"∀ m ≥ ?bound i. P i m"
      \<proof>
    assume "n ≥ ?N"
    also have "?N ≥ ?bound i"
      \<proof>
    ultimately have "n ≥ ?bound i"
      \<proof>
    hence "P i n"
      \<proof>
  }
  thus "∃N. ∀i n. N ≤ n ⟶ P i n"
    \<proof>
qed
```

```
proof (prove)
goal (1 subgoal):
 1. ∃N. ∀i n. N ≤ n ⟶ P i n
```

```
proof (prove)
using this:
  ∀i. ∃N. ∀n≥N. P i n

goal (1 subgoal):
 1. (⋀M. ∀n≥M. P i n ⟹ thesis) ⟹ thesis
```

```
proof (prove)
using this:
  ∀n≥M. P i n

goal (1 subgoal):
 1. ∀m≥Inf {N. ∀n≥N. P i n}. P i m
```

```
proof (state)
this:
  Max {Inf {N. ∀n≥N. P i n} |i. i ∈ UNIV} ≤ ?n2 ⟹ P ?i2 ?n2

goal (1 subgoal):
 1. ∃N. ∀i n. N ≤ n ⟶ P i n
```

# An Isabelle Isar-style Proof

User completes apply-style intermediate steps

```
lemma
  fixes P :: "('a::finite) ⇒ nat ⇒ bool"
  assumes "∀i. ∃N::nat. ∀n ≥ N. P i n"
  shows "∃N. ∀i. ∀n ≥ N. P i n"
proof-
  let "?bound i" = "Inf {N . ∀n ≥ N. P i n}"
  let ?N = "Max {?bound i |i. i ∈ UNIV}"
  {fix n::nat and i::'a
    from assms
    obtain M where "∀n ≥ M. P i n"
      by blast
    hence obs:"∀ m ≥ ?bound i. P i m"
      using wellorder_Inf_lemma[of "λn. P i n"]
      by blast
    assume "n ≥ ?N"
    also have "?N ≥ ?bound i"
      using finite_image
      by(fastforce intro: Max_ge)
    ultimately have "n ≥ ?bound i"
      using order.trans
      by blast
    hence "P i n"
      using obs
      by blast
  }
  thus "∃N. ∀i n. N ≤ n ⟶ P i n"
    by blast
qed
```

```
proof (prove)
goal (1 subgoal):
 1. ∃N. ∀i n. N ≤ n ⟶ P i n
```

```
proof (prove)
using this:
  ∀i. ∃N. ∀n≥N. P i n

goal (1 subgoal):
 1. (⋀M. ∀n≥M. P i n ⟹ thesis) ⟹ thesis
```

```
proof (prove)
using this:
  ∀n≥M. P i n

goal (1 subgoal):
 1. ∀m≥Inf {N. ∀n≥N. P i n}. P i m
```

```
proof (prove)
using this:
  ▪ ∀n≥M. P i n
  ▪ ∃N. ∀n≥N. P i n ⟹ P i (Inf {N. ∀n≥N. P i n})
  ▪ ∃N. ∀n≥N. P i n ⟹ ∀n≥Inf {N. ∀n≥N. P i n}. P i n

goal (1 subgoal):
 1. ∀m≥Inf {N. ∀n≥N. P i n}. P i m
```

```
proof (state)
this:
  Max {Inf {N. ∀n≥N. P i n} |i. i ∈ UNIV} ≤ ?n2 ⟹ P ?i2 ?n2

goal (1 subgoal):
 1. ∃N. ∀i n. N ≤ n ⟶ P i n
```

```
goal:
No subgoals!
```

# Previous machine learning approaches on interactive theorem provers

Detailed:

- *TacticToe for HOL4*
- *Draft, Sketch, Proof and Magnushammer for Isabelle*

# Previous ML approaches on ITPs: TacticToe

k-nearest neighbours (kNN) and Monte-Carlo tree search (MCTS) to predict next proof-step

Proves 66.4% of 7164 top-level theorems from HOL4 standard library

```
lemma
  fixes P :: "('a::finite) ⇒ nat ⇒ bool"
  assumes "∀i. ∃N::nat. ∀n ≥ N. P i n"
  shows "∃N. ∀i. ∀n ≥ N. P i n"
  apply (rule_tac x="Max {Inf {N. ∀n≥N. P i n} |i::'a. i ∈ UNIV}" in exI)
  apply clarify
  subgoal for i n
    using wellorder_Inf_lemma[of "λn. P i n"] assms
      Max_ge[of "{Inf {N. ∀n≥N. P i n} |i::'a. i ∈ UNIV}", OF finite_image]
    apply -
    apply (erule_tac x=i in allE)
    apply (subgoal_tac "∀n≥Inf {N. ∀n≥N. P i n}. P i n")
    apply (metis (mono_tags, lifting) dual_order.trans iso_tuple_UNIV_I mem_Collect_eq)
    apply blast
    done
done
```

```
proof (prove)
goal (1 subgoal):
 1. ⋀i n. Max {Inf {N. ∀n≥N. P i n} |i. i ∈ UNIV} ≤ n ⟹ P i n ≡ φ
```

$$\min_{\varphi_i \in \mathcal{D}, 1<=i<=k} d(\varphi, \varphi_i)$$

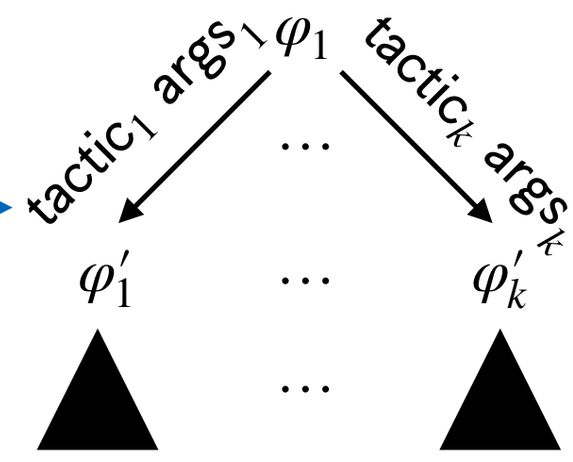
$$\text{arg} = d(\text{known\_thm}, \varphi)$$

**kNN**

$\mathcal{D}$

| Tactic | Goal | Outputs |
|---|---|---|
| clarify | $\varphi_1$ | thms1 |
| blast | $\varphi_2$ | thms2 |
| auto | $\varphi_3$ | thms3 |

**MCTS**



# Previous ML approaches on ITPs: “Draft, Sketch, Prove” and Magnushammer

Isabelle

Informal to LaTeX

Formal proof sketching

LaTeX

```
lemma
  fixes P :: "('a::finite) ⇒ nat ⇒ bool"
  assumes "∀i. ∃N::nat. ∀n ≥ N. P i n"
  shows "∃N. ∀i. ∀n ≥ N. P i n"
proof-
  let "?bound i" = "Inf {N . ∀n ≥ N. P i n}"
  let ?N = "Max {?bound i |i. i ∈ UNIV}"
  {fix n::nat and i::'a
    from assms
    obtain M where "∀n ≥ M. P i n"
      \<proof>
    hence obs:"∀ m ≥ ?bound i. P i m"
      \<proof>
    assume "n ≥ ?N"
    also have "?N ≥ ?bound i"
      \<proof>
    ultimately have "n ≥ ?bound i"
      \<proof>
    hence "P i n"
      \<proof>
  }
  thus "∃N. ∀i n. N ≤ n ⟶ P i n"
    \<proof>
qed
```

Call hammers to complete the proofs

Since ATPs only serve for premise selection, replace them with neural networks (NNs)

**Combined approach:** Proof-rate of 71% on publicly available benchmark of 183K Isabelle proofs from the AFP

# Other previous ML approaches on ITPs

- **TacticZero for HOL4:** It is an RL approach using NNs. It achieves ~49% success on a test set of 268 problems.

- **Tactician for Coq (now Rocq):** It is similar to TacticToe but extends the approach with a graph neural network (GNN) and compares it with the kNN.
  - It compares them with an LLM trained only on Coq data.
  - Intersection between kNN and GNN is thin.
  - GNN almost fully covers CoqHammer's results while
  - GNN proves some that CoqHammer does not prove.

- **HOL-list environment for HOL Light:** It is a light-weight tool for testing different approaches for tactic and premise selection with a Python interface.
  - It has tested GNNs and an RL algorithm.
  - The RL approach solves 56.3% of a set of 3225 theorems of the HOL Light's mathematical library.

- **LeanDojo for Lean:** Open project that interfaces Python with the Lean ITP.

- **More LLMs in Lean:** Many recent articles on using LLMs for autoformalization, premise selection, tactic suggestion, and proof search.

# Previous ML approaches on ITPs

| Prover\Strategy | Premise selection | Reinforcement Learning | Next step prediction for proof completion | Autoformalisation |
|---|---|---|---|---|
| **HOL4** | TacticToe (kNN) | Tactic Zero | TacticToe (kNN) | |
| **Coq (now Rocq)** | | | CoqPilot, Tactician (kNN, GNN, and LLM) | |
| **Isabelle** | Magnushammer | | Thor, ***DeepIsaHOL*** | Draft-Sketch-Prove, LEGO-Prover |
| **Lean** | | | LeanDojo's ReProver | Deep Seek Prover |
| **HOL Light** | | *HOL-list (RL)* | *HOL-list (GNN)* | |

# Lessons learned

# Lessons learned

- Main approaches discussed:
  - ML (kNN or GNN or LLM) for predicting next tactic + premises
  - LLM for autoformalisation + hammer for premise selection
  - Medium LM as a replacement for hammer as premise selection
  - Reinforcement learning replacing the standard best-first search exploration

- Language models alone have not sufficed for full automation in theorem proving

- Premise selection is a very important aid for small and medium sized language models in these projects (takes them as far as 50-60% on already curated proofs)

- The approach of scaling the model to a LARGE language model for memorising proof structures and autoformalising also helps

- More work is needed to:
  - Combine complementary approaches to improve proof success rate
  - Compare these approaches in terms of resources needed (training time, testing time, computing power) and across provers
  - Create tools for users (only know of TacticToe, Tactician, LeanCopilot)

# Progress: Project’s current outputs

# 1. Data generation example output as JSON attributes

Simple interface for data extraction either as a single command:

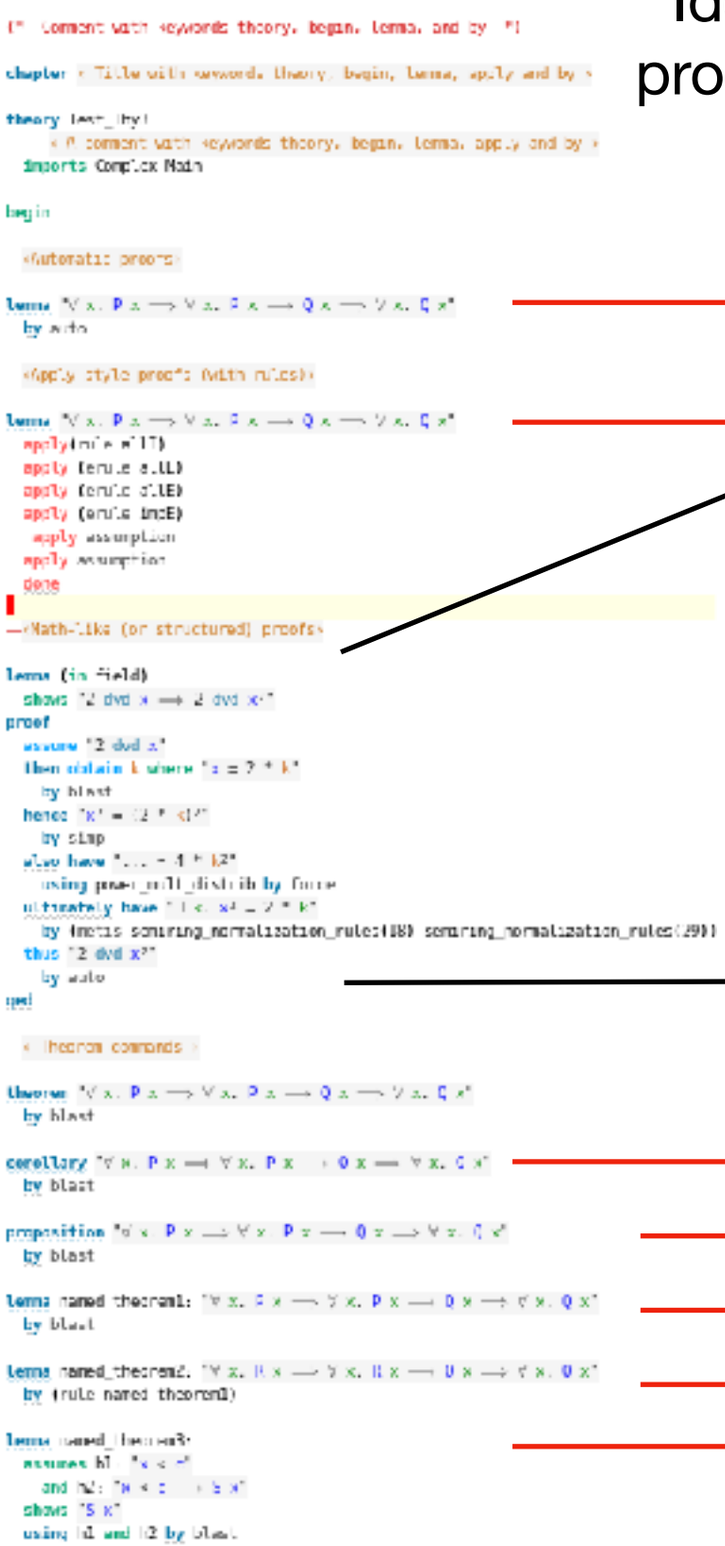
```
sbt "run /path/to/a/read/directory/ /path/to/a/writing/directory/"
```

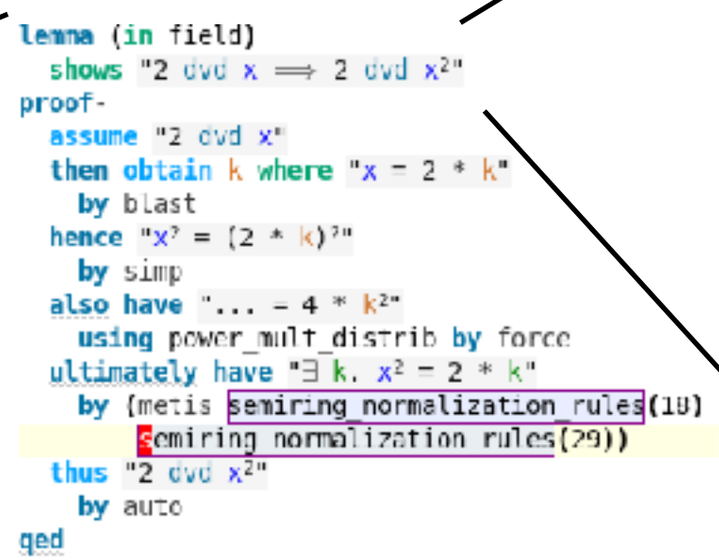
or via the Scala or Python REPL:

```
import isabelle_rl._
import de.unruh.isabelle.control.Isabelle

val logic = "LTL" // Isabelle session name

val writer = new Writer(
    "/path/to/a/read/directory/", // it can have a ROOT, or a .thy file
    "/path/to/a/writing/directory/",
    logic)

writer.write_data("your/file.thy")

writer.write_all()

val minion = writer.get_minion()

implicit val isabelle: Isabelle = minion.isabelle

isabelle.destroy()
```

# 1. Data generation example output as JSON attributes

Identifies each proof in a .thy file








For each step (command + argument) in the proof it writes specific step's data

# 1. Data generation example output as JSON attributes


```
"term": "(dvd.dvd times (numeral.numeral one plus (num.Bit0 num.One)) x
  \\<Longrightarrow> dvd.dvd times (numeral.numeral one plus (num.Bit0 num.One)) (power.power one times x 2))
  \\<Longrightarrow> (dvd.dvd times (numeral.numeral one plus (num.Bit0 num.One)) x
  \\<Longrightarrow> dvd.dvd times (numeral.numeral one plus (num.Bit0 num.One)) (power.power one times x 2))",
```

```
"thms": [
  {"thm": {"name": "Pure.protectI", "term": "PROP ?A \\<Longrightarrow> (PROP ?A)"}},
  {"thm": {"name": "Pure.protectD", "term": "(PROP ?A) \\<Longrightarrow> PROP ?A"}},
  {"thm": {"name": "HOL.allE", "term": "\\<forall>x. ?P x \\<Longrightarrow> (?P ?x \\<Longrightarrow> ?R) \\<Longrightarrow> ?R"}},
  {"thm": {"name": "HOL.notE", "term": "\\<not> ?P \\<Longrightarrow> ?P \\<Longrightarrow> ?R"}},
  ....
  {"thm": {"name": "HOL.allI", "term": "(\\<And>x. ?P x) \\<Longrightarrow> \\<forall>x. ?P x"}},
],
```

```
"keywords": [
  {"name": "\\<proof>"},
  {"name": "sorry"},
  {"name": "by"},
  {"name": ".."},
  {"name": "."},
  {"name": "unfolding"},
  {"name": "including"},
  {"name": "using"},
  {"name": "prefer"},
  {"name": "defer"},
  {"name": "apply"},
  {"name": "back"},
  {"name": "done"},
  {"name": "oops"},
  {"name": "proof"},
  {"name": "subgoal"}
],
```


- More lemmas found per .thy file than in previous Isabelle-focused approaches
- More context-aware data generation: syntax, keywords, methods and theorems are user-extensible, and the data-generation accounts for that


```
"methods": [
  {"name": "HOL.split"},
  {"name": "HOL.force"},
  {"name": "HOL.cases"},
  {"name": "HOL.blast"},
  ....
  {"name": "HOL.auto"},
  {"name": "SMT.smt"},
  {"name": "SAT.sat"},
  {"name": "Pure.-"}
],
```

# 2. Data interaction example

Simple interface for interaction with Isabelle either in Scala or Python:

```
>>> import sys
>>> sys.path.append('/path/to/this/project/src/main/python')
>>> from repl import REPL

>>> repl = REPL("HOL")
REPL and minion initialized.

>>> repl.apply("lemma \"\\<forall>x. P x \\<Longrightarrow> P c\"")
'proof (prove) goal (1 subgoal): 1. \\<forall>x. P x \\<Longrightarrow> P
    c'

>>> repl.apply("apply simp")
'proof (prove)  goal:  No subgoals!'

>>> repl.apply("done")
''
```

# 3. Training of T5 LLMs for simple experiments

| Model | Predictions | Progress | by commands | Correct by | Proofs | Finished proofs |
|---|---|---|---|---|---|---|
| t5_small (simple) | 4,185,571 | 62.48 % | 98,369 | 41.05 % | 24,071 | 6.63 % |
| t5_small (finetuned) | 2,055,838 | 37.29 % | 97,907 | 8.29 % | 24,740 | 11.46 % |
| t5_small (more data per sample) | 1,859,629 | 49.01 % | 35,885 | 14.76 % | 27,597 | 8.55 % |
| t5_base | 2,983,954 | 60.65 % | 48,141 | 26.78 % | 20,252 | 8.92 % |

Model performance metrics after 4 training iterations through the 64% of the AFP except for the more-data-per-sample (3 iterations). Ongoing development of the proof exploration tree.

# Next steps

# Next steps

- Implementation of a generic suggestion functionality that sends data from Isabelle and retrieves the response from a model

- Continue the small scale experiment
  - Variation of the model architecture (from encoder-decoder to autoencoder)
  - Usage of the latest optimised models (Gemma or DeepSeek-R1)
  - Train on small-scale premise selection
  - Train on “conjecturing” predicting the next state
  - Variations on the proof-exploration tree (best-first search, MCTS)

- Create infrastructure for reinforcement learning

- Focus on cleverly engineered premise selection to improve the models’ performance

# Conclusion

- Isabelle chosen because of
  - its closeness to natural language via its scripting language Isar
  - its automation via Sledgehammer
  - its large repository of proofs
  - its growing number of users

- The literature shows that progress in premise selection leads to better performance for next step predictors

- DeepIsaHOL is an open project intended to serve the ML community and the ITP community.

- You are all invited to collaborate, use it, and provide feedback

Jonathan Julián Huerta y Munive
huertjon@cvut.cz
Czech Technical University in Prague

https://github.com/yonoteam/DeepIsaHOL

# Extras

# Extras on the small-scale experiment 1

| Model | Train Loss | Train Accuracy | Validation Loss | Validation Accuracy |
|---|---|---|---|---|
| t5_small (simple) | 3.02 | 0.445 | 3.13 | 0.436 |
| t5_small (finetuned) | 1.18 | 0.720 | 1.14 | 0.733 |
| t5_small (more data per sample) | 2.96 | 0.417 | 3.29 | 0.375 |
| t5_base | 2.84 | 0.474 | 3.06 | 0.441 |

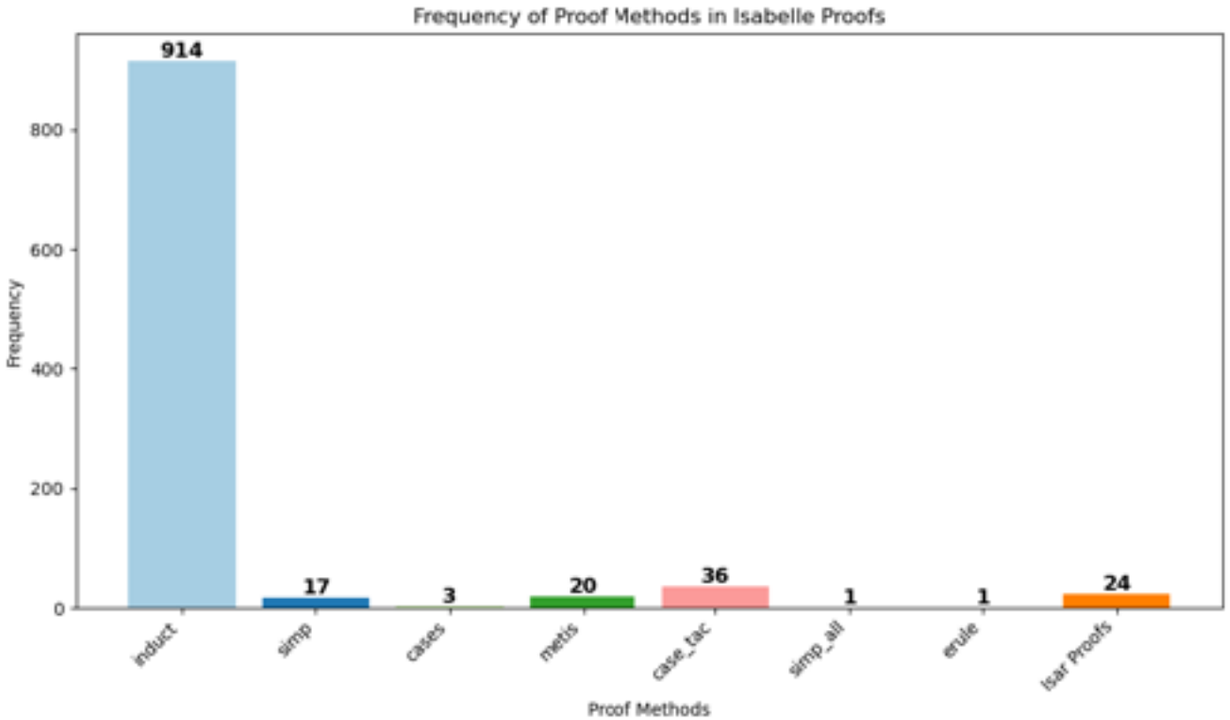
Model train and validation loss and accuracy after 4 training iterations through the AFP except for the more-data-per-sample (3 iterations)

# Extras on the small-scale experiment 2

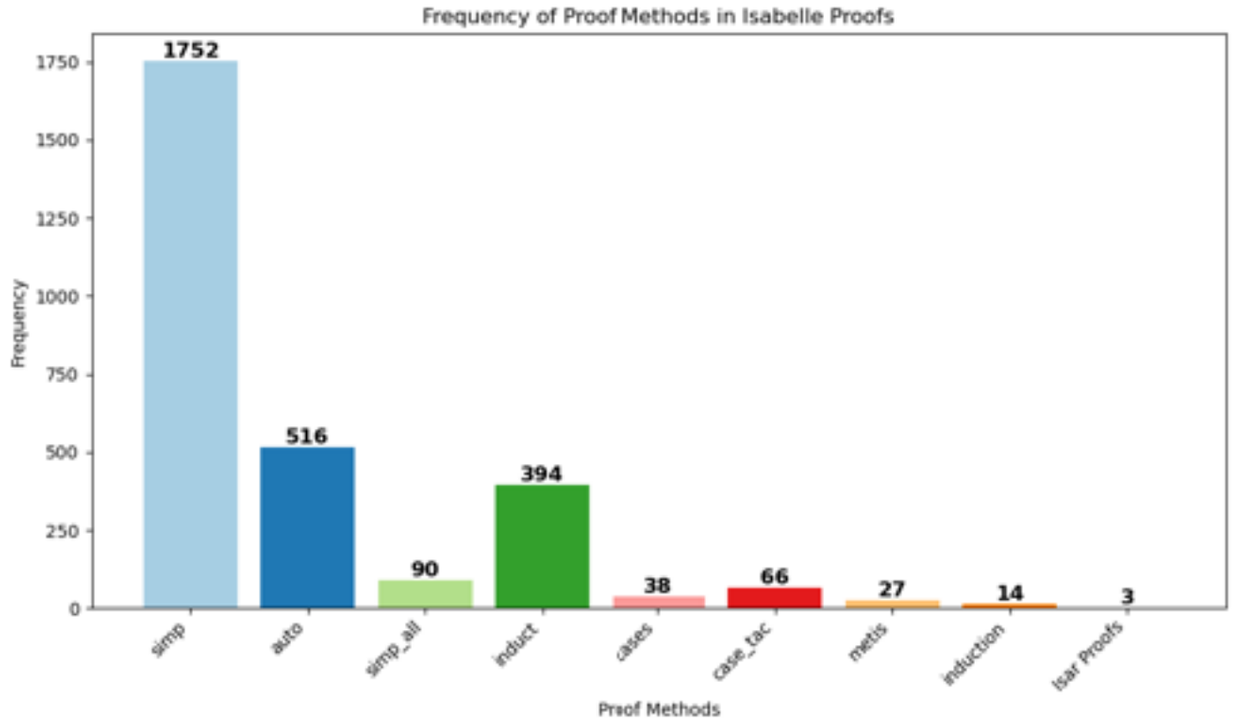
| Model | Exact Accuracy | 1st-Token Accuracy | All wrong ratio |
|---|---|---|---|
| t5_small (simple) | 0.137 | 0.316 | 0.303 |
| t5_small (finetuned) | 0.150 | 0.302 | 0.231 |
| t5_small (more data per sample) | 0.042 | 0.254 | 0.553 |
| t5_base | 0.170 | 0.307 | 0.299 |

Model performance on metrics over the testing dataset

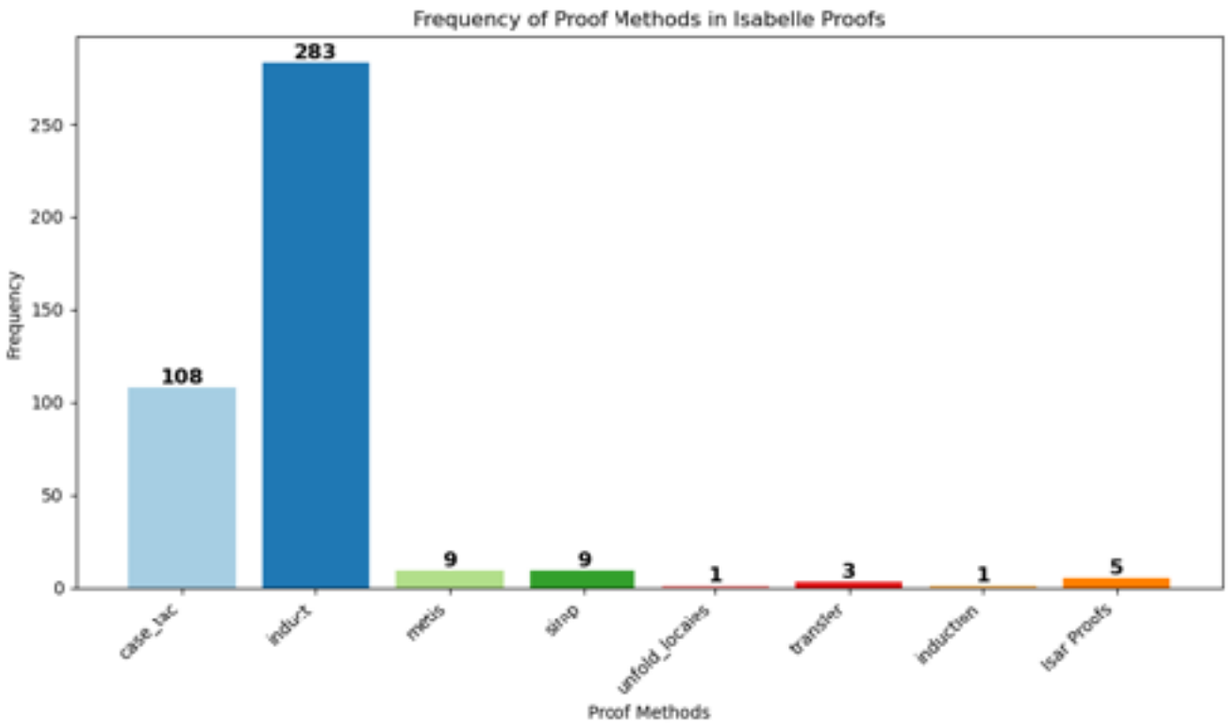
# Extras on the small-scale experiment 3



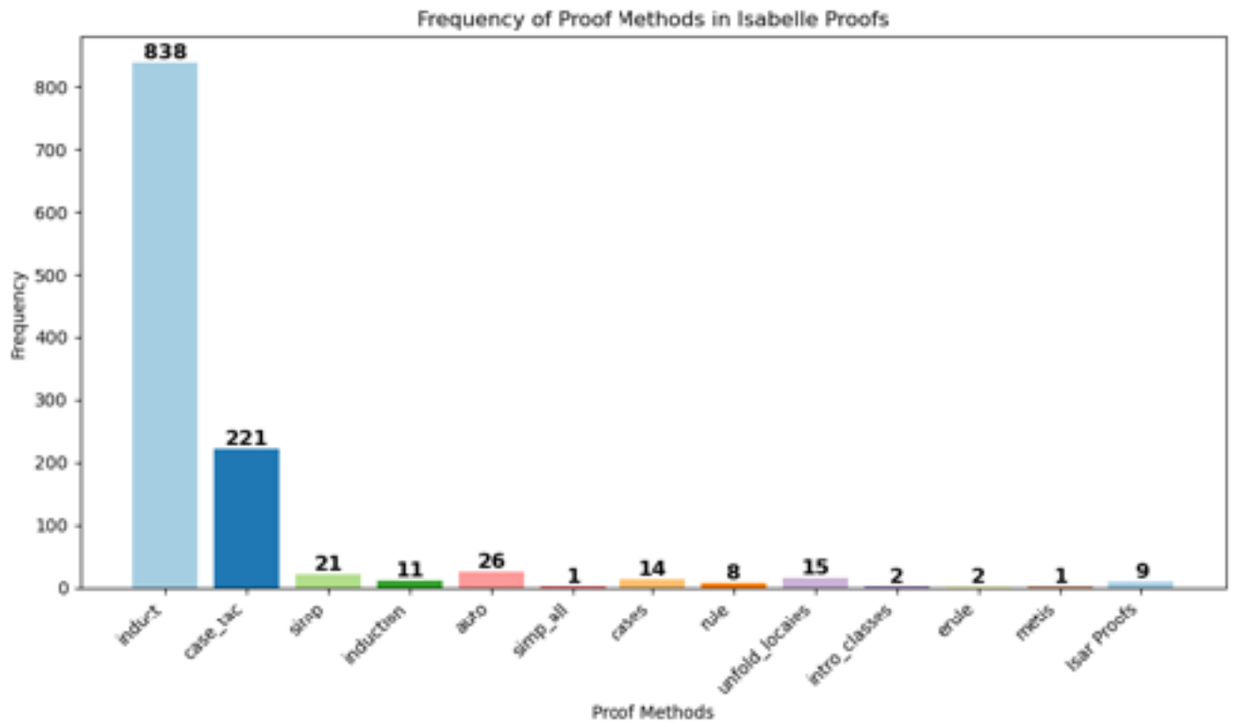

t5_small (simple)




t5_small (finetuned)




t5_small (more data per sample)




t5_base (simple)

# Extras on the small-scale experiment 4

