VILNIUS UNIVERSITY

FACULTY OF MATHEMATICS AND INFORMATICS

INFORMATICS STUDY PROGRAMME

Master's thesis

Analysis of artificial neural networks for solving reasoning tasks

Loginių samprotavimų uždavinių sprendimo, naudojant dirbtinius neuroninius tinklus, galimybių tyrimas

Author : Belkacem Sadi

Supervisor : Doc. Dr. Haroldas Giedra

Reviewer : Dr. Dalius Gudeika

Vilnius
2026

Acknowledgements

I want to thank Doc. Dr. Haroldas Giedra for his supervision and guidance throughout all four semesters of this research. His advice and ideas were incorporated in all stages of the work, and his continuous support was instrumental to the completion of this thesis.

The author would also like to express his gratitude to Vilnius University Information Technology Research Centre for providing high-performance computing facilities. This thesis describes all experiments which have been performed on the Vilnius University HPC cluster MIF with the NVIDIA Tesla V100-SXM2-32GB GPU accelerator cards, without which running these extensive experiments would not have been possible.

Abstract

This thesis examines the use of artificial neural networks for automated theorem proving in the Lean 4 formal proof assistant. The central problem is semantic hallucination: models often produce proof steps that are syntactically correct but logically incorrect. A formal error probability framework is presented to quantify this problem at the step level.

Phase I evaluates three models under identical conditions on 1,000 theorems from the LeanDojo Benchmark 4. InternLM2.5-StepProver achieves 45.7% Pass@1, ReProver 44.6%, and DeepSeek-Prover-v1.5-RL 40.3%. The framework shows that 75–83% of syntactically valid tactics are logically incorrect across all models.

Phase II proposes two modifications of InternLM2.5-StepProver. Proposition A introduces a feature-based value head that prunes unpromising proof branches, achieving 51.9% Pass@1 (+6.2pp). Proposition B trains an error-conditioned refiner using Direct Preference Optimization on real Lean 4 compiler error pairs, achieving 45.9% Pass@1 with a 16.3% correction rate.

Keywords: artificial neural networks, theorem proving, Lean 4, beam search, semantic hallucination, value head, Direct Preference Optimization.

Santrauka

Šiame darbe tiriamas dirbtinių neuroninių tinklų taikymas automatiniam teoremos įrodymui Lean 4 formalių įrodymų asistente. Pagrindinė problema yra semantinė haliucinacija: modeliai dažnai generuoja sintaktiškai teisingus, tačiau logiškai netinkamus įrodymo žingsnius. Pristatoma formali klaidų tikimybės sistema šiai problemai matuoti.

I etape trys modeliai įvertinami identiškomis sąlygomis su 1 000 teoremų iš LeanDojo Benchmark 4. InternLM2.5-StepProver pasiekia 45,7% Pass@1, ReProver – 44,6%, DeepSeekProver-v1.5 RL – 40,3%. Nustatyta, kad 75–83% sintaktiškai teisingų taktikų yra logiškai neteisingos. II etape siūlomos dvi InternLM2.5-StepProver modifikacijos. A propozicija prideda bružiais pagrįstą vertės galvutę, genėjančią mažai perspektyvias būsenas, ir pasiekia 51,9% Pass@1 (+6,2 proc. punkto). B propozicija apmokymo metu naudoja tiesioginių pirmenybių optimizaciją su realiomis Lean 4 kompiliatoriaus klaidų poromis, pasiekdama 45,9% Pass@1 su 16,3% taisymo dažniu.

Raktiniai žodžiai: dirbtiniai neuroniniai tinklai, teoremos įrodymas, Lean 4, spindulių paieška, semantinė haliucinacija, vertės galvutė, tiesioginių pirmenybių optimizacija.

List of Tables

Contents

List of Figures

Abbreviations

AdamW. Adam with Weight Decay

ANN. Artificial Neural Network

ATP. Automated Theorem Proving

DPO. Direct Preference Optimization

GPU. Graphics Processing Unit

HPC. High-Performance Computing

LLM. Large Language Model

LoRA. Low-Rank Adaptation

MCTS. Monte Carlo Tree Search

MLP. Multi-Layer Perceptron

MSE. Mean Squared Error

RL. Reinforcement Learning

SFT. Supervised Fine-Tuning

SLM. Small Language Model

SLURM. Simple Linux Utility for Resource Management

TDI. Tactic Diversity Index

TTP. Time-to-Proof

VRAM. Video Random Access Memory

1 Introduction

ANNs are instrumental in influencing the advancements of modern AI. They exhibit outstanding performance in tasks like pattern recognition, text generation, and mathematical reasoning. In addition to this, the activity of science and engineering is the proof of mathematical theories. A theorem is a statement which can be proven to be true through a series of logical steps. These proofs have been traditionally written by hand and examined by other humans for many centuries, not always in correct form. With the advent of interactive proof assistants, proof has now been set up in a formal language with complete logical accuracy. It is still hard to use ANNs for mathematical theorem proving effectively due to limitations in generalising across different proof structures [4]. Lean 4 is a leading proof assistant. It attempts to test the validity of every step in a proof by its own type-checking kernel. It accepts a tactic only if it is syntactically correct and logically valid at this point in the proof. The Mathlib library for Lean 4 contains more than 100,000 formally verified theorems, which is the largest library of machine-checked mathematics available today.

Lean's syntax, its tactics, and the best order of steps in each goal takes time and skill to master in writing formal proofs. Due to this challenge, efforts have been made to write proofs automatically using a neural network trained with a set of existing formal proofs, which then suggests the next step in the proof based on the proof's current state. The formal proof assistants require very strict conditions. The result of any tactic evaluation is binary: it can either advance or refuse the proof. There are two reasons why tactics can be rejected. Either it is syntactically malformed, meaning it is not in the Lean 4 syntax, or it is semantically invalid, meaning it is syntactically acceptable but has no logical validity at the current position of the proof. The second type of error is called semantic hallucination. The main aim of this thesis is to reduce and measure this problem.

Aim

This research aims to explore the ability of ANNs to prove theorems. It checks the performance of ANN models in this area by identifying the strengths and weaknesses of these models to enhance and modify an existing model for the proving task. This research also integrates the ANN model and Lean 4 to guarantee that any errors produced by this model are detected. It aims to understand their ability to perform theorem proving and to propose ways by which the ANNs can enhance their reasoning ability.

Objectives

The research has the following main goals:

1. To identify, analyse and evaluate existing datasets for storing mathematical theorems, proofs and reasoning data.

2. To identify, assess and choose a formal proof-checking tool that can verify the logical consisten-

cy of theorems produced by the ANN to ensure logical reliability within an accepted or rejected framework.

3. To test several ANN models under the same test sets and conditions to identify which ones work well to prove theorems, and which ones do not.
4. To calculate the probability of errors in proof steps generated by ANNs, separating syntactic errors and semantic hallucinations.
5. To suggest ways to further strengthen the performance and reliability of ANNs for theorem proving tasks based on the evaluation findings.
6. To benchmark the performance of the improved model against the baseline and other models considered.

Novelty

This thesis provides a standardised comparison between three neural theorem-proving models for the first time in a controlled environment. Three models of varying neural complexity are evaluated in the same controlled environment, which, to the best of the author's knowledge, was not performed before. The current research also proposes a formal error probability framework to measure error at the step level, which has not been reported systematically before. It also presents a value-augmented architecture for addressing backbone representation invariance in the proof-state solvability estimation. Lastly, it creates Lean 4 compiler error and correction pairs to train the DPO with real compiler error pairs rather than synthetic ones.

Thesis Outline

The thesis is divided into the following chapters. Chapter 2 provides a survey of existing literature on automated theorem proving, including datasets, neural models, and proof assistants, and the choices made for this research. Chapter 3 describes the methodology based on the experimental design, data preparation, evaluation protocol and infrastructure. Chapter 4 presents the Phase I comparative results. Chapter 5 describes the architectural changes designed in Phase II. Chapter 6 discusses the results in the context of published literature and the key findings of this research. Chapter 7 presents the Phase II evaluation results and analysis with advanced metrics. Finally, Chapter 8 summarises the key findings, identifies limitations, and provides future work directions.

2 Literature Review

2.1 Introduction

This chapter provides a summary of the previous research that is pertinent to this thesis. It describes ATP, the available datasets for training and assessment, the dominant ANN-based theorem-proving models, and the formal proof assistants employed in ATP. For each subsection, the choice made for this thesis and the reasons for that choice are presented at the end.

2.2 Historical Development

Computers have been employed over the past sixty years to establish mathematical theorems. In 1956 Newell and Simon released the Logic Theorist, which was very impressive at the time, solving 38 out of 52 theorems from a well-known mathematics book [12]. The use of logic-based search algorithms in the 1960s and 1970s took great strides in the field. The Davis-Putnam algorithm [2] is an example of early advances. These approaches were applicable to first-order logic but were not very helpful with more complex mathematical structures. The first interactive proof assistants, Isabelle, HOL, and Coq were introduced in the decades of the 80s and 90s. These interactive proof assistants provided users with the ability to construct proofs step by step with verification from the underlying engine of every step. With the advent of such techniques, formal verification of significant mathematical theories became possible. The verification of Four Colour Theorem by Gonthier in 2008 [3] is one of the significant accomplishments in this domain.

The area of machine-learning oriented theorem proving is quite recent. Yang and Deng [7] developed the dataset CoqGym comprising of over 71,000 proof states and demonstrated that tactics can be generated through neural networks from the proof states. Polu and Sutskever demonstrated that LLMs can be trained on formal mathematics and generate novel theorems accepted by the Metamath system [13]. Yang et al. [8] introduced LeanDojo in 2023, which provides a comprehensive framework for training and evaluating neural models in Lean 4. This is the underlying structure on which the present thesis is built.

2.3 Datasets

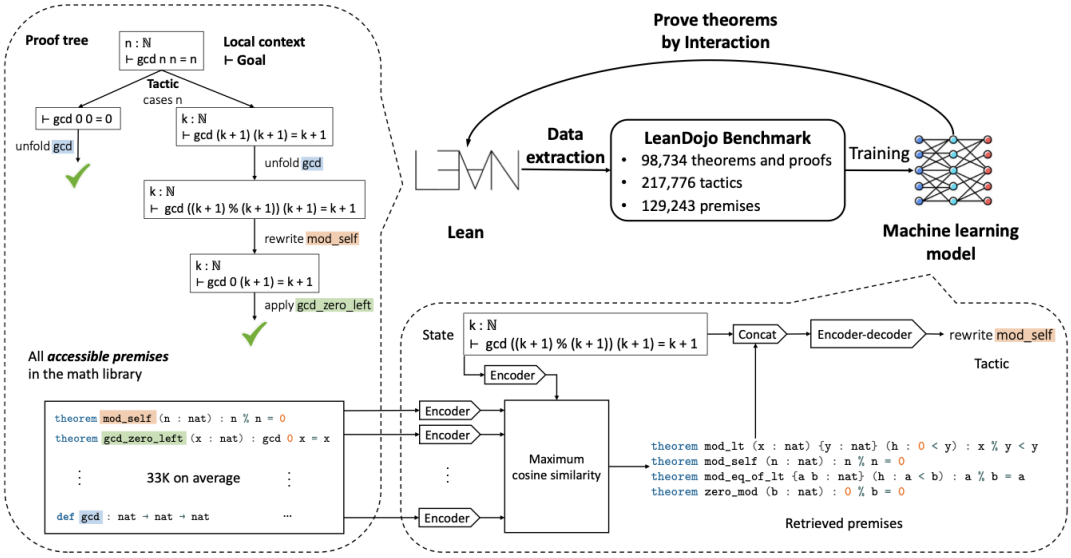
The selection of an adequate dataset is essential in the study of theorem proving. To evaluate a model against its proof patterns and to learn correct ones, the dataset must provide formally verified steps in the proof.

CoqGym [7]

Contains 71,000 proofs from 123 Coq projects. The proof state is given before and after each step. It was the first large step-supervised Coq dataset to facilitate early neural theorem-proving research. This dataset is limited to the Coq environment.

LeanDojo Benchmark 4 [8]

Comprises 98,734 pairs of theorems and their respective proofs from the Mathlib library of Lean 4. All steps are recorded and correspond to a proof state. It has a fixed train/validation/test split for fair comparison. The Lean 4 kernel can verify each predicted tactic in real time. The training split has 118,517 fully traced proofs. This dataset is uniquely suited for step-level neural evaluation.



1 Figure *The LeanDojo framework and ReProver architecture. Left: the proof tree showing how tactics are applied step by step to reach* `ProofFinished`*. Right: LeanDojo extracts theorems and tactics from Lean 4 to train a retrieval-augmented model that retrieves relevant premises and predicts the next tactic [8].*

NaturalProofs [19]

Includes an estimated 20,000 natural-language mathematical proofs. It covers many mathematics topics but does not have formal verification, so it is not useful for step-level error analysis.

TPTP [18]

Uses around 25,000 first-order logic problems. It serves as the standard benchmark for classical automated theorem provers like Vampire and E. Its format differs from Lean's tactic language and does not support step-level neural training.

ProofWiki [14]

Contains more than 28,000 natural-language proofs. It contributes to NaturalProofs but also lacks formal verification.

Selected dataset: LeanDojo Benchmark 4 [8]. It provides (1) formally verified, step-level supervision and (2) real-time interaction with the Lean 4 kernel. These properties are essential for this thesis. Other datasets lack formal verification (NaturalProofs, ProofWiki) or use different proof systems (CoqGym, TPTP).

1 Table *Comparison of Theorem-Proving Datasets*

Dataset	Language	Size	Verified?	Step-Level?	Reference
CoqGym	Coq	71k proofs	Yes	Yes	[7]
LeanDojo B4	Lean 4	98k theorems	Yes	Yes	[8]
NaturalProofs	None	∼20k entries	No	No	[19]
TPTP	Vampire/E	∼25k problems	Yes	No	[18]
ProofWiki	None	28k+ entries	No	No	[14]

2.4 ANN Models for Theorem Proving

ANN models for theorem proving differ in their learning paradigms, supervision sources, and proof assistants they target.

Transformer Architecture

Modern neural theorem provers are built on the transformer architecture [8, 21, 22]. Transformers process input as a sequence of tokens and use an attention mechanism to determine the relevance of each token relative to others. In theorem proving, the input sequence is the proof state — a textual description of the remaining goals and available hypotheses. The self-attention mechanism connects each part of the proof state with every other part, which is useful for predicting proof steps. Transformer architectures therefore have an advantage over recurrent architectures, which suffered from long-range dependency problems. Large language models such as InternLM2.5-StepProver [21] and DeepSeek-Prover-v1.5-RL [22] are pre-trained on large corpora of formal text and code and then fine-tuned on formal mathematical data. This approach allows the models to develop a general understanding of mathematical language before specialising in formal proof generation.

GPT-f [13]

A transformer model fine-tuned on the Metamath library. It is the first system to demonstrate that an LLM can generate new formal theorems accepted by a proof assistant. On complex number theory problems, the error rate is 12.8%, but it only works in the Metamath system.

ReProver [8]

A Lean 4 tactic prover combining neural tactic generation with retrieval of useful lemmas from Mathlib. This hybrid approach supplies the model with relevant mathematical facts beyond what is stored in its parameters. ReProver achieves 26.5% Pass@1 on miniF2F [23].

DeepSeek-Prover-v1.5-RL [22]

Trained with reinforcement learning from Lean 4 feedback and Monte-Carlo Tree Search. It achieves 63.5% Pass@1 on miniF2F-test with RMaxTS enabled [22, 23].

InternLM2.5-StepProver [21]

A large language model fine-tuned on step-level Lean 4 data. For each proof state, it predicts the next correct tactic. It achieves 65.9% Pass@1 on miniF2F [23].

ASTactic [7]

Works in Coq and generates tactics as abstract syntax trees. It is trained on CoqGym and achieves 41.7% proof completion.

HOList [1]

A reinforcement learning environment for HOL Light using graph neural networks for theorem embeddings. It reports a 38.2% success rate on HOL Light library problems.

TacticZero [20]

Applies deep reinforcement learning in HOL4, outperforming heuristic provers by 28% on unseen theorems.

***2 Table** Comparison of ANN Models for Theorem Proving (values from original papers on their respective benchmarks)*

Model	Type	Training	Reported Result	Prover	Ref.
GPT-f	LLM	Supervised	12.8% error	Metamath	[13]
ReProver	LLM	Supervised+Retrieval	26.5% Pass@1	Lean 4	[8]
DeepSeek-v1.5-RL	LLM	RL from feedback	63.5% Pass@1	Lean 4	[22]
InternLM2.5-SP	LLM	Step-level SFT	65.9% Pass@1	Lean 4	[21]
ASTactic	SLM	Supervised	41.7% compl.	Coq	[7]
HOList	SLM	RL	38.2% success	HOL Light	[1]
TacticZero	SLM	Deep RL	+28% vs heur.	HOL4	[20]

Selected models for Phase I: DeepSeek-Prover-v1.5-RL, ReProver and InternLM2.5-StepProver [8, 21, 22]. These were selected because they all run in Lean 4 and can be tested under the same conditions using LeanDojo Benchmark 4. Models supporting other proof systems cannot be meaningfully compared due to differences in environment, library, and interaction protocol. The selected models represent three distinct training philosophies: reinforcement learning, retrieval augmentation, and supervised step-learning.

Note: The quantitative results from this thesis (45.7%, 44.6%, 40.3%) are presented in Chapter 4 under the unified evaluation conditions of this research. These differ from the paper values above because the benchmarks and evaluation protocols are different.

2.5 Formal Proof Assistants

A formal proof assistant is a computer program that checks every step of a mathematical proof. The choice of proof assistant determines the proof language, available libraries, and how neural models can interact with the system.

Lean 4 [10]

Based on dependent type theory with a fast kernel that automatically checks every tactic step. It includes the world's largest library of formally verified mathematics, covering algebra, analysis, category theory, topology, number theory, and more. The LeanDojo framework [8] provides real-time tactic execution via a Python interface, making large-scale evaluation feasible.

Coq [9]

Based on the Calculus of Inductive Constructions. It has been applied in formalising major results such as the Four Colour Theorem [3] and the Feit-Thompson Theorem. CoqGym [7] provides ML integration, but without the same level of standardised tooling as Lean.

Isabelle/HOL

Implemented in classical higher-order logic, Isabelle/HOL is written in Standard ML and Scala. Its Isar language supports human-readable structured proofs, and Sledgehammer enables ATP integration. The Archive of Formal Proofs hosts over 500 verified projects. Nagashima and He [11] present PaMpeR, a ML proof method recommendation system for Isabelle/HOL that correctly predicts experienced users' tactic choices, particularly for special-purpose proof methods.

HOL Light

A lightweight higher-order logic system with a small, trusted kernel. It has been applied to major formalisation projects such as the Kepler Conjecture. Machine-learning integration is possible but limited.

HOL4

Another higher-order logic system widely used for hardware verification research. It is the environment for TacticZero [20].

Selected proof assistant: Lean 4 [10]. This selection was made for three reasons. First, the LeanDojo framework provides a stable Python interface for real-time interaction with Lean 4, which is essential for the beam search evaluation protocol. Second, Lean 4's Mathlib library is the largest formally verified library available. Third, all three models used in Phase I support Lean 4, allowing a common evaluation environment. No other proof assistant satisfies all three criteria simultaneously.

3 Table *Comparison of Formal Proof Assistants*

System	Logic	ANN Integration	Library
Lean 4	Dependent Type Theory	High via LeanDojo	Mathlib (large)
Coq	Inductive Type Theory	Moderate via CoqGym	MathComp
Isabelle/HOL	Higher-Order Logic	Low-Moderate	AFP (large)
HOL Light	Classical HOL	Low	Small
HOL4	Classical HOL	Moderate	Medium

2.6 Analysis of Challenges and Gaps

While there have been significant recent developments in the use of ANNs in mathematical theorem proving, persistent challenges highlight the profound differences between machine and human proof strategies.

Lack of fair comparison. Comparing models in existing studies is not reliable as they are tested in varying environments. For instance, ASTactic, HOList, and ReProver are tested in Coq, HOL Light, and Lean respectively. There are significant variations that can be attributed to the environment rather than the model [8]. This thesis addresses this by providing a standardised environment for all models.

Semantic hallucination is not properly measured. Models can produce proof steps that are syntactically correct but not logically correct. Polu and Sutskever [13] report a 12.8% error rate on complex number theory problems in Metamath when executing autoregressive generation patterns, but prior works does not distinguished between syntactic and semantic errors. This thesis introduces a formal error-probability framework to analyse hallucinations in detail.

Inefficient proof search. In traditional beam search, all branches, regardless of their promise, are treated equally. Systems that prioritise high-value proof states have been shown to achieve faster proof completion in related work. This thesis proposes a value head that estimates solvability and eliminates low-value states.

No error recovery. Current systems tend to ignore the kernel's error message; valuable information is lost when a tactic fails. This thesis presents an error-conditioned refiner that uses these error messages to propose corrected tactics.

2.7 Summary

This chapter introduced the history of automated theorem proving, prominent datasets, data-driven theorem-proving models based on ANNs, and the primary proof assistants. LeanDojo Benchmark 4 is selected for its step-level supervision and real-time Lean 4 interaction. DeepSeek-Prover-v1.5-RL, ReProver, and InternLM2.5-StepProver were selected based on their Lean 4 support and distinct modelling paradigms. Lean 4 was chosen for its LeanDojo integration, large Mathlib library, and fast verification. Four main challenges are identified: lack of fair comparison, insufficient measurement of semantic hallucination, inefficient proof search, and no error recovery mechanism. These challenges are directly addressed in the chapters that follow.

3 Methodology

3.1 Research Design

The study is experimental in nature and consists of two phases. One of the major drawbacks in existing research on formal theorem proving using ANNs is the difficulty of comparing models operating in different environments [8]. Observed differences in performance may reflect the environment rather than the model architecture. The fundamental principle of this research is therefore uniformity: all models are evaluated using the same proof assistant (Lean 4), the same benchmark (LeanDojo Benchmark 4), the same evaluation procedure, and the same resource limits. Any perceived differences in performance reflect genuine architectural distinctions. Phase I compares three models tested without any changes in their parameters. Phase II takes the best Phase I model and proposes two independent modifications, training and evaluating each separately.

3.2 Standardisation of the Proof Checker

All experiments in this research use Lean 4 as the proof checker. All proof steps generated by the neural models go through verification in the Lean 4 kernel. Lean 4 is also used via a Python interface (LeanDojo framework [8]), which extracts proof states and submits tactics. All these fixed components are shared by all models, so differences in performance between the models are directly attributable to differences in the model itself. There are many differences between the mathematical foundations used by different proof assistants. For instance, the Mathematical Components library [9] implemented in Coq and the Mathlib library implemented in Lean 4 differ in both coverage and organisation. This work standardises the environment for all experiments, removing the inter-experiment confounding factors associated with different environments.

3.3 Data Selection and Processing

All training and evaluation use the LeanDojo Benchmark 4 [8]. An initial portion of this research indicated that NaturalProofs [19] could be useful, but the final methodology uses LeanDojo Benchmark 4 exclusively. While NaturalProofs offers a large library of human-readable proofs, it does not contain any formal system that mechanically checks each step. Without such verification, it is not possible to measure proof validity or error rates objectively. The LeanDojo Benchmark 4 comes from the Lean 4 Mathlib library. The kernel performs real-time checks of generated proof steps. This dataset consists of more than 90,000 formal theorems, which is sufficient for training and evaluation. Step-level fully traced tactics provide the supervision signal for Phase II fine-tuning.

3.3.1 Test Split Strategy

The methodology ensures evaluation integrity by using the exact random split defined by LeanDojo Benchmark 4. The test set is used exclusively for final performance evaluation. Phase II architectural changes use the training split. For all official evaluations, 1,000 theorems are selected

from the test split. There is no selection bias because theorems are sorted by a deterministic hash of the file path and full name. The same 1,000 theorems are used for all three Phase I models and all Phase II official evaluations. Theorems requiring long compilation times are filtered to maintain feasibility within hardware constraints.

3.4 Beam Search Inference Protocol

All models use beam search to explore the proof space. Beam search keeps track of the most promising proof states at each step. At each proof step, the model generates several candidate tactics for each active proof state, which are submitted to the Lean 4 kernel, and the best-scoring new proof states are kept for the next step. Beam width $K = 16$ is used for all experiments. The algorithm employs nucleus sampling [5] to propose tactic candidates, with temperature $\tau = 0.7$ and top-p $= 0.95$. In this way, at each decoding step, the model ranks vocabulary items according to their probabilities and keeps only the minimum subset of them so that their cumulative probability equals 0.95. This pruning allows eliminating less likely candidates while maintaining a reasonable variety of tactic patterns. Prior to being fed into the Lean kernel, any generated tactic is preprocessed. Specifically, it strips off code fence characters, takes only the first line of the tactic, and filters out clearly malformed tactics with unclosed brackets. Then the resulting tactic candidates are processed via the LeanDojo `run_tactic` function. Evaluation is done based on three factors: the degree to which a tactic decreases the number of remaining goals, whether or not the tactic makes a real change in the proof state, and a length penalty. Concretely, the heuristic score is:

$$\text{score}_{\text{heuristic}} = 22.0 \cdot \Delta g + 2.5 \cdot \mathbb{1}[c] - 1.8 \cdot g_{\text{new}} - 0.0002 \cdot |s| - 0.05 \cdot d$$

where Δg is the reduction in remaining goals, $\mathbb{1}[c]$ indicates whether the proof state changed, g_{new} is the number of remaining goals, $|s|$ is the character length of the new proof state, and d is the current proof depth. The beam search algorithm stops either after completing the proof (`ProofFinished`), exceeding the allowed timeout, reaching the maximum number of generated nodes, or running out of valid states. The code listing is provided in Listing 1.

```
    for step_idx in range(max_steps):
        if time.time() - start_time > timeout_s:
            trace["final"] = {"status": "timeout", "expanded_nodes": expanded_nodes}
            return False, [], "timeout", metrics

        new_nodes: List[BeamNode] = []
        step_trace = {"step": step_idx, "beams_in": len(beams), "expanded": []}

        for b_idx, node in enumerate(beams):
            if time.time() - start_time > timeout_s or expanded_nodes >= max_nodes:
                break

            state_pp = short_state_pp(get_state_pp(node.state))
            prompt = build_prompt(state_pp, node.proof)
            candidates = bootstrap_tactics(state_pp, node.proof)

            raw = model.propose_tactics(prompt, n_candidates, temperature, top_p)
            cleaned = []
            for txt in raw:
                tac = clean_generated_tactic(txt)
                if tac:
                    cleaned.append(tac)
                else:
                    metrics["error_invalid_model_output"] += 1

            candidates = uniq_keep_order(candidates + uniq_keep_order(cleaned))
            candidates = rank_tactics(candidates, step_idx, state_pp, node.proof)[:n_candidat

            beam_info = {
                "beam_index": b_idx,
                "num_prev_tactics": len(node.proof),
                "state_num_goals": get_num_goals(node.state),
                "tried": [],
            }

            for tac in candidates:
                if time.time() - start_time > timeout_s or expanded_nodes >= max_nodes:
                    break
                if repeated_or_looping_tactic(tac, node.proof):
                    continue

                expanded_nodes += 1
                metrics["Ngen"] += 1
```

Listing 1. Beam search core loop — `internlm_prover.py`, lines 404–447

3.5 Error Probability Framework

This methodology introduces a structured way to analyse error probability in neural theorem proving. The goal is to assess model reliability inside a formal proof system rather than relying only on aggregate success rates. Failure in neural theorem proving is not a uniform phenomenon. Distinguishing types of errors yields more useful diagnostic information.

Syntactic errors occur when the Lean parser rejects a tactic before elaboration begins. Semantic hallucinations arise during type-checking and elaboration when a syntactically valid tactic fails to align with the underlying logic.

Three metrics are defined. Let N_{gen} be the total number of tactics submitted to the Lean kernel, N_{syn} be the number that pass syntactic parsing, and N_{valid} be the number that successfully advance the proof state.

$$R_{\text{syn}} = \frac{N_{\text{syn}}}{N_{\text{gen}}}, \quad R_{\text{log}} = \frac{N_{\text{valid}}}{N_{\text{syn}}}, \quad P(\text{Error}) = 1 - R_{\text{log}}$$

A smaller $P(\text{Error})$ value implies fewer semantically wrong tactics, thus conserving the search budget and increasing proof probability.

3.6 Phase I Evaluation Protocol

All three candidate models are tested with the same inference configuration: $K = 16$, per-theorem timeout = 180 s, max expanded nodes = 5,000, candidates per beam node = 12, $\tau = 0.7$, top-p nucleus sampling = 0.95, and max tokens per tactic = 64. All three models use the same 1,000 theorems from the LeanDojo test split.

The best architecture in terms of Pass@1 is identified as the reference model for Phase II. Pass@1 is the percentage of theorems for which at least one complete proof is found within a single search run under the fixed parameters. A theorem is counted as proved only when the Lean 4 kernel returns `ProofFinished`, confirming all goals are closed and the proof is logically sound.

3.7 Phase II Fine-Tuning: LoRA and DPO

3.7.1 Low-Rank Adaptation (LoRA)

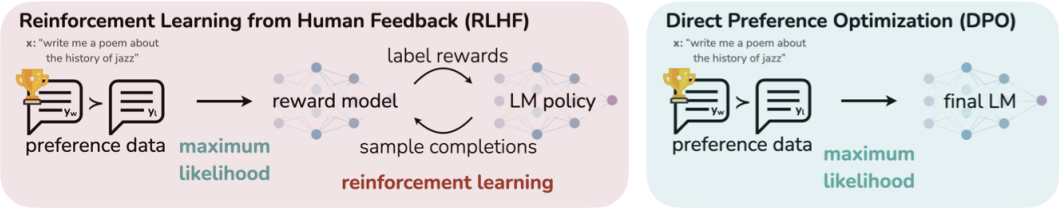
Both Phase II propositions use Low-Rank Adaptation (LoRA) [6] as the fine-tuning mechanism. Standard fine-tuning of a 7-billion-parameter model requires large memory and compute, as it involves updating all parameters. LoRA addresses this by introducing small, trainable matrices added to the frozen original model weights. For a frozen weight matrix W, the update is $W + \Delta W = W + B \cdot A$, where B and A are much smaller matrices with rank r. With rank $r = 8$ and $\alpha = 16$ applied to the wqkv and wo layers of InternLM2.5-StepProver, the number of trainable parameters drops from approximately 7 billion to 4.7 million (0.06% of total), keeping GPU memory within the 32 GB VRAM limit of the V100 accelerator.

3.7.2 Direct Preference Optimization (DPO)

Proposition B uses Direct Preference Optimization (DPO) [15] to train the error-conditioned refiner. DPO trains the model to prefer one output over another given the same input. The model learns to prefer a corrected tactic $T_{\text{corrected}}$ over the failed tactic T_{failed}, given the proof state and the Lean 4 error message. The DPO loss function is:

$$\mathcal{L}_{\text{DPO}} = -\mathbb{E}[\log \sigma(\beta \cdot ((\log \pi_\theta(T_c|x) - \log \pi_{\text{ref}}(T_c|x)) - (\log \pi_\theta(T_f|x) - \log \pi_{\text{ref}}(T_f|x))))]$$

where π_θ is the trained policy, π_{ref} is a frozen reference model with the same initial weights, and $\beta = 0.1$ controls how far the policy can deviate from the reference. DPO shows greater stability than standard reinforcement learning.



2 Figure *The DPO training pipeline. The model learns to prefer the chosen response* $T_{corrected}$ *over the rejected response* T_{failed} *without requiring a separate reward model [15].*

3.7.3 Hyperparameter Configuration

The temperature parameter $\tau = 0.7$ is used in Phase I and for Proposition B. This value sharpens the token probability distribution compared to the default $\tau = 1.0$, reducing syntactically invalid tactics. Values that get close to $\tau \to 0$ introduce determinism and result in the search becoming stagnant [5]. Temperature $\tau = 0.85$ is employed in Proposition A to minimize the no_more_states failure cases, which will allow for diversity in tactics generation. The beam width of $K = 16$ ensures the best balance between coverage of the proof space and the associated computational effort. For Phase I, there is a timeout of 180 seconds per theorem, which is increased to 240 seconds in Phase II.

3.7.4 LeanDojo Interface

One of the important features of the methodology is the interaction of Python neural models with Lean.4. LeanDojo [8] is a Lean process, which is controlled by the Python script through inter-process communication. The proof state is retrieved from the Lean kernel at each proof step as a pretty-printed string. It includes information about the proof goals and hypotheses. The return value can be either `ProofFinished`, new `TacticState` or `LeanError`. A watchdog system guarantees that local errors will not impact the whole evaluation process.

3.7.5 Advanced Evaluation Metrics

In addition to the Pass@1 binary measure, there are two other measures employed in this research.

Time-to-Proof (TTP) is defined as the actual time taken from when a theorem is provided to the moment the `ProofFinished` signal is received. The shorter the TTP, the better it is in terms of finding proofs quickly.

Tactic Diversity Index (TDI) assesses whether a model over-relies on a small set of tactics, measured through Shannon entropy:

$$H(T) = -\sum_{i} P(t_i) \log_2 P(t_i) \tag{1}$$

where t_i are the tactic stems. Low entropy shows over-reliance on dominant tactics.

3.7.6 Computing Infrastructure

Experiments are carried out at the HPC facility of Vilnius University MIF. Each job uses one NVIDIA Tesla V100-SXM2-32GB GPU and Intel Xeon processors, with the Lustre parallel file system for data storage. The SLURM workload manager handles job scheduling and resource allocation. The software environment uses Python 3.10, PyTorch 2.1.2 with CUDA 12.1, Hugging Face Transformers library version 4.36, LeanDojo version 1.6.0, and the PEFT library for LoRA fine-tuning. The Lean toolchain version 4.3.0 is managed by the elan version manager. Phase I evaluation of 1,000 theorems requires approximately 21.8 wall-clock hours for InternLM2.5-StepProver (78.3 seconds average per theorem). Phase II training runs need 8–10 hours per run.

The complete source code for all experiments is publicly available at [16].

4 Phase I: Comparative Analysis

4.1 Overview

In Phase I, three models are tested on the same 1,000 theorems from a random test split of LeanDojo Benchmark 4 to assess their performance in the same environment. None of the model parameters are altered during Phase I. All official findings in this chapter are from the 1,000-theorem evaluation.

4.2 InternLM2.5-StepProver

InternLM2.5-StepProver proves 457 out of 1,000 theorems for a Pass@1 rate of 45.7%, the highest result in Phase I. Out of the 65,999 tactics generated during the evaluation, 56,254 (85.2%) pass the syntactic parsing test and 9,661 (17.2%) successfully advance the proof state. The step-wise hallucination rate is $P(\text{Error}) = 0.828$.

The failure mode breakdown is: 198 theorems time out (19.8%), 324 fail because no more reachable proof states exist (32.4%), and 21 fail due to errors. A large no_more_states value indicates that the model frequently produces identical strategies over and over again until running out of search space within the allotted time. The analysis at the step level reveals 9,745 syntax errors and 46,593 semantic hallucinations, which proves the dominance of logical invalidity in terms of wastage of search budget.

4.3 ReProver

ReProver proves 446 out of 1,000 theorems for a Pass@1 rate of 44.6%, placing it 1.1 percentage points below InternLM2.5-StepProver. Four theorems are discarded due to framework initialisation failures. The average evaluation time per theorem is 113.5 seconds, significantly higher than InternLM2.5-StepProver's 78.3 seconds. The runtime difference is due to ReProver's best-first search exploring more states per theorem: 377.4 nodes on average versus 12.2 searched nodes.

The DistributedProver framework used for the official ReProver evaluation does not expose step-level tactic metrics, as individual tactic outcomes are not accessible. A separate sequential evaluation using the same retrieval-augmented tactic generator through LeanDojo's Dojo interface on the same 1,000 theorems gives Pass@1 $= 38.5\%$, $R_{\text{syn}} = 0.537$, $R_{\text{log}} = 0.243$, and $P(\text{Error}) = 0.758$. These metrics characterise the tactic generator rather than the search strategy.

4.4 DeepSeek-Prover-v1.5-RL

DeepSeek-Prover-v1.5-RL proves 403 out of 1,000 theorems for a Pass@1 rate of 40.3%, the lowest result among the three models. The model generates 174,274 tactics with an average of 174.3 expanded nodes per theorem. The syntax compliance rate is $R_{\text{syn}} = 0.819$, lower than InternLM2.5-StepProver's 0.852, consistent with RL training introducing more syntactic variability. The logical

precision is $R_{\text{log}} = 0.174$ and hallucination rate $P(\text{Error}) = 0.826$. Failure modes show 323 timeouts (32.3%), 186 no_more_states (18.6%), and 87 exception failures (8.7%).

4.5 Failure Pattern Comparison

The three models exhibit different failure patterns due to their different training paradigms. The reason why InternLM2.5-StepProver has the largest percentage of no_more_states (32.4%) is that it frequently exhausts the reachable proof states, resulting in the no_more_states failure mode rather than a timeout. This makes sense because in supervised learning, the model tends to repeat tactics close to what it was taught in the distribution, hence when the necessary tactic deviates from that distribution, then it becomes exhausted. Conversely, DeepSeek-Prover-v1.5-RL has the lowest percentage of no_more_states and the highest timeouts (18.6%, 32.3%).This diversity comes at the cost of syntactic quality ($R_{\text{syn}} = 0.819$ vs InternLM's 0.852).

According to ReProver's failure breakdown from the sequential evaluation: 557 timeouts (55.7%), 54 no_more_states (5.4%), and 4 exceptions (0.4%). The very high timeout rate in the sequential evaluation reflects the fact that the sequential search often reaches the time limit, whereas the parallel DistributedProver uses a best-first strategy. The low no_more_states count shows that the retrieval-augmented generator produces very diverse tactic suggestions.

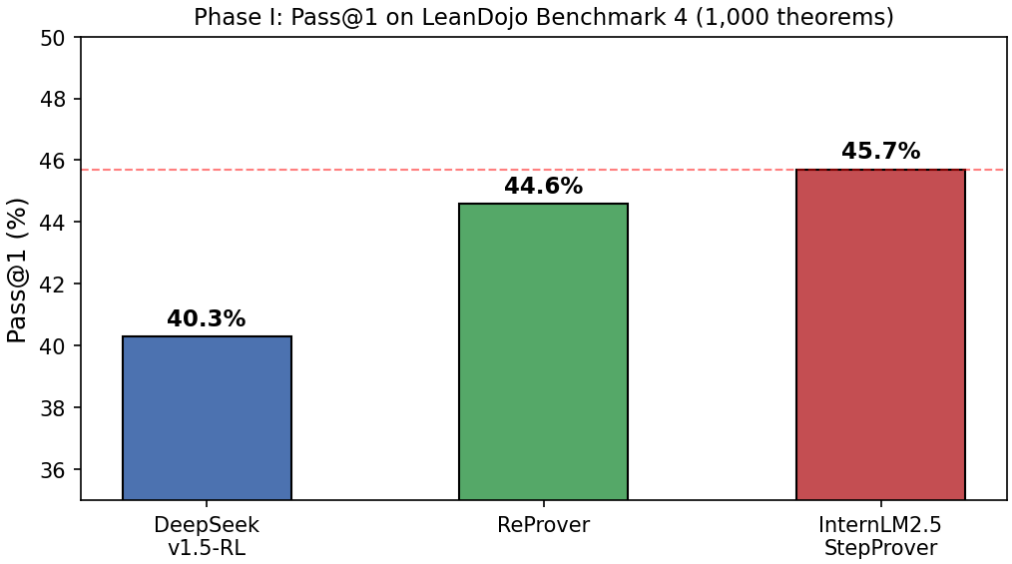
These failure patterns directly influence the Phase II design choices. The high no_more_states count for InternLM motivates using a higher temperature for Proposition A ($\tau = 0.85$) to generate a greater variety of tactics. The high rate of semantic hallucinations across all models motivates both the value head (Proposition A) and the error-conditioned refiner (Proposition B).

4.6 Comparative Results

4 Table *Phase I Primary Results on LeanDojo Benchmark 4 (1,000 theorems, random test split)*

Model	Proved	Failed	Total	Pass@1	Avg Time (s)	Verifier
InternLM2.5-StepProver	457	543	1,000	45.7%	78.3	Lean 4
ReProver†	446	550	1,000	44.6%	113.5	Lean 4
DeepSeek-Prover-v1.5-RL	403	597	1,000	40.3%	96.5	Lean 4

†ReProver: 4 theorems discarded due to framework initialisation failures (446 + 550 + 4 = 1,000).



3 Figure *Pass@1 results for the three Phase I models on the LeanDojo Benchmark 4 (1,000 theorems, random test split). InternLM2.5-StepProver achieves the best result at 45.7%.*

4.7 Failure Mode Analysis

5 Table *Phase I Failure Mode Distribution. ReProver (seq.): values from sequential Dojo evaluation (Pass@1 $= 38.5\%$, 1,000 theorems). Official parallel DistributedProver achieves Pass@1 $= 44.6\%$ but does not expose failure breakdown data.*

Model	Proved	Timeout	No More States	Error/Disc.	Max Nodes
InternLM2.5-StepProver	457	198	324	21	0
ReProver (seq.)	385	557	54	4 (exc.)	N/A
DeepSeek-Prover-v1.5-RL	403	323	186	88 (exc.+max)	N/A

4.8 Error Probability Analysis

6 Table *Error Probability Metrics. ReProver (seq.): all values from sequential Dojo evaluation. Official parallel DistributedProver gives Pass@1 $= 44.6\%$ but does not expose step-level metrics.*

Model	N_{gen}	N_{syn}	N_{valid}	R_{syn}	R_{log}	$P(\text{Error})$
InternLM2.5-StepProver	65,999	56,254	9,661	0.852	0.172	0.828
ReProver (seq.)	774,495	416,041	100,884	0.537	0.243	0.758
DeepSeek-Prover-v1.5-RL	174,274	142,696	24,807	0.819	0.174	0.826

The error probability analysis reveals that semantic hallucination is the predominant error source for all three models. All three achieve R_{syn} above 0.81, showing the syntax problem is largely resolved. The key bottleneck is semantic: 82.8% of InternLM2.5-StepProver's syntactically valid tactics are logically wrong, 82.6% for DeepSeek, and 75.8% for ReProver's generator. This finding directly motivates both Phase II modifications.

4.9 ReProver Runtime Metrics

7 Table *ReProver Runtime Metrics (1,000 theorems, DistributedProver framework)*

Metric	Mean	Min	Max	Notes
Total time per theorem (s)	113.5	1.1	182.3	Model + Lean kernel
Actor time (model, s)	16.9	0.8	45.2	Generation only
Environment time (Lean, s)	91.9	0.3	177.5	Kernel calls only
Total nodes explored	377.4	2	1,500	States in tree
Searched nodes	12.2	1	87	Best-first search

4.10 Selection of Reference Model

InternLM2.5-StepProver is selected as the reference model for Phase II based on the highest Pass@1 (45.7%), the highest syntax compliance rate ($R_{\text{syn}} = 0.852$), and the most complete step-level metrics. Its standard autoregressive architecture makes it straightforward to add auxiliary heads and implement new inference mechanisms. DeepSeek's RL training and ReProver's retrieval architecture would require more significant changes to implement the proposed modifications.

5 Phase II: Proposed Architectural Modifications

5.1 Overview

In Phase II, InternLM2.5-StepProver is used as the reference model and two independent modifications, Proposition A and Proposition B, are proposed to address the semantic hallucination problem identified in Phase I ($P(\text{Error}) = 0.828$). Proposition A addresses inefficient proof search by estimating proof state solvability and pruning unproductive branches. Proposition B addresses failure recovery by using Lean 4 compiler error messages to correct failed tactics. Both modifications use LoRA with rank $r = 8$ and $\alpha = 16$ applied to the wqkv and wo attention layers.

5.2 Proposition A: Value-Augmented Architecture

5.2.1 Architecture

Proposition A adds a value head to InternLM2.5-StepProver, a small two-layer MLP that takes a description of the current proof state and estimates the probability that it can be solved:

$$V_\varphi(f) = \sigma(W_2 \cdot \text{ReLU}(W_1 \cdot f + b_1) + b_2)$$

where f is a feature vector, W_1, W_2 are learnable weight matrices, b_1, b_2 are bias vectors, and the sigmoid function maps the output to $[0, 1]$.

5.2.2 The Hidden-State Problem and Feature-Based Solution

The initial design was based on the hidden states of the backbone model (the internal representation at the last token of the final transformer layer). However, this approach did not work for InternLM2.5-StepProver. The backbone's hidden states look very similar for all Lean 4 proof states regardless of whether the state is easy or hard to solve. InternLM was pre-trained on all types of Lean 4 content and learned to produce similar internal representations for all valid states. As a result, the value head outputs near 1.0 for every state. This is a contribution of this research: backbone hidden states in models pre-trained on Lean 4 corpora do not encode proof-state solvability.

The solution uses seven handcrafted proof-state features as input to the value head. These features are extracted from the proof state text, the proof depth, and the tactic history, and they measure properties known to correlate with proof difficulty:

- `has_and`: whether the goal contains a conjunction ($\wedge$).
- `has_or`: whether the goal contains a disjunction ($\vee$).
- `has_iff`: whether the goal contains a biconditional ($\leftrightarrow$).
- `has_exists`: whether the goal has an existential quantifier ($\exists$); such goals need a witness.

- `has_eq`: whether the goal contains an equation ($=$); which is commonly solved using `rfl` or `simp`.
- `has_order`: whether the goal has order relations ($\leq$, $<$); often handled by `linarith` or `omega`.
- `many_goals`: whether more than one proof goal remains ($\vdash$ count > 1).

```
def state_features(state_pp: str) -> Dict[str, bool]:
    return {
        "has_and":    "∧" in state_pp,
        "has_or":     "∨" in state_pp,
        "has_iff":    "↔" in state_pp,
        "has_exists": "∃" in state_pp,
        "has_eq":     "=" in state_pp,
        "has_order":  "≤" in state_pp or "<" in state_pp,
        "many_goals": state_pp.count("⊢") > 1,
    }
```

Listing 2. Proof-state feature extraction — `internlm_value_augmented_prover.py`, lines 173–183

5.2.3 Training

The value head is trained jointly with the LoRA adapters using a multi-task loss: $\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{CLM}} + \lambda \cdot \mathcal{L}_{\text{MSE}}$ with $\lambda = 0.5$. $\mathcal{L}_{\text{CLM}}$ is the standard language modelling loss on positive examples only. $\mathcal{L}_{\text{MSE}}$ is the weighted mean squared error for value head prediction. Negative examples receive a weight of 3.0 to counteract the model's tendency to predict high values for all states. The training dataset contains 100,000 examples with a 50/50 balance. Positive examples (label 1.0) are proof states from verified proofs in the LeanDojo training split. Negative examples (label 0.0) come from failed theorem attempts in the Phase I beam-search traces. These real failed states are significantly better than synthetic corrupted states because they represent genuine Lean 4 states that the model explored but could not solve.

***8 Table** Proposition A Value Head Training Configuration*

Parameter	Value	Parameter	Value
LoRA rank r	8	Negative weight	3.0
Learning rate	2×10^{-4}	Training examples	100,000
Effective batch size	16	Epochs	3
λ	0.5	Final MSE	0.0186
Value head size	$7 \to 512 \to 1$	Training time	$\sim$8h V100

5.2.4 Inference Integration

During beam search, after each tactic produces a valid new proof state, the value head computes the seven features from the new state and estimates solvability. States below the threshold (`value_threshold` $= 0.60$) are pruned. Surviving states receive a combined score:

$$\text{score}_{\text{combined}} = \text{score}_{\text{heuristic}} + \text{value\_weight} \times (v - 0.5) \times 10$$

where value_weight $= 2.0$ and $\text{score}_{\text{heuristic}}$ is defined in Section 3.4. The value head contribution ranges from -10 to $+10$, making it a meaningful but not dominant adjustment relative to the heuristic score.

5.3 Proposition B: Error-Conditioned Architecture

5.3.1 Architecture

Since the standard model sees only the current proof state when generating a tactic, Proposition B adds a recursive error correction mechanism. The error-conditioned refiner also sees the failed tactic and the exact error message returned by the Lean 4 kernel, producing a composite input:

$$\text{Input} = \text{State } S_t \ \oplus \ \text{Failed Tactic } T_{\text{failed}} \ \oplus \ \text{Error Message } E_{\text{msg}}$$

Segment marker tokens separate the three parts so the model can distinguish the proof goal from the error context.

5.3.2 Training Data: Why Real Error Pairs Matter

Proposition B required two training approaches. The first used synthetic error pairs: a proof state from the training split paired with a randomly chosen wrong tactic and a generic error message, producing 5,000–20,000 examples. All configurations trained on synthetic errors reduced overall performance. Synthetic error messages do not reflect what Lean 4 actually says when a tactic fails; the model learned unrealistic correction patterns that hurt general tactic generation.

The second approach collected real error pairs using a dedicated data collection script. For each of 2,000 training theorems, the script opens a LeanDojo session and applies commonly-failing tactics at each step of the verified proof. The Lean 4 kernel returns a real error message, providing a genuine $(S_t, T_{\text{failed}}, E_{\text{msg}}, T_{\text{corrected}})$ tuple. This process produced 13,403 real DPO pairs, of which 10,000 are used for training.

```
    for step_idx, tt in enumerate(traced):
        if len(pairs) >= max_pairs_per_theorem:
            break

        correct_tac = (tt.get("tactic") or "").strip()
        state_pp    = getattr(current_state, "pp", "").strip()

        if not correct_tac or not state_pp:
            break

        # Exclude trivial or incomplete tactics from the corrected label
        if correct_tac.lower() in {"sorry", "admit", ""}:
            break

        # Apply incorrect tactics to the current proof state to elicit real error messages
        wrong_subset = rng.sample(wrong_tactics, min(6, len(wrong_tactics)))
        for wrong_tac in wrong_subset:
            if wrong_tac == correct_tac:
                continue
            is_err, err_msg = try_tactic(dojo, current_state, wrong_tac)
            if is_err and err_msg and "sorry" not in err_msg.lower():
                pairs.append({
                    "state_pp":      state_pp[:2000],
                    "failed_tactic": wrong_tac,
                    "error_msg":     err_msg,
                    "correct_tactic": correct_tac,
                    "theorem":       entry["full_name"],
                    "step_idx":      step_idx,
                })
                if len(pairs) >= max_pairs_per_theorem:
                    break

        # Advance proof state using the verified correct tactic
        try:
            for method in ("run_tactic", "run_tac", "run"):
                if hasattr(dojo, method):
```

Listing 3. Real DPO error pair collection from the Lean 4 kernel — `dpo_pair_collector.py`, lines 123–158

5.3.3 Training

DPO training uses AdamW with learning rate 5×10^{-5}, effective batch size 16, for 3 epochs on 10,000 real error pairs. Here, `chosen_better` is the fraction of training examples where the policy assigns higher log-probability to the corrected tactic than to the failed tactic. The reward margin is the mean difference in β-scaled log-probability ratio between the chosen and rejected responses. The training converges strongly: chosen_better $= 1.000$, indicating the trained model always assigns higher probability to the corrected tactic than the failed tactic in all training examples, with reward margin $= 13.62$. These values reflect training set performance; the official 1,000-theorem evaluation serves as external validation of generalisation. Total training time is approximately 8–10 hours on the V100 GPU.

5.3.4 Inference Integration

In beam search, when a tactic fails with a semantic error (not a pure syntax error), the refiner is called with the composite input. If the generated correction differs from the failed tactic and passes basic validation, it is submitted to the Lean 4 kernel. Refiner calls are limited to 2 per beam node to control overhead. Only semantic errors trigger the refiner, since syntactic errors require a structurally different tactic rather than a semantic correction.

6 Discussion of Phase I Results and Phase II Design Rationale

6.1 Comparison with Published Results

The results obtained in this thesis differ from those reported in the original papers. InternLM2.5-StepProver achieves 65.9% Pass@1 on miniF2F [23] in its paper [21], compared to 45.7% in this thesis on the LeanDojo Benchmark 4 random test split. DeepSeek-Prover-v1.5-RL reports 63.5% on miniF2F-test [22, 23], compared to 40.3% here. These discrepancies arise from three factors.

- The benchmarks are fundamentally different. The miniF2F benchmark [23] draws from olympiad competitions, whereas the LeanDojo Benchmark 4 random split covers a much broader range of Mathlib theorems across all difficulty levels. Models fine-tuned for competition mathematics may not generalise equally well to library-style theorems.
- There are variations in inference configurations. Published results benefit from considerably more compute: DeepSeek, for instance, uses MCTS to explore a far larger search tree than the beam search used here. All models in this thesis use the same setup: beam width of 16 and a 180-second timeout for Phase I.
- The paper experiments may employ different versions of Lean 4 and Mathlib. Lean 4 and Mathlib change over time, and therefore the compatibility with model weights and the latest version of the library may play a role.

However, despite these discrepancies, InternLM2.5-StepProver ranks first in both evaluations. The reversed order of ReProver and DeepSeek is likely due to the benchmark difference: retrieval augmentation helps ReProver more on library-style theorems, while DeepSeek's RL training is better suited to competition-style problems.

6.2 Why Semantic Hallucination is Hard to Solve

The results of our error probability analysis demonstrate that the issue lies in semantic hallucinations ($P(\text{Error}) > 0.82$ for both InternLM and DeepSeek), rather than syntactic information. The issue here runs much deeper. Even if a model can successfully predict the syntactic representation of a tactic, it still needs to understand the logical connection between the proof state and tactics available at each step. For example, while using the `ring` tactic on goals containing real-valued functions, rather than ring elements, may be syntactically correct, it is logically nonsensical. Avoiding mistakes of this kind requires knowledge of the type-theoretic structure of the goal. While bigger models do output a greater variety of syntactically diverse tactics, they still have to deal with the same logical issues. Having more parameters does not mean better understanding of type-theoretic structure. That is why we introduce the two architectural strategies described in Proposition A and B of Phase II. Proposition A maximizes the efficiency of the search budget spent by cutting out impossible states as early as possible, while Proposition B tries to correct hallucinations based on the error message.

6.3 Implications of the Backbone Invariance Finding

The finding that InternLM2.5-StepProver's backbone hidden states are invariant to proof-state solvability has broader implications for neural theorem proving research. Simply adding a value head to an existing model and training it on labelled proof states will not work if the base model's representations do not encode the relevant information. In contrast, encoding position value in the representations from the beginning is extremely useful in game playing: in AlphaZero-style systems [17], the model is trained end-to-end with a value signal throughout.

What is avoided in this thesis is the representation problem altogether, using handcrafted features that directly measure observable properties of the proof state. A more principled approach would be to retrain a model from scratch using both a tactic prediction objective and a value prediction objective, so that the learned representations encode both types of information. This is a promising direction for future research.

7 Phase II Results and Advanced Metrics

7.1 Proposition A: Ablation Study

An ablation study was conducted before the official 1,000-theorem evaluation to identify the best configuration. Five configurations were tested by varying the value threshold, number of candidates, and temperature. All ablation runs were evaluated on the same 100-theorem subset used for preliminary experiments.

9 Table *Proposition A Ablation Study (feature-based value head)*

Configuration	Pass@1	Pruned	Avg Value	Threshold	n-cand	τ
v1: synthetic negatives	50%	2	0.966	0.08	12	0.70
v2: real negatives, 6.5k	51%	35	0.697	0.08	12	0.70
v3: 100k, threshold=0.30	48%	395	0.536	0.30	32	0.70
v4: threshold=0.15, n=20	55%	11	0.541	0.15	20	0.70
v5 BEST: thr=0.60, τ=0.85	56%	110	0.797	0.60	20	0.85

Configuration v1 used synthetic corrupted proof states as negatives, producing a degenerate value head that outputs near-1.0 for almost all states and prunes almost nothing. Configuration v2 with actual failed states performs better with calibration (average value 0.697) and 35 branches pruned. Config v3 demonstrates the negative impact of over-pruning with 395 branches pruned but the accuracy is reduced to Pass@1 with 48%, where the correct branches were trimmed off. Config v5 is able to achieve 56% through the use of a reasonable threshold of 0.60 and 20 candidates at τ= 0.85.

7.2 Proposition A: Results

10 Table *Proposition A Results*

Configuration	Pass@1	Pruned	Avg Value	Notes
Phase I Baseline (InternLM)	45.7%	—	—	Reference
Prop A v5 (best config)	56.0%	110	0.797	Feature value head
Prop A (1,000 thm, official)	51.9%	640	0.811	+6.2pp over baseline

For the official evaluation, the 1,000-theorem evaluation obtains a 51.9% Pass@1 score with 640 branches pruned while achieving an average value score of 0.811. This constitutes a +6.2 percentage points gain compared to the 45.7% baseline. Our implementation of the feature-based value head works, given the average value score of 0.811 is not always a constant of 0.998 like the hidden-state approach.

One thing that must be pointed out is the difference between the Proposition A and Phase I baselines for the syntax compliance rate is slightly lower ($R_{\text{syn}} = 0.852$ vs $R_{\text{syn}} = 0.842$) while the hallucination rate increased ($P(\text{Error}) = 0.828$ vs $P(\text{Error}) = 0.854$) . These are to be expected

because the Proposition A model uses a higher temperature of $\tau = 0.85$ compared to Phase I's $\tau = 0.70$. Increasing the temperature results in greater tactic diversity, thereby reducing the number of occurrences of no_more_states errors from 324 in Phase I to 184 in Proposition A, but at the expense of some decrease in syntactic quality. However, this trade-off is favorable as our +6.2 percentage points improvement indicates.

7.3 Proposition B: Results

11 Table *Proposition B Results. v1–v3 ablation on the 100-theorem subset; v3 official is the 1,000-theorem evaluation.*

Configuration	Pass@1	Ref Calls	Ref Succ.	Ref Rate	No More	Timeout
Baseline (1K thm)	45.7%	—	—	—	324	198
v1: 5k synthetic, max-ref=3 (100 thm)	46%	1,496	278	18.6%	33	21
v2: 5k synthetic, max-ref=2 (100 thm)	45%	717	186	25.9%	34	21
v3: 10k real pairs (100 thm)	51%	1,176	204	17.3%	25	21
v3 official (1,000 thm)	45.9%	10,899	1,773	16.3%	322	188

Proposition B v3, trained on 10,000 real Lean 4 error pairs, achieves 45.9% Pass@1 on 1,000 theorems with 1,773 successful corrections out of 10,899 refiner calls at 16.3%. In about 1 in 6 cases where a tactic fails with a semantic error, the refiner generates a corrected tactic that advances the proof. The 1,000-theorem result of 45.9% is 0.2 percentage points above the baseline of 45.7%. The failure mode distribution shows: proved $= 459$, no_more_states $= 322$, timeout $= 188$, and error $= 31$.

A comparison of synthetic and real error training is clear. Synthetic versions v1 and v2 achieve no better than baseline performance because synthetic error messages cause unrealistic correction patterns that hurt general tactic generation. The real error version v3 provides a 16.3–17.3% correction rate without significantly impacting baseline performance. Real Lean 4 compiler error messages are essential for effective DPO training.

7.4 Time-to-Proof Analysis

12 Table *Time-to-Proof Statistics*

Configuration	N (proved)	Mean TTP (s)	Median (s)	Min (s)	Max (s)
InternLM Phase I (1,000 thm)	457	35.9	22.1	9.5	152.3
Prop A (1,000 thm)	519	27.8	14.7	2.0	252.2
Prop B (1,000 thm)	459	34.1	19.6	3.9	247.8

Proved theorems are resolved much faster than the timeout suggests. The mean TTP for InternLM2.5-StepProver is 35.9 seconds, well below the 180-second timeout. The minimum TTP of 9.5 seconds for the baseline represents theorems solved in the first few steps using tactics like`rfl` or `simp`. Proposition A shows a lower mean TTP of 27.8 seconds compared to the baseline's 35.9 seconds, suggesting that the value head directs the search toward more promising branches, allowing proofs to be found more quickly.

7.5 Tactic Diversity Analysis

13 Table *Tactic Diversity Comparison*

Model	TDI (bits)	Avg Proof Len	Top Tactic	Top %	Proofs Analysed
InternLM2.5-StepProver	3.59	1.63	`exact?`	17.3%	457 (1k thm)
ReProver (seq.)	4.26	2.15	`simp`	31.3%	385 (1k thm)
DeepSeek-Prover-v1.5-RL	3.65	1.65	`aesop?`	22.3%	403 (1k thm)

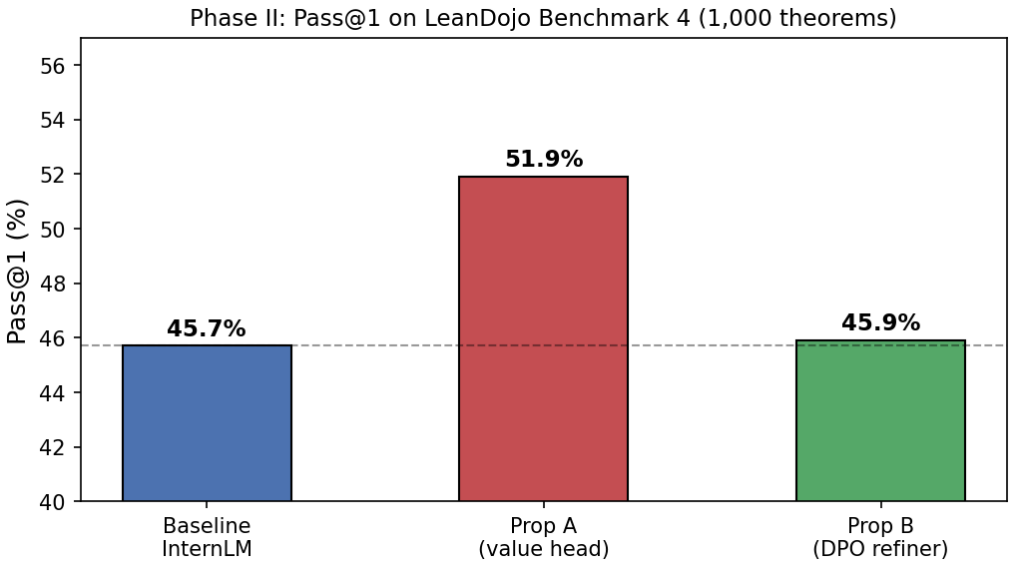
All three models show similar tactic diversity ranging from 3.59 to 4.26 bits. The average proof lengths are short: 1.63 (InternLM), 2.15 (ReProver seq.), and 1.65 (DeepSeek). This reflects the prevalence of short proofs where a single powerful automation tactic closes the goal. The top five tactics for InternLM are: `exact?` at 17.3%, `rfl` at 15.8%, `simp` at 13.7%, `aesop?` at 13.3%, and `exact` at 8.2%. For DeepSeek: `aesop?` at 22.3%, `simp` at 14.5%, `exact` at 12.0%, `apply` at 9.9%, and `rw` at 9.2%. For ReProver (seq.): `simp` at 31.3%, `rw` at 8.6%, `exact` at 7.0%, `aesop` at 5.0%, and `cases` at 4.2%.

7.6 Complete Results Comparison

Table 14 and Figure 4 bring together all configurations evaluated in this thesis, allowing direct comparison of Phase I models and Phase II modifications under the same 1,000-theorem evaluation.

14 Table *Complete Results Comparison. ReProver: Pass@1 from parallel DistributedProver (44.6%);* R_{syn}*/*$P(\text{Error})$ *from sequential Dojo evaluation.*

Configuration	**Theorems**	**Pass@1**	R_{syn}	$P(\text{Error})$	**Key Feature**
DeepSeek-Prover-v1.5-RL	1,000	40.3%	0.819	0.826	RL training
ReProver	1,000	44.6%	0.537	0.758	Retrieval augmented
InternLM2.5-StepProver	1,000	45.7%	0.852	0.828	Step-level SFT
Prop A (feature value head)	1,000	51.9%	0.842	0.854	640 branches pruned
Prop B (real DPO pairs)	1,000	45.9%	0.738	0.858	16.3% correction



4 Figure *Phase II results compared to the Phase I baseline. Proposition A achieves the largest improvement (+6.2pp), while Proposition B matches the baseline with an added 16.3% error correction capability.*

8 Results and Conclusions

Summary of Results

Phase I. Three models are evaluated under identical conditions on 1,000 theorems. InternLM2.5-StepProver, ReProver, and DeepSeek-Prover-v1.5-RL achieve Pass@1 of 45.7%, 44.6%, and 40.3% respectively. The error probability framework shows $P(\text{Error}) = 0.828$ for InternLM2.5-StepProver, 0.826 for DeepSeek, and 0.758 for ReProver's generator (from the sequential evaluation). Semantic hallucination accounts for 74–82% of step-level failures across all models.

Proposition A. With 640 branches pruned, it achieves 51.9% Pass@1 on 1,000 theorems (+6.2pp over baseline). A key finding is that backbone hidden states of InternLM2.5-StepProver are invariant to proof-state solvability, making them unsuitable for value estimation. Handcrafted features measuring observable proof-state properties provide an effective alternative.

Proposition B. The error-conditioned refiner trained on 10,000 real Lean 4 compiler error pairs achieves 45.9% Pass@1 on 1,000 theorems with a 16.3% per-tactic correction rate. Training on real error pairs is essential: synthetic errors reduce performance while real errors add meaningful correction capability without hurting the baseline.

Conclusions

1. **Environmental standardisation is essential.** Pass@1 scores are 40.3–45.7% across the three models under identical conditions, a gap of only 5.4 percentage points. Differences found in studies comparing models under different settings might not reflect genuine architectural advantages. This study presents a comparison that is neutral and balanced.

2. **Semantic hallucination is the main bottleneck.** With $P(\text{Error})$ above 0.82 for InternLM and DeepSeek, more than 4 out of 5 syntactically valid tactics are logically wrong. Improving syntactic accuracy alone cannot significantly improve Pass@1. The key opportunity lies in semantic understanding.

3. **Feature-based value estimation works for beam pruning.** The 640 pruned branches and +6.2 percentage point improvement confirm that handcrafted proof-state features guide beam search effectively. Backbone hidden states are unsuitable for this purpose in models pre-trained on all Lean 4 content.

4. **Real compiler error pairs are essential for DPO training.** Synthetic error messages hurt performance while real Lean 4 error messages enable a 16.3% correction rate. Data quality is more important than data quantity for DPO-based error correction.

5. **The error probability framework reveals what pass rates cannot.** Models with similar Pass@1 rates show different failure patterns. ReProver's generator has better logical precision ($R_{\text{log}} = 0.243$) than InternLM ($R_{\text{log}} = 0.172$), though InternLM has better syntax compliance. These insights guide targeted architectural improvements.

Ethical Considerations

- **Environmental Impact of Neural Training:** The training and adjustment of large language models requires intensive computation and significant energy consumption. To reduce this environmental footprint, a transfer-learning strategy is adopted throughout all experimental phases. Instead of training models from scratch — a process that typically consumes trillions of tokens and months of GPU time — existing pre-trained backbones are used for all candidate architectures. By initialising weights from the pre-trained InternLM2.5-StepProver backbone and applying LoRA-based fine-tuning only within the specific domain of theorem proving during Phase II, the computational cost is reduced by several orders of magnitude compared to full pre-training. DeepSeek-Prover-v1.5-RL and ReProver are used in Phase I as-is, without any weight updates. This approach supports sustainable AI principles by maximising the value of existing open-source computation.

- **Interpretability Limitation of Neural Generators:** A major limitation of this work concerns the interpretability of neural generators. The Lean 4 kernel provides strict logical verification, ensuring that any accepted proof is logically correct. However, the internal process by which the model chooses one tactic instead of another cannot be directly inspected. In practice, one can observe certain reasoning strategies in the output — such as choosing induction on a specific variable — but the underlying justification for why the model favoured that strategy cannot be formally extracted.

  However, part of the gap regarding interpretability can be bridged by the error-probability approach presented in Chapter 3. By measuring R_{syn}, R_{log}, and $P(\text{Error})$, the approach helps establish some measure of confidence with regard to the reliability of the logic used in the model, although it may not reveal all of it internally.

- **Scope of the Research:** Restriction of the experiments is set in that only theorems from the Mathlib collection in Lean 4 are included, as provided by LeanDojo Benchmark 4 [8]. The system is meant for reasoning over formalized mathematics that is already proven and navigating in it based on machine-learned tactic sequences, and not for building new mathematics from axioms. The results have to be interpreted taking this into account. The tested theorems are library-style theorems that differ from the competition style ones described in Chapter 6.

Limitations and Future Work

However, the main drawback of Proposition A lies in the fact that the value head is trained on a fixed snapshot of proof states. As the search strategy changes, the distribution shifts and the value head may not generalise well. Future work should retrain the value head iteratively as the search parameters are updated. For Proposition B, only 10,000 pairs from 2,000 theorems were used. The full training split has 118,517 theorems, and scaling the data collection could produce over 100,000 real error pairs, substantially improving the correction rate.

The most promising direction is combining Proposition A and Proposition B in a single architecture. The value head would prune dead-end branches while the refiner recovers from errors within the promising branches. This combined architecture is the most natural extension of the work presented.

A further direction is applying the proposed modifications to larger models. The value head and error-conditioned refiner could show greater benefits at larger parameter scales, where the base tactic generation quality is already higher.

References

[1] K. Bansal, S. Loos, M. Rabe, C. Szegedy, S. Wilcox. „HOList: An Environment for Machine Learning of Higher Order Logic Theorem Proving“. Iš: *Proceedings of the 36th International Conference on Machine Learning (ICML 2019)*. PMLR, 2019, puslapiai 454–463. URL: https://proceedings.mlr.press/v97/bansal19a.html.

[2] M. Davis, H. Putnam. „A computing procedure for quantification theory“. Iš: *Journal of the ACM* 7.3 (1960), puslapiai 201–215. https://doi.org/10.1145/321033.321034.

[3] G. Gonthier. „Formal Proof — The Four-Color Theorem“. Iš: *Notices of the American Mathematical Society* 55.11 (2008), puslapiai 1382–1393. URL: https://www.ams.org/notices/200811/tx081101382p.pdf.

[4] J. M. Han, J. Rute, Y. Wu, E. Ayers, S. Polu. „Proof Artifact Co-Training for Theorem Proving with Language Models“. Iš: *International Conference on Learning Representations (ICLR)*. 2022. URL: https://arxiv.org/abs/2102.06203.

[5] A. Holtzman, J. Buys, L. Du, M. Forbes, Y. Choi. „The Curious Case of Neural Text Degeneration“. Iš: *International Conference on Learning Representations (ICLR)*. 2020. URL: https://arxiv.org/abs/1904.09751.

[6] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, W. Chen. *LoRA: Low-Rank Adaptation of Large Language Models*. Techninė ataskaita. arXiv:2106.09685. 2021. URL: https://arxiv.org/abs/2106.09685.

[7] K. Yang, J. Deng. „Learning to Prove Theorems via Interacting with Proof Assistants“. Iš: *Proceedings of the 36th International Conference on Machine Learning (ICML)*. PMLR, 2019, puslapiai 6984–6994. https://doi.org/10.48550/arXiv.1905.09381.

[8] K. Yang, A. M. Swope, A. Gu, R. Chalamala, P. Song, S. Yu, S. Godil, R. Prenger, A. Anandkumar. „LeanDojo: Theorem Proving with Retrieval-Augmented Language Models“. Iš: *Advances in Neural Information Processing Systems*. Tomas 36. Curran Associates, Inc., 2023. URL: https://leandojo.org.

[9] A. Mahboubi, E. Tassi. *Mathematical Components*. Inria, 2022. URL: https://math-comp.github.io/mcb/.

[10] L. de Moura, S. Ullrich. „The Lean 4 Theorem Prover and Programming Language“. Iš: *Automated Deduction – CADE 28*. Tomas 12699. Lecture Notes in Computer Science. Springer, Cham, 2021, puslapiai 625–635. https://doi.org/10.1007/978-3-030-79876-5_37.

[11] Y. Nagashima, Y. He. *PaMpeR: Proof Method Recommendation System for Isabelle/HOL*. Techninė ataskaita. arXiv:1806.07239. 2018. URL: https://arxiv.org/abs/1806.07239.

[12] A. Newell, H. A. Simon. „The Logic Theory Machine: A Complex Information Processing System“. Iš: *IRE Transactions on Information Theory* 2.3 (1956), puslapiai 61–79. https://doi.org/10.1109/TIT.1956.1056797.

[13] S. Polu, I. Sutskever. *Generative Language Modeling for Automated Theorem Proving*. Techninė ataskaita. arXiv:2009.03393. 2020. URL: https://arxiv.org/abs/2009.03393.

[14] ProofWiki Contributors. *ProofWiki: The Online Compendium of Mathematical Proofs*. Accessed 2025. 2008. URL: https://proofwiki.org.

[15] R. Rafailov, A. Sharma, E. Mitchell, S. Ermon, C. D. Manning, C. Finn. „Direct Preference Optimization: Your Language Model is Secretly a Reward Model“. Iš: *Advances in Neural Information Processing Systems*. Tomas 36. Curran Associates, Inc., 2023, puslapiai 53728–53741. URL: https://arxiv.org/abs/2305.18290.

[16] B. Sadi. *Analysis of Artificial Neural Networks for Solving Reasoning Tasks — Source Code*. https://github.com/younes-sadi/Master-s_Project_2026. Accessed: May 2026. 2026.

[17] D. Silver, T. Hubert, J. Schrittwieser ir kiti. „A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play“. Iš: *Science* 362.6419 (2018), puslapiai 1140–1144. https://doi.org/10.1126/science.aar6404.

[18] G. Sutcliffe. „The TPTP Problem Library and Associated Infrastructure: From CNF to TH0, TPTP v6.4.0“. Iš: *Journal of Automated Reasoning* 59.4 (2017), puslapiai 483–502. https://doi.org/10.1007/s10817-017-9407-7.

[19] S. Welleck, J. Liu, R. Le Bras, H. Hajishirzi, Y. Choi, K. Cho. „NaturalProofs: Mathematical Theorem Proving in Natural Language“. Iš: *Advances in Neural Information Processing Systems*. Tomas 34. Curran Associates, Inc., 2021. https://doi.org/10.48550/arXiv.2104.01112.

[20] M. Wu, M. Norrish, C. Walder, A. Dezfouli. „TacticZero: Learning to Prove Theorems via Interacting with Proof Assistants“. Iš: *Advances in Neural Information Processing Systems*. Tomas 34. Curran Associates, Inc., 2021. URL: https://arxiv.org/abs/2102.09756.

[21] Z. Wu, S. Huang, Z. Zhou, H. Ying, J. Wang, D. Lin, K. Chen. *InternLM2.5-StepProver: Advancing Automated Theorem Proving via Expert Iteration on Large-Scale LEAN Problems*. Techninė ataskaita. arXiv:2410.15700. 2024. URL: https://arxiv.org/abs/2410.15700.

[22] H. Xin, Z. Z. Ren, J. Song ir kiti. *DeepSeek-Prover-V1.5: Harnessing Proof Assistant Feedback for Reinforcement Learning and Monte-Carlo Tree Search*. Techninė ataskaita. arXiv:2408.08152. 2024. URL: https://arxiv.org/abs/2408.08152.

[23] K. Zheng, J. M. Han, S. Polu. „miniF2F: a cross-system benchmark for formal olympiad-level mathematics“. Iš: *International Conference on Learning Representations (ICLR)*. 2022. URL: https://arxiv.org/abs/2109.00110.

A Complete Hyperparameter Configuration

Table 15 summarises all hyperparameters used across all experiments in this thesis.

15 Table *Complete Hyperparameter Configuration for All Experiments*

Parameter	**Phase I**	**Prop A**	**Prop B v3**
Beam width K	16	16	16
Timeout per theorem (s)	180	240	240
n-candidates per node	12	20	20
Max expanded nodes	5,000	8,000	8,000
Temperature τ	0.7	0.85	0.7
Top-p sampling	0.95	0.95	0.95
Max new tokens	64	64	64
LoRA rank r	—	8	8
LoRA α	—	16	16
LoRA target layers	—	wqkv, wo	wqkv, wo
Learning rate	—	2×10^{-4}	5×10^{-5}
Effective batch size	—	16	16
Training epochs	—	3	3
Training examples	—	100,000	10,000 (real)
Value threshold	—	0.60	—
Value weight	—	2.0	—
DPO β	—	—	0.1
max-refine-per-step	—	—	2
GPU	V100-32GB	V100-32GB	V100-32GB
LeanDojo version	1.6.0	1.6.0	1.6.0
PyTorch + CUDA	2.1.2+12.1	2.1.2+12.1	2.1.2+12.1