OSDI '26

Neuro-Symbolic Proof Generation for Scaling Systems Software Verification

PRESENTER

Baoding He

JOINT WORK WITH

Zenan Li · Wei Sun · Yuan Yao · Taolue Chen
Xiaoxing Ma · Zhendong Su

NANJING UNIVERSITY

ETH zürich

Birkbeck UNIVERSITY OF LONDON

OSDI '26 · 19th USENIX Symposium on Operating Systems Design and Implementation · 2026

Interactive Theorem Proving (ITP) & Formal Verification

ITP-based formal verification



Human-written theorems and proof scripts

Mainstream ITPs

Lean 4 | Isabelle/HOL | Rocq



Guaranteed correctness

Importance

- Eliminate catastrophic bugs in safety-critical systems; offer stronger guarantees beyond testing

Landmark artifacts

- seL4, CompCert, CertiKOS, FSCQ, SeKVM …
- kernels, compilers, filesystems, hypervisors …

Cost of Verification at Scale

- **The proof size is usually larger than the code size**

Project	Lines of code	Lines of proof	Proof to code
CompCert	5.9K	35K	~6.7x
CertiKOS	6.5K	135K	~20.7x
SeKVM	15.5K	32.7K	~2.1x
seL4	10K	200K	~20.0x

Can automated tools reduce this burden
without sacrificing correctness guarantees?

LLM Queries Fail on seL4 Proof Tasks

Human experts: write out the proof steps: tactics (e.g., wp) and lemmas (cap_swap_def)

```
(* When there are no dangling pointers in
the address space of the seL4 microkernel,
for two valid capability slots belonging
to two thread control blocks, swapping
the contents of two capability slots
preserves the absence of dangling
pointers. *)
```

```
lemma cap_swap_valid_objs[wp]:
  "{valid_objs
     and valid_cap c and valid_cap c'
     and tcb_cap_valid c b
     and tcb_cap_valid c' a}
   cap_swap c a c' b
   {λrv. valid_objs}"
```

```
apply (simp add:
cap_swap_def)
```

```
apply (wp set_cap_valid_objs
       | simp split del: if_split)+
done
```

Theorem prover: verify each proof step and propagate (forward) the proof state.

```
proof (prove)
goal (1 subgoal):
 1. {valid_objs
     and valid_cap c and valid_cap c'
     and tcb_cap_valid c b
     and tcb_cap_valid c' a}
    cap_swap c a c' b {λrv.
valid_objs}
```

```
proof (prove)
goal (1 subgoal):
 1. {valid_objs
     and valid_cap c and valid_cap c'
     and tcb_cap_valid c b
     and tcb_cap_valid c' a}
    do y <- set_cap c' a;
       y <- set_cap c b;
       … …
    {λrv. valid_objs}
```

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

Large language model: substitute human experts to automatically generate proofs

```
### Prove the following Isabelle
statement.
### Input:
lemma cap_swap_valid_objs[wp]: … …
### Response:
```

Gemini

```
This lemma comes from the seL4
kernel verification context … …
apply (simp add: cap_swap_def)  ✗
apply wp
apply clarsimp
done
```

```
Failed to finish proof:
goal (7 subgoals): ……
```

GPT

```
The user wants a proof of the lemma,
likely in the seL4 Isabelle/HOL...
apply (rule hoare_seq_ext)  ✗
apply (wp set_cap_valid_objs
  set_cdt_wp do_extended_op_wp
  set_original_wp gets_wp
  | simp add: bind_assoc)+
apply (wp | wpc |
simp add:  bind_assoc)+
```

```
Undefined fact:
"hoare_seq_ext"
```

3-step proof. Asking LLM (Gemini / GPT-5.1): fails — wrong tactic or hallucinated lemma.

Why Is Automated Proof Generation Hard?

Challenge 1 — Lack of Specialized Expertise

- Domain-specific tactics/lemmas unknown to general LLMs
- GPT-4: only 20% success on seL4, even 40% need just 1 line (Selene)
- LLMs hallucinate premises not in the proof context

Challenge 2 — Paucity of Usable Training Data

- seL4: ~20K theorems — too few for fine-tuning
- Proof semantics implicit in prover state — not visible in scripts
- FVEL: <10% success after fine-tuning on all seL4 data

→ Fix: integrate ITP symbolic tools with neural generation.

Approach: Core Idea & Components

Best-First Tree Search
Searches over ITP proof states using global heuristics

Fine-Tuned LLM Guidance
Proposes and ranks candidate proof steps in real time

Symbolic Tools Synergy
Repairs, filters, and discharges failed steps automatically

INPUT

Theorem

Initial proof goal

Best-First Tree Search
over ITP proof states

CORE ENGINE

NEURAL

Step Proposal
LLM generates candidate proof steps

State Ranking
Heuristic ordering of search paths

SYMBOLIC

State Revision
Repairs failed proof steps

State Filtering
Prunes via counterexamples

Hammer
Discharges residual subgoals

VERIFIED

Isabelle Proof

Fully certified

Proof Step Generation & Revision

1) Proof Step Generation (fine-tune an LLM with data amplification)

- Since we focus on single-step prediction: extract state–step pairs by proof replay
- Proof replay: 26K theorems → 181K state–step pairs (~7×)
- Fine-tune two models: Qwen3-1.7B and Mistral-7B

2) Proof Step Revision

- Failed steps can be partially correct (each step is a combo of tactics and premises)
- Tactic repair: if the step is syntactically valid but semantically wrong, extract premises from the failed proof step, recombine with frequent tactic library (simp, auto, blast…)
- Premise repair: if the step cites a hallucinated lemma, compute edit distance between cited name and all names in active context, substitute it with the closest one

Enables proving **24.7%** of theorems where the LLM initially produced zero valid steps

Proof State Filtering and Ranking

1) Proof State Filtering

- Counterexample Detection: QuickCheck/Nitpick perform model-based finite counterexample search; if a counterexample is found, the state is unprovable and pruned
- Duplicate Pruning: model may propose identical proof steps; different proof steps can also reach the same proof obligations; track and prune the duplicate

On a representative theorem: 8,000+ candidate states → **>97% eliminated**
(44% duplicates, 53% unprovable)

2) Proof State Ranking

- Cumulative LLM log-probability along the path, normalized by proof length (L^{α})

Proof state ranking score

$$score(s_L) = \frac{\sum_{t=0}^{L-1} \log p(a_t \mid s_t)}{L^{\alpha}}$$

Following the confidence of LLM

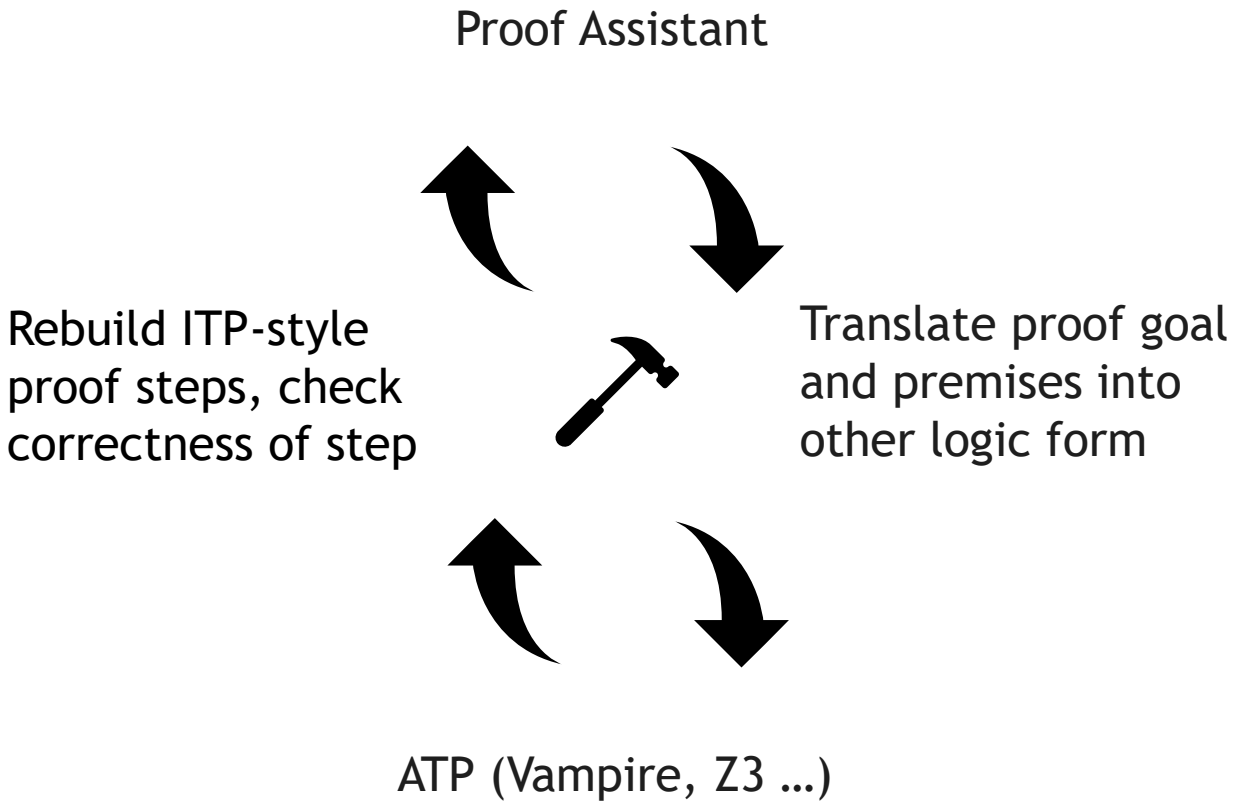
L^{α} normalizes for proof length (default $\alpha = 1$)

What is Sledgehammer?

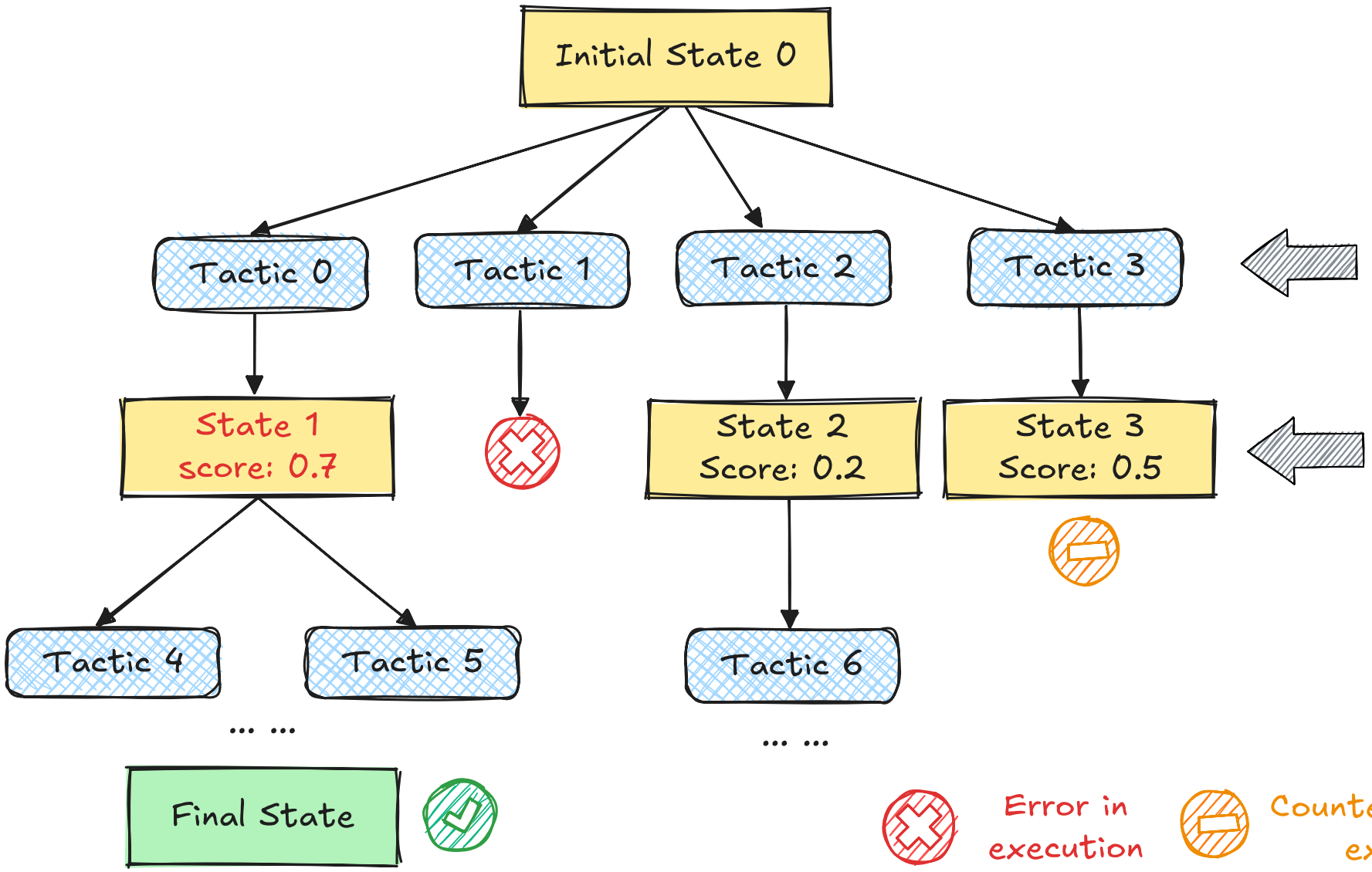
- Isabelle's built-in bridge to external ATP/SMT solvers (E, Vampire, Z3, CVC5…)
- Can discharge proof obligations automatically
- Result is a verified native Isabelle tactic — no trust gap

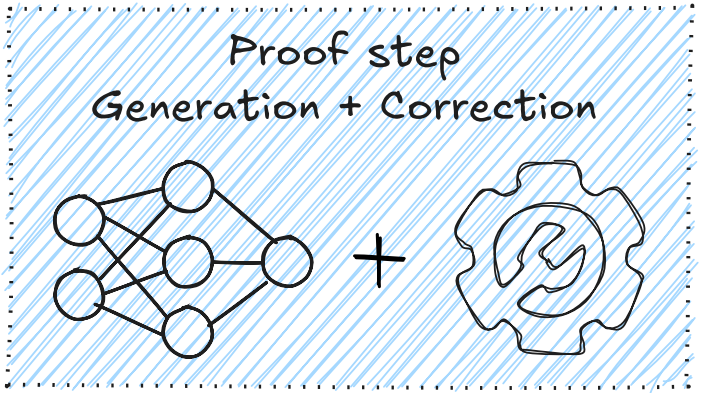
How we use it

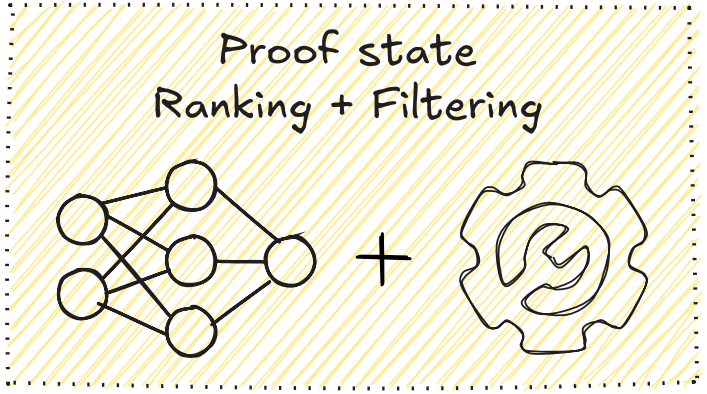
- Invoked only when LLM tree search exhausts its budget
- Take the highest-scoring residual proof states and submit each
- Proof is successful if Hammer discharges residual proof goals of any submitted states



Overall: Neuro-Symbolic Proof Search







Error in execution

Counterexample exists

Done or proved by Sledgehammer

Experimental Setup

Primary benchmark: seL4 (FVELER)

- 29,125 theorems | Train 26K / Val 1,115 / Test 1,077 / Hard 852
- Test-hard: 3 unseen sessions (SysInit, SysInitExamples, LibTest)
- 181,887 state-step pairs via proof replay

Additional benchmarks (generalizability)

- X86 Semantics (86), IEEE Float (188), SATSolver (564) — AFP
- Code2Inv (399) — loop invariants → Isabelle

Execute Isabelle Scripts

- Large complex theory dependencies can heavily degrade compilation efficiency
- For example, in the CRefine theory of seL4, a single proof step takes more than one minute to process
- We implement interactive REPL using a caching mechanism which replaces checked proof segments into sorry, avoiding redundant execution (with other utilities)

Proof Generation Success Rate — FVELER Benchmark (RQ1)

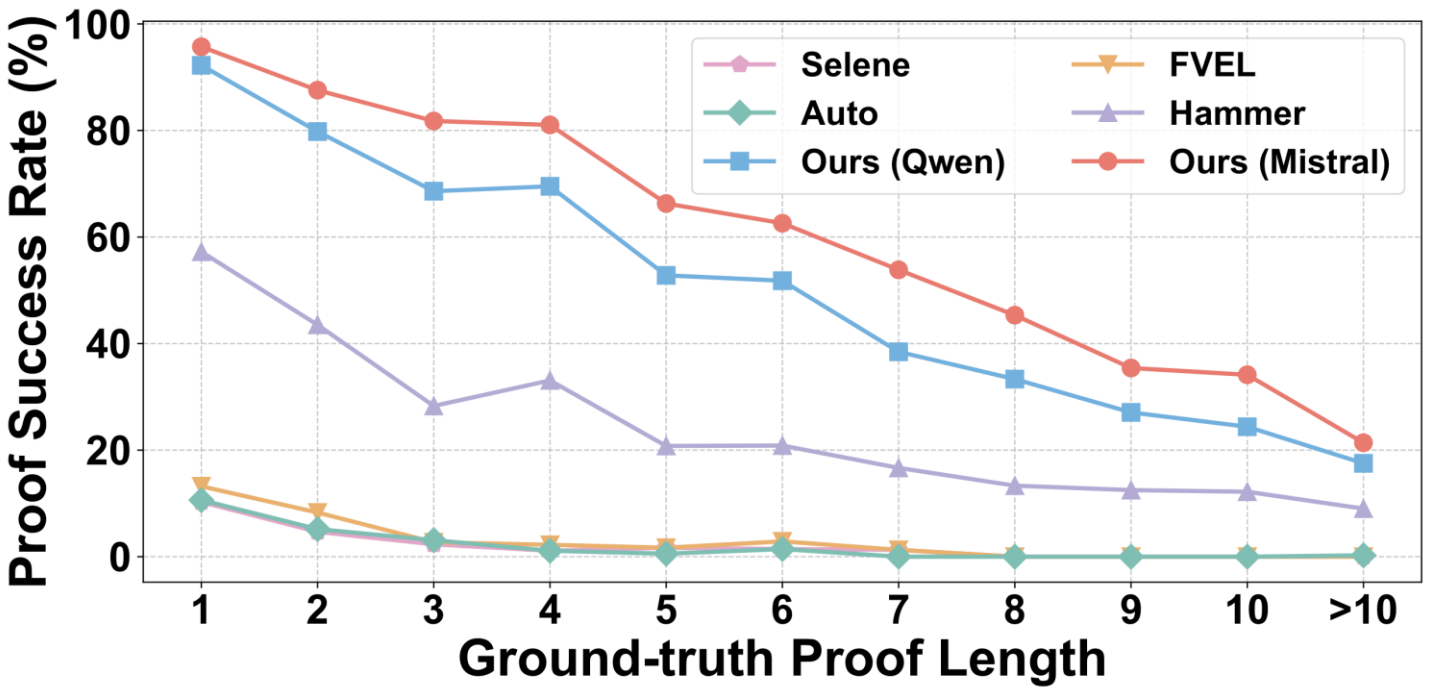
Method	Val	Test	Test-hard	Total
Selene (GPT-4o ICL)	6.1%	7.0%	3.3%	5.6%
FVEL (Mistral-7B)	8.9%	9.5%	4.5%	7.8%
Auto (symbolic)	4.9%	6.7%	6.1%	5.9%
Sledgehammer	40.5%	39.5%	40.9%	40.3%
Ours — Qwen3-1.7B	73.6%	74.9%	61.1%	70.4%
Ours — Mistral-7B	79.8%	89.0%	69.8%	**77.6%**

Success Rates

- Mistral-7B: 77.6% — +37pp vs. Sledgehammer, +70pp vs. FVEL
- Qwen3-1.7B: **70.4%** — small models already effective
- Test-hard (unseen): 69.8% — generalizes beyond training

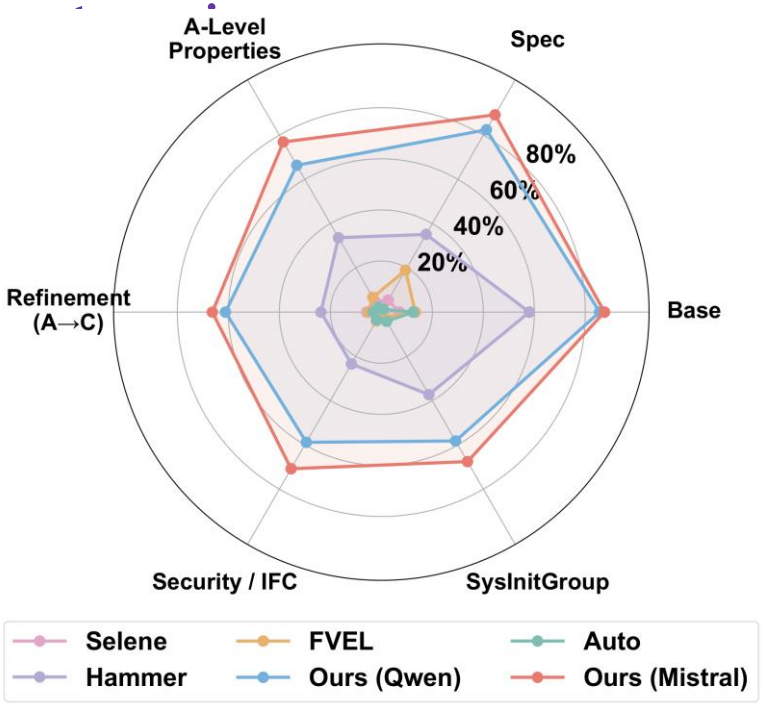
Multi-Step Proofs and Cross-Session Generalization (RQ1)

Proof success vs. proof length



Success rate remains ~20% even for proofs exceeding 10 steps (393 theorems); prior neural methods approach 0% beyond length 4.

Across seL4 session



SysInitGroup is entirely unseen during training; our approach achieves 67.6% on this split.

Reducing Manual Proof Effort (RQ2)

Proof line coverage (automatically generated)

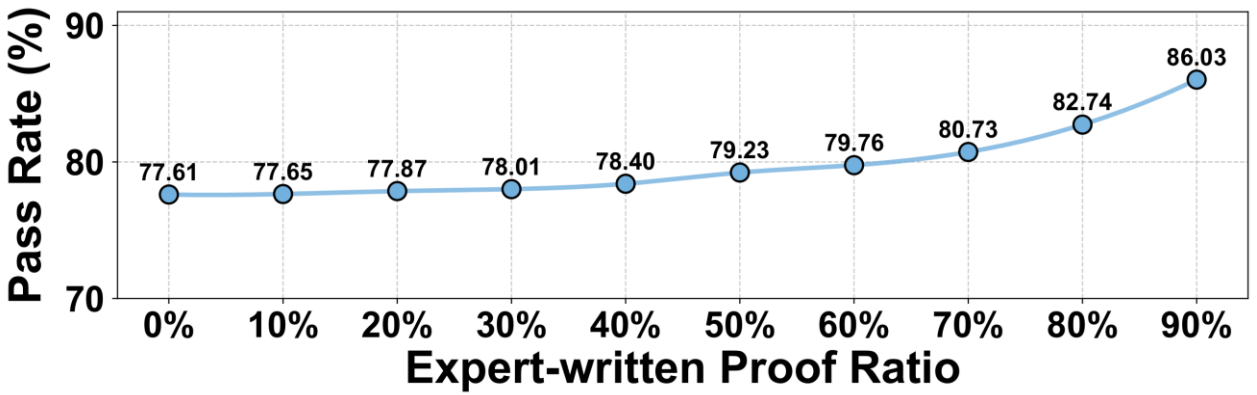
Method	Lines covered	Coverage
Selene	197	1.1%
FVEL	271	1.6%
Sledgehammer	2,581	15.0%
Ours	**6,235**	**36.2%**

AI-human collaboration metric

Supply σ fraction of expert lines → system completes the rest. η (area under curve) = 77.6%

Time Consuming

58% of proofs found in < 10 min; 80% within 2 hours.



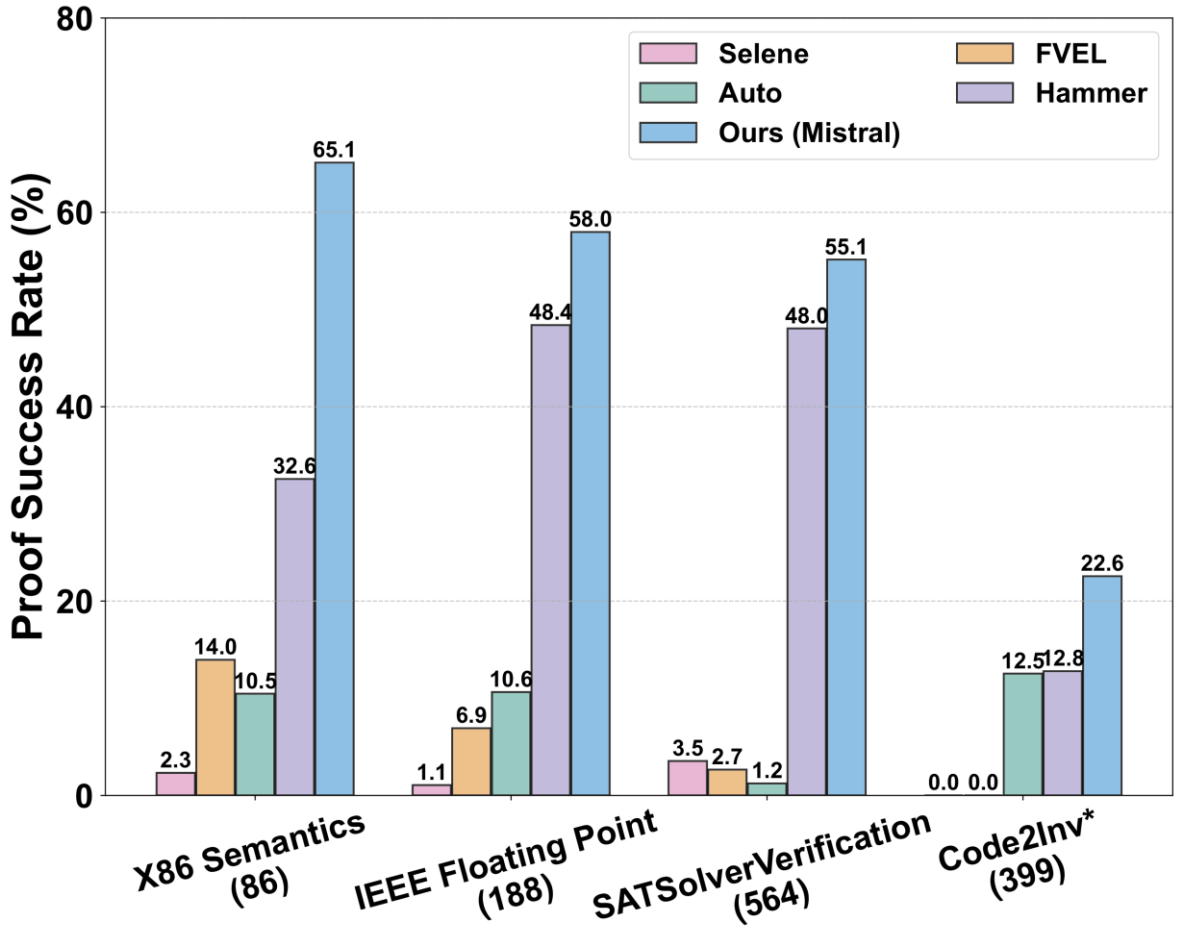
Generalizability to Other Projects (RQ3)

Four benchmarks outside seL4 — no retraining

- X86 Semantics (86 theorems, AFP)
- IEEE Floating Point (188, AFP)
- SATSolverVerification (564, AFP)
- Code2Inv (399) — loop invariants translated to Isabelle

Results

- Leads on all four benchmarks; avg. +36.9% over strongest baseline



Contribution of Components

Off-the-shelf vs. fine-tuned model

Configuration	Success rate
DeepSeek v3.2 (no search)	5.5%
+ Tree search	12.5%
+ RAG + Tree search	33.0%
+ RAG + Tree search + Sledgehammer	52.0%
Ours (SFT + Search + Hammer)	**70.0%**

Effect of supervised fine-tuning

Configuration	Success rate
Mistral-7B (no SFT) + Tree search	0.0%
Qwen3-1.7B (no SFT) + Tree search	0.1%
Mistral-7B SFT + Tree search	**59.8%**
Qwen3-1.7B SFT + Tree search	**57.1%**

SFT + tree search + step revision + filtering + Sledgehammer — essential, work together.

Patterns of Failure

Pattern 1 — Lemma Hallucination

- 61.1% of errors: undefined fact — LLM cites premises absent from context
- 17.9%: tactic inapplicable — together, 79% of wasted search budget

Pattern 2 — Search Budget Exhaustion

- 72% of failed runs reach depth >30 — exhausted budget on residual goals
- Sledgehammer also cannot close these residual goals

692 theorems (22.4%) remain unsolved. Two recurring patterns emerge.

Possible solutions: premise grounding and retrieval, longer-horizon planning.

Summary

Problem

Constructing proofs for large-scale verified systems (e.g., seL4) is highly manual — 20 person-years of expert effort for one project. Direct LLM querying does not close this gap.

Approach

Neuro-symbolic proof search: fine-tuned LLMs propose candidate steps; symbolic tools filter unprovable states; Sledgehammer closes residual subgoals.

Key results

77.6% proof success on FVELER (seL4); +37 pp over Sledgehammer. Generalizes to unseen seL4 sessions (67.6%) and benchmarks constructed from three AFP projects and Code2Inv.

Thank you

Q&A

Backup Slides

Proof Search Algorithm (Pseudocode)


```
Require: initial state s0, LLM G, checker C, max iters Imax
init tree T with root s0
for i in [1, Imax]:
    n = highest_score_unexplored(T)
    succ, fail = [], []
    for step in G(n.state):           # LLM step generation
        s = apply(step, n.state)      # ITP execution
        if s is error:  fail += [(s, step, lp)]
        else:           succ += [(s, step, lp)]
    for (s,d,lp) in REVISE(fail):   # symbolic revision
        succ += [(apply(d, n.state), d, lp)]
    for (s, d, lp) in succ:
        if no subgoals:  return proof
        if C(s) detects duplicate or counterexample:  skip
        score = combine(n.score, lp)
        insert_node(T, (s, score))
return FAIL
```

Key hyperparameters

- 128 candidate steps generated per state
- Top-5 states expanded per iteration
- 16 states passed to Sledgehammer on failure
- 120 min global time limit per theorem
- $\alpha = 1$ for length normalisation
- LLaMA-Factory + DeepSpeed ZeRO-2
- Full-parameter SFT, 3 epochs, LR 1e-5

Step-Level Error Breakdown

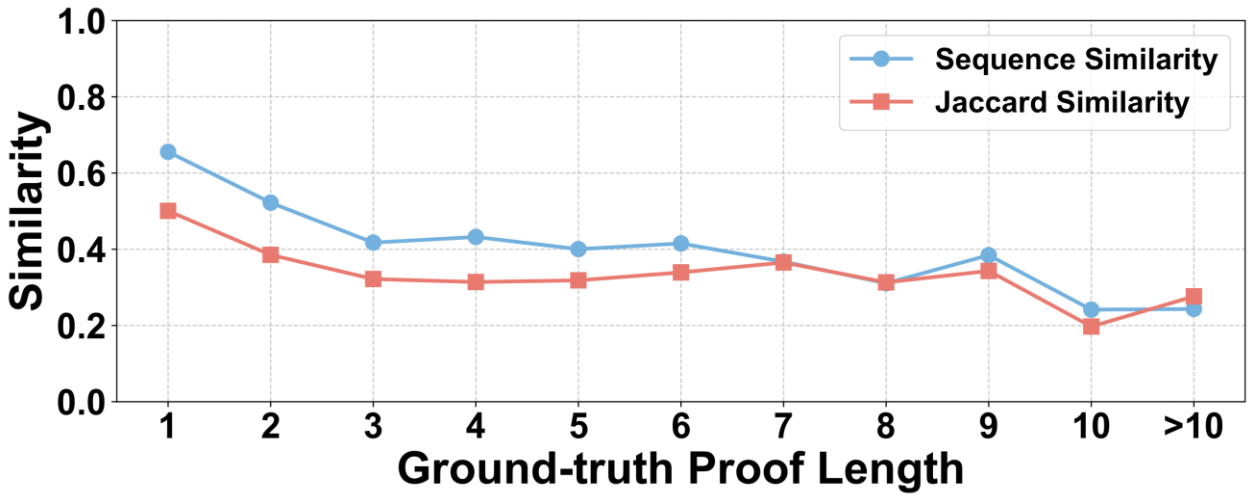
7.77M tactic attempts across 241 failed theorems (validation split).

Error category	Count	%
Undefined fact	3,263,494	**60.1**
Tactic inapplicable	976,398	17.9
Multi-line / other	423,982	7.8
Unknown variable	161,842	2.9
Outer syntax error	153,037	2.8
Type unification	110,717	2.0
Tactic timeout	102,099	1.8
Bad method args.	65,430	1.2
Inner syntax error	50,033	0.9

Undefined fact alone accounts for 60% of all errors. The model most often cites premises that do not exist in the active proof context.

Together with tactic-inapplicable errors (shape mismatch), these two categories explain roughly 79% of all wasted budget.

Generated Proofs vs. Ground-Truth Proofs



Similarity analysis

- Both sequence similarity and Jaccard similarity remain low and decrease as proof length grows
- If the model were memorising proofs, similarity would stay high regardless of length
- Among one- and two-line proofs (the smallest solution space), only 16.7% are identical to the ground truth
- The observed trend suggests the model synthesises novel proof strategies rather than recalling training examples