A project report on proof data extraction, interaction, and evaluation in Isabelle

Jonathan Julián Huerta y Munive
jjhymuni@cs.aau.dk
Aalborg University, Copenhagen, Denmark
September 21, 2026



AALBORG UNIVERSITY



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.

An Isabelle proof

User action: a command and its arguments

```
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
```

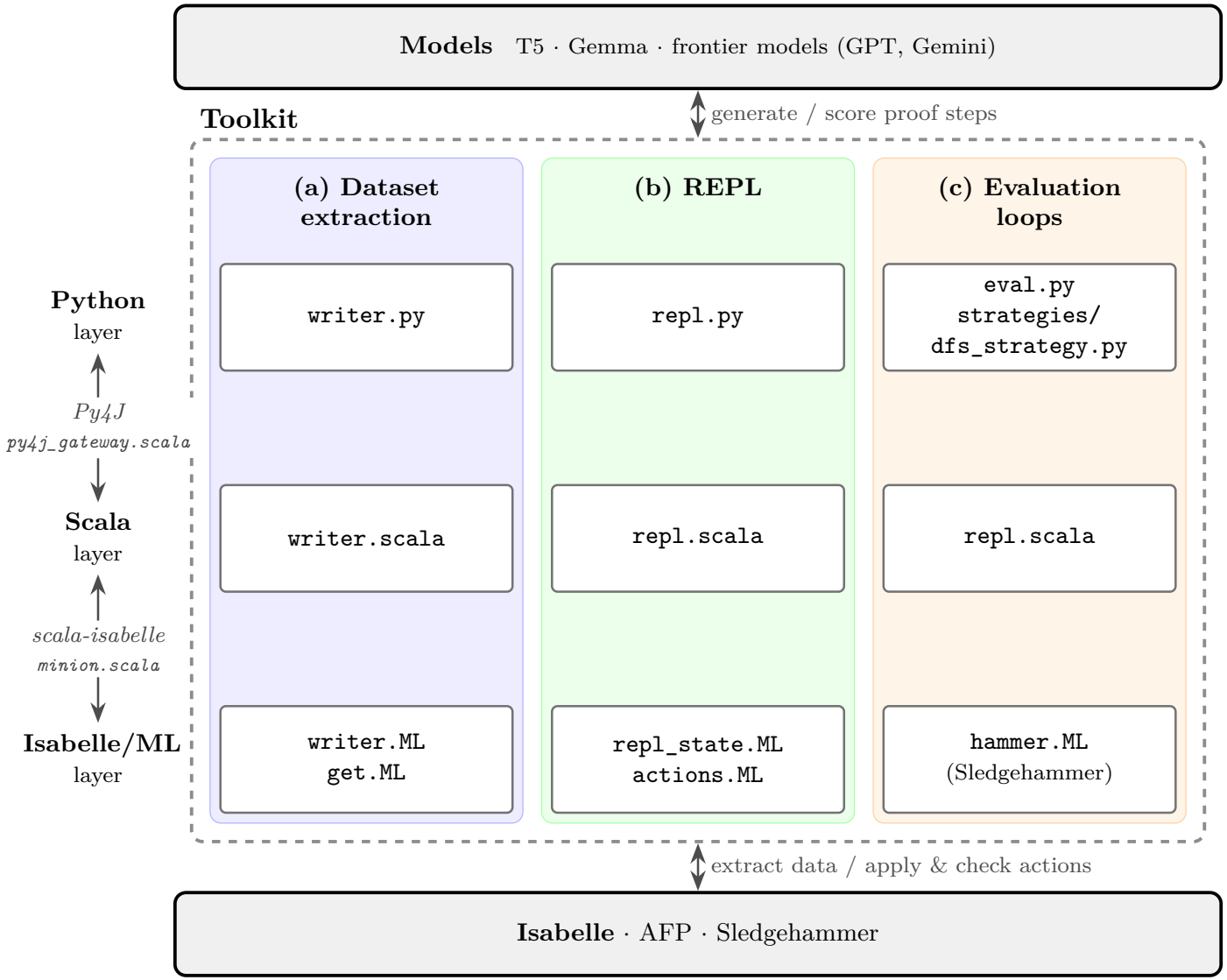
User states: the output the user sees in-between actions

```
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
```

The contributions (and architecture)



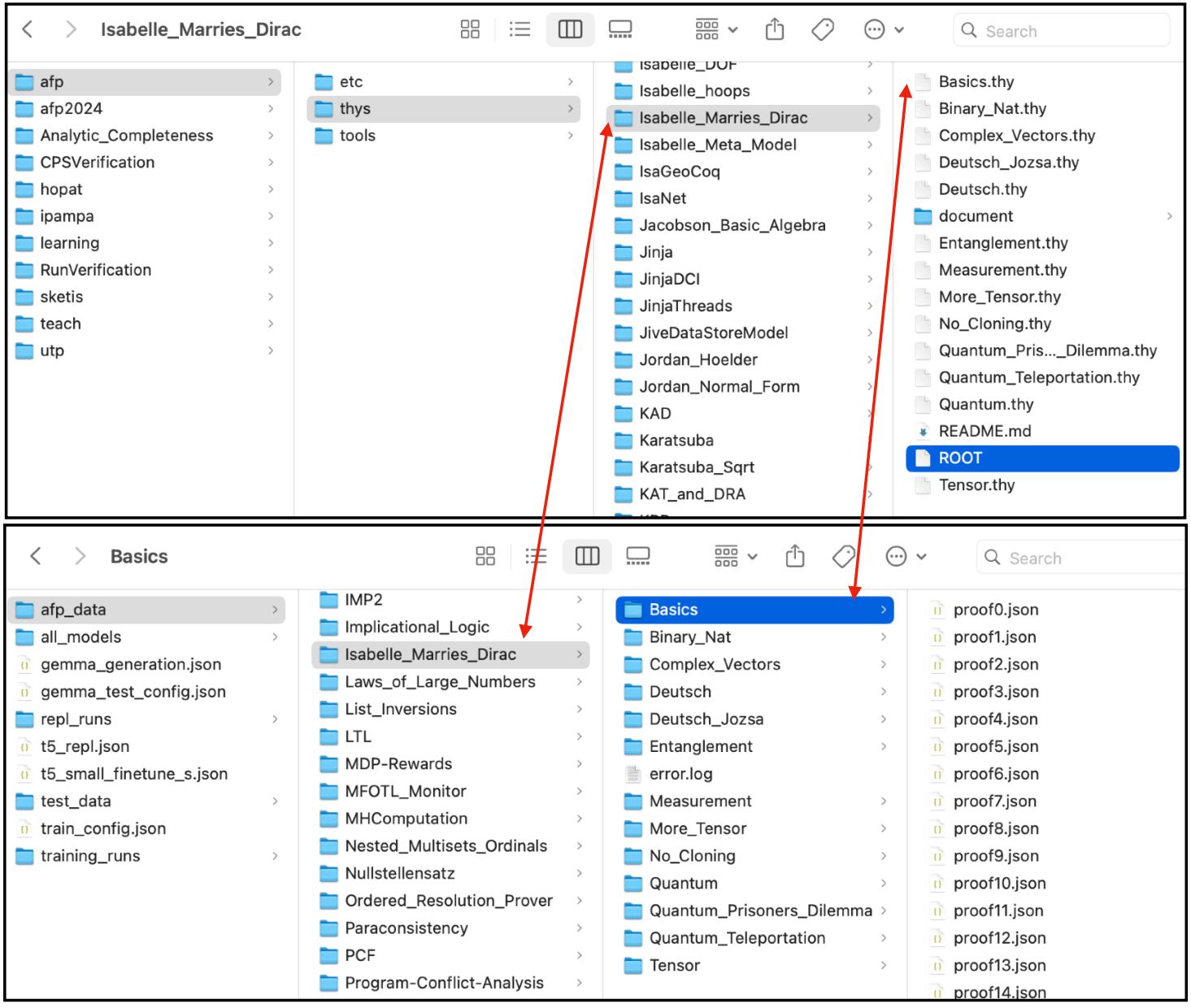
Plan: remove the `scala-isabelle` dependency with Isabelle tooling.

Data extraction process

AALBORG UNIVERSITY

AFP

Generated Data

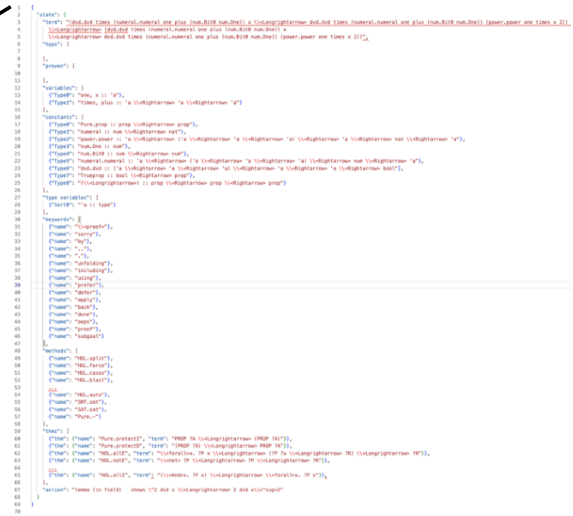


Data extraction process

Identifies each proof in a .thy file

```
(*  Comment with keywords theory, begin, lemma, and by  *)

chapter ‹ Title with keywords theory, begin, lemma, apply and by ›

theory Test_Thy3
  — ‹ A comment with keywords theory, begin, lemma, apply and by ›
  imports Complex_Main

begin

—‹Automatic proofs›

lemma "∀ x. P x ⟹ ∀ x. P x ⟶ Q x ⟹ ∀ x. Q x"
  by auto

—‹Apply-style proofs (with rules)›

lemma "∀ x. P x ⟹ ∀ x. P x ⟶ Q x ⟹ ∀ x. Q x"
  apply(rule allI)
  apply (erule allE)
  apply (erule allE)
  apply (erule impE)
   apply assumption
  apply assumption
  done

—‹Math-like (or structured) proofs›

lemma (in field)
  shows "2 dvd x ⟹ 2 dvd x²"
proof-
  assume "2 dvd x"
  then obtain k where "x = 2 * k"
    by blast
  hence "x² = (2 * k)²"
    by simp
  also have "... = 4 * k²"
    using power_mult_distrib by force
  ultimately have "∃ k. x² = 2 * k"
    by (metis semiring_normalization_rules(18) semiring_normalization_rules(29))
  thus "2 dvd x²"
    by auto
qed

—‹ Theorem commands ›

theorem "∀ x. P x ⟹ ∀ x. P x ⟶ Q x ⟹ ∀ x. Q x"
  by blast

corollary "∀ x. P x ⟹ ∀ x. P x ⟶ Q x ⟹ ∀ x. Q x"
  by blast

proposition "∀ x. P x ⟹ ∀ x. P x ⟶ Q x ⟹ ∀ x. Q x"
  by blast

lemma named_theorem1: "∀ x. P x ⟹ ∀ x. P x ⟶ Q x ⟹ ∀ x. Q x"
  by blast

lemma named_theorem2: "∀ x. R x ⟹ ∀ x. R x ⟶ Q x ⟹ ∀ x. Q x"
  by (rule named_theorem1)

lemma named_theorem3:
  assumes h1: "x < c"
    and h2: "x < c ⟶ S x"
  shows "S x"
  using h1 and h2 by blast
```

```
lemma (in field)
  shows "2 dvd x ⟹ 2 dvd x²"
proof-
  assume "2 dvd x"
  then obtain k where "x = 2 * k"
    by blast
  hence "x² = (2 * k)²"
    by simp
  also have "... = 4 * k²"
    using power_mult_distrib by force
  ultimately have "∃ k. x² = 2 * k"
    by (metis semiring_normalization_rules(18)
        semiring_normalization_rules(29))
  thus "2 dvd x²"
    by auto
qed
```



For each user action in the proof, it writes corresponding data

Content of the generated data

proofN.json (illustrative excerpt)

```
{ "proof": {
  "steps": [{ "step": {
    "action": "apply simp",
    "user_state":
      "goal (1 subgoal):\n
      1. P ==> P",
    "term": "P ==> P",
    "hyps": [], "proven": [],
    "variables": [...],
    "constants": [...],
    "type variables": [...]
  }}, ...],
  "apply_kwrds": ["apply", ...],
  "isar_kwrds": ["have", ...],
  "methods": ["simp", "rule", ...],
  "deps": [...]
}}
```

Per proof step

action	user command
user_state	state before action
term	proof obligation
hyps, proven	local context
variables	names with types
constants	names with types
type variables	names with sorts

Per proof

apply_kwrds	apply commands
isar_kwrds	Isar commands
methods	proof methods
deps	referenced lemmas

Extensible: classes · locales · definitions · conjecturing · Sledgehammer premises

Designed for next-step prediction and proof-search experiments.

REPL

Start the gateway `sbt "runMain isabelle_rl.Py4j_Gateway_Main"`

Python ↔ Py4J ↔ Scala ↔ Isabelle/ML

Python client

```
repl = REPL("HOL")
```

→

```
REPL and minion initialized.
```

```
repl.apply('lemma "∀x. P x ⟹ P c"')
```

→

```
goal (1 subgoal):
1. ∀x. P x ⟹ P c
```

```
repl.apply("apply simp")
```

→

```
goal: No subgoals!
```

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

→

```
""                                    proof closed
```

Isabelle reply

Backtrack, navigate, inspect, or delegate:

10 s timeout per action

`undo()` `reset()` `switch_to()` `go_to()` `call_hammer()` `proof_so_far()`

CICM 2025 tutorial: cicm-conference.org/2025/ml-isabelle/general.html

Experimental setup

AALBORG UNIVERSITY

Models and baselines

T5
T5-small (77M)
T5-base (248M)

Local LLMs
Gemma 3n E2B (fine-tuned)
Gemma 4 E4B

API models
GPT-5.3-chat
Gemini 3 Flash Preview
Gemini 2.5 Flash Lite

Baseline: Sledgehammer

T5 comparison axes

Prediction task
Proof context $\rightarrow$ next action

Input context
`s`: proof prefix + state
`spk`: also premises and keywords

Learning regime
From scratch vs fine-tuned

Prediction metrics
Exact-action accuracy
First-token accuracy

T5 data protocol

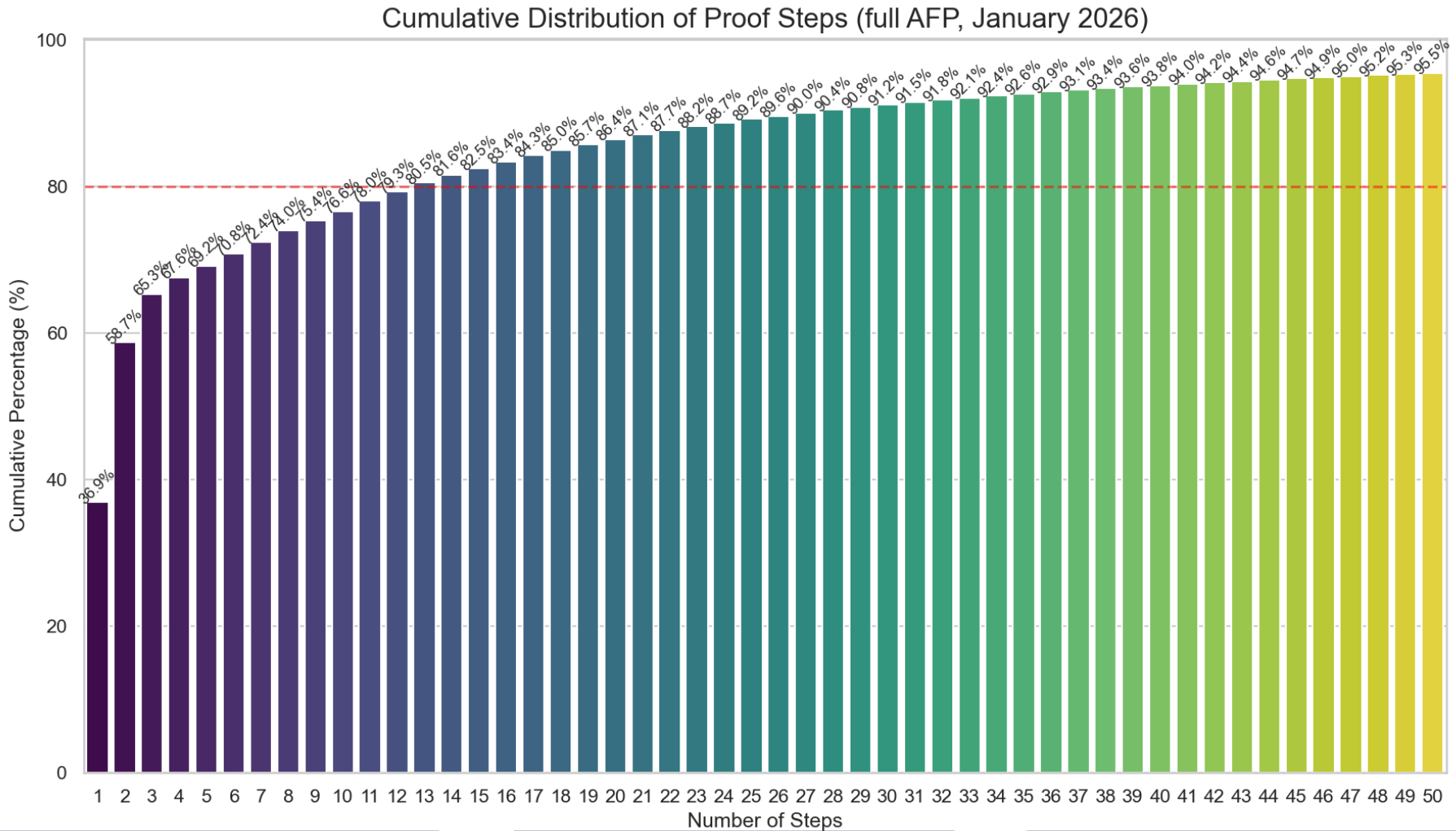
AFP extraction
17 Dec 2024
187,210 proofs

Sequential split per theory
64% train
16% validation
20% test

Maximum input length
512 tokens (`s`)
Up to 1,024 tokens (`spk`)

Training stacks: Hugging Face `accelerate` for T5;
Unsloth + QLoRA for Gemma 3n E2B (2B effective parameters).

Experimental setup



241,126 proofs
January 2026 AFP snapshot

36.9%
finish in one step

68.6%
finish within five steps

All proving experiments used proofs of ≤ 5 steps to bound time and API cost.

Experimental results

Recorded REPL experiments: all evaluated proofs had ≤ 5 steps

Model	Proofs	Predictions	Progress	by cmds	Correct by	Finished	Finished %
T5 models — 17 Dec 2024 AFP split							
small_s	43,538	7,426,213	61.58%	187,683	42.02%	3,103	7.13%
small_finetune_s	42,408	3,508,777	37.79%	162,986	8.40%	4,784	11.28%
small_spk	43,218	2,939,773	48.82%	55,195	14.76%	3,808	8.81%
base_s	43,001	5,955,871	59.45%	101,268	25.36%	4,090	9.51%
base_finetune_spk	38,532	2,315,016	52.57%	118,959	36.70%	6,438	**16.71%**
LLM backends — recorded AFP cohorts							
GPT-5.3-chat	300	1,334	37.03%	164	29.27%	104	**34.67%**
Gemini 3 Flash Preview	1,833	1,351	51.74%	580	68.79%	430	23.46%
Gemma 3n E2B fine-tuned	252	4,697	31.17%	351	15.38%	59	23.41%
Gemini 2.5 Flash Lite	2,807	18,264	19.49%	286	10.14%	55	1.96%
Gemma 4 E4B	1,330	6,191	4.36%	0	—	14	1.05%
Sledgehammer	34,223	32,931	50.18%	—	—	15,832	**46.26%**

ML backends: GPT-5.3 achieved 34.67%; Gemini 3 Flash and fine-tuned Gemma 3n reached ∼23%.

Recorded baseline: Sledgehammer completed **46.26%**, the highest rate in these runs.

Cohorts attempted different proof sets: not a controlled head-to-head ranking.

Later agent studies solved all 15 obligations of a separate Q0 benchmark (see final slides).

Experimental results

AALBORG UNIVERSITY

Fine-tuned Gemma 3n E2B `Fishers_Inequality.Incidence_Matrices`

```
lemma ordered_incidence_sysII:
  assumes "finite_incidence_system 𝒱 ℬ"
      and "set 𝒱ₛ = 𝒱"
      and "distinct 𝒱ₛ"
      and "mset ℬₛ = ℬ"
  shows "ordered_incidence_system 𝒱ₛ ℬₛ"
using assms
apply (unfold_locales,
       simp add: finite_incidence_system_def
                 incidence_system.wellformed)
apply (rule incidence_system.wellformed)
apply (simp_all add: finite_incidence_system_def)
done
```

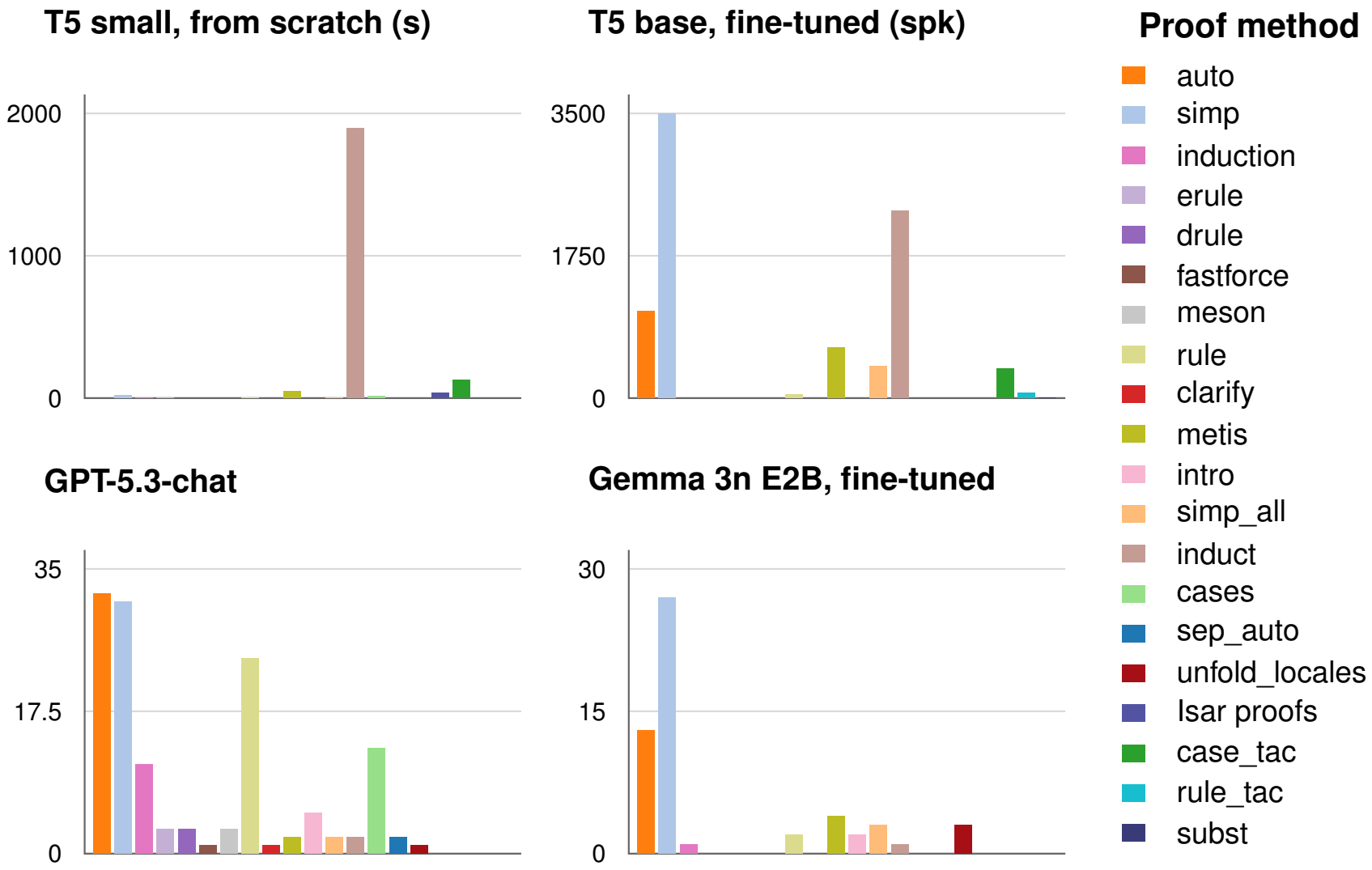
Compact
Four generated proof actions

Locale-aware
Removing `unfold_locales` makes the proof fail

Verified
The complete script is accepted by Isabelle

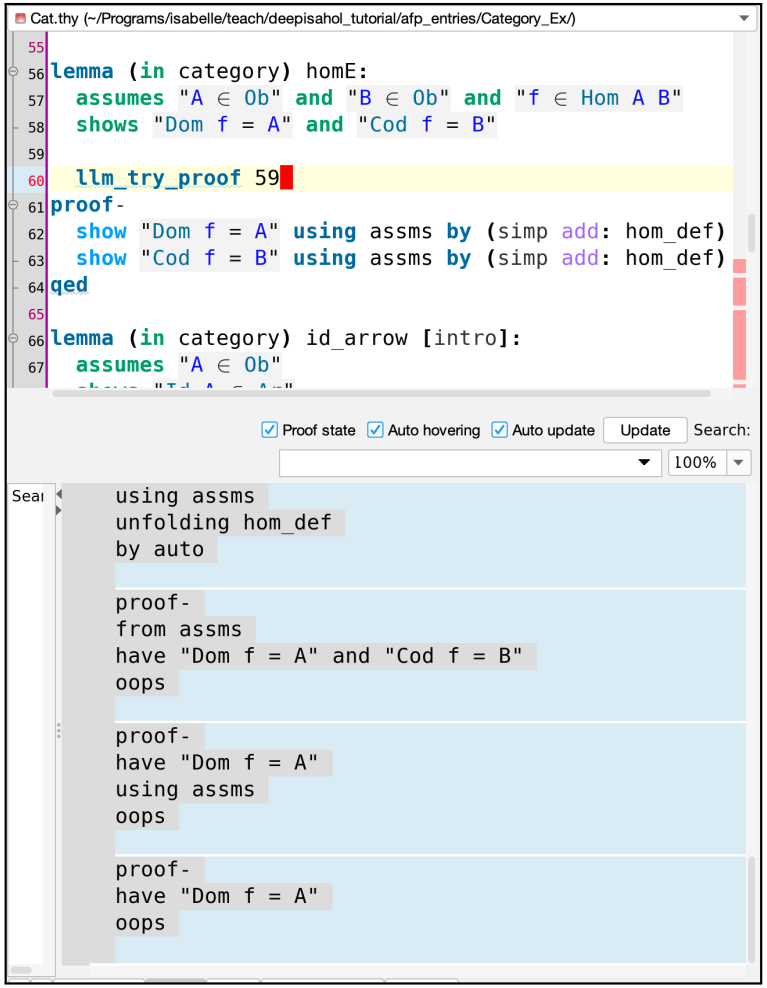
Proof methods analysis

Frequency of proof methods in completed proofs

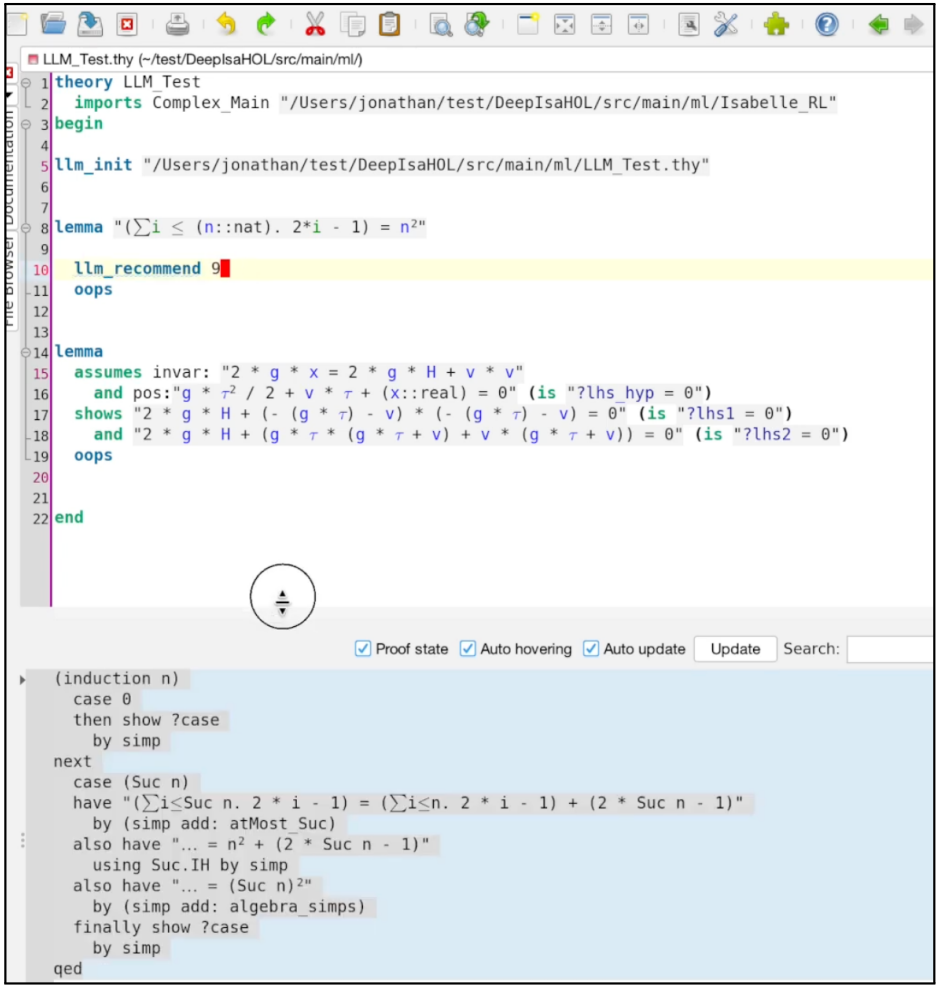


Other work: calling models from Isabelle

Isabelle commands invoke a model and return candidate proofs directly in the editor.



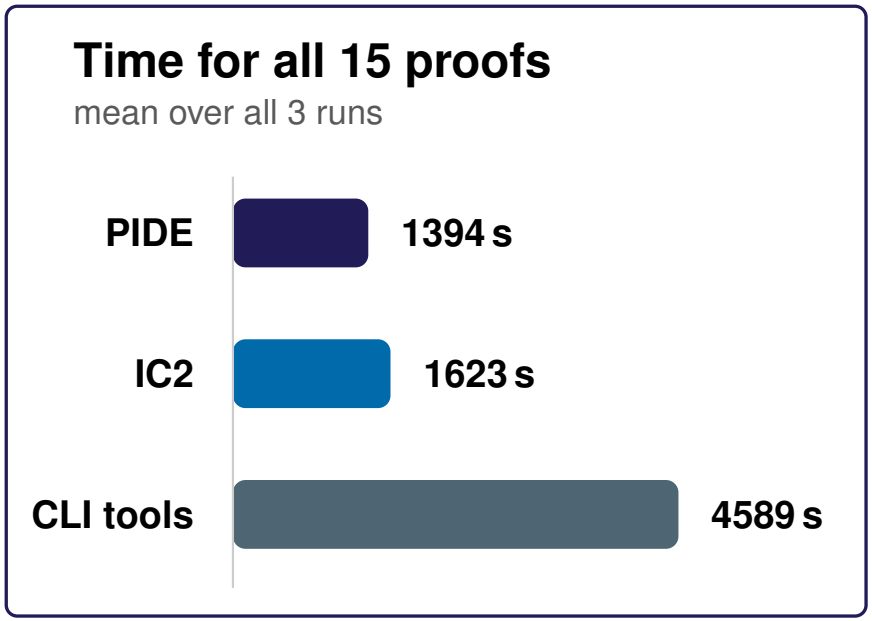
`llm_try_proof`
requests several candidate proof scripts for the current goal

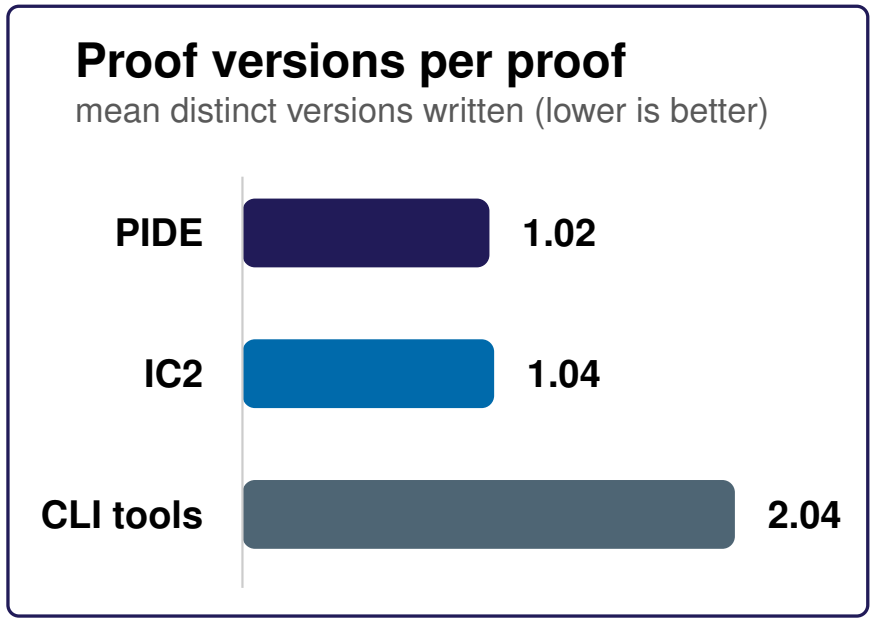


`llm_recommend`
inserts a model recommendation at a selected source position

Future work: agent tools for Isabelle

Community-provided tools (see their respective repos), evaluated in their published versions: speed, iteration, and token cost.
9 runs on 15 obligations of the recently submitted Completeness of Q0,
3 runs per tool, all with Opus 5 via Claude Code.





Mean per run	PIDE	IC2	CLI tools
Generated tokens	96,683	89,069	142,609
Total token cost	$20.06	$22.17	$20.37

Token cost included input and cache usage, not only generated text.

All means include 3 runs per tool.

Earliest detected task-completion (replay was not exhaustive, so timings are upper bounds). The time for the last verification is excluded.

Future work: Agilisa

Summary-prompt: Take the data from the experiment and design a cheaper and faster tool.

Result: **Agilisa** is an MCP/CLI adapter that keeps one Isabelle/ML process alive and executes only the proof commands sent by the agent:

Evaluated Agilisa revision

Interface	Runs	Solve-Time†	Active	Tokens*	Cost/run
Agilisa (median)	5	1,448 s	**1,657 s**	**2.91 M**	**$14.55**
PIDE (same day)	1	1,993 s	2,230 s	3.93 M	$19.65
PIDE MCP	3	1,387 s	1,765 s	3.97 M	$19.85
IC2	3	1,507 s	1,956 s	4.31 M	$21.55
CLI tools	3	5,186 s	5,155 s	4.35 M	$21.75

* Price-weighted input-token equivalents (not raw token counts), at $5/M.

† Elapsed time to confirmed correct task version. Retrospective checks excluded.

Recorded cost: Agilisa's median was ∼27–33% below the other tools' medians. The experiments are ongoing: not claiming a clear advantage of Agilisa yet.

Thank you

Everyone is invited to collaborate, use the tools, and provide feedback.



zenodo.org/records/15080049

Jonathan Julián Huerta y Munive
jjhymuni@cs.aau.dk

Extras: price-weighted input-tokens

A common **cost meric**:
how many ordinary input tokens would cost the same?

Token category	Study price / M	Weight
Ordinary input	$5.00	1
Cache write (1-hour cache)	$10.00	2
Cache read	$0.50	0.1
Generated output	$25.00	5

Weighted tokens = input + $2\times$ cache writes + $0.1\times$ cache reads + $5\times$ output

Estimated cost = weighted tokens $\times$ $\$5/1{,}000{,}000$

Example: 1,000 output tokens + 10,000 cache-read tokens
$= 6{,}000$ input-token equivalents $= \$0.03$.

Pricing fixed for the study (9 September 2026), not guaranteed to be current price.
It is an API-price estimate for reproducing the run and not the actual subscription bill.