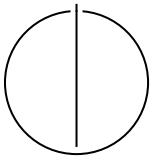
SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY — INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Improving Reasoning Capabilities of Large Language Models using the Automated Theorem Prover Isabelle

Christoph Waffler



SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY — INFORMATICS

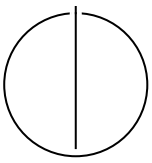
TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Improving Reasoning Capabilities of Large Language Models using the Automated Theorem Prover Isabelle

Verbesserung der Schlussfolgerungsfähigkeiten großer Sprachmodelle mithilfe des automatisierten Theorembeweisers Isabelle

Author:	Christoph Waffler
Examiner:	Prof. Dr. Georg Groh
Supervisor:	Lukas Ellinger
Submission Date:	29.09.2025



I confirm that this bachelor's thesis is my own work and I have documented all sources and material used.

Munich, 29.09.2025 Christoph Waffler

Acknowledgments

I would like to thank my supervisor Lukas Ellinger for offering me the opportunity to explore this challenging topic and for his continuous support and feedback throughout the project. His guidance helped me navigate both the theoretical and practical challenges of combining large language models with interactive theorem proving.

I am also thankful to the startup Prime Intellect for providing the computational resources necessary to carry out the experiments.

Lastly, I appreciate the encouraging support of my friends and family, who constantly encouraged me during the more demanding phases of this work.

Abstract

This thesis investigates the application of Monte Carlo Tree Search (MCTS) to enhance the reasoning capabilities of Large Language Models (LLMs) in automated theorem proving using Isabelle/HOL. We implement and compare two approaches: a beam search baseline and an MCTS-based method that incorporates learned value functions for guided proof search.

The beam search baseline achieves a 20% success rate on a subset of 50 problems from the miniF2F test dataset, demonstrating the feasibility of LLM-guided automated proof generation. Our MCTS implementation, evaluated on 244 problems, achieves a 12.7% success rate while showing superior computational efficiency with an average of 367.2 seconds per problem compared to beam search's 749 seconds.

Our analysis reveals that both approaches excel at problems requiring direct applications of standard Isabelle tactics but struggle with complex, multi-step proofs. Common failure modes include illegal proof commands, type system errors, and proof state mismatches, highlighting challenges in LLM understanding of formal proof systems.

Key contributions include a complete pipeline integrating LLMs with Isabelle/HOL through the QIsabelle interface, detailed performance analysis across mathematical problem categories, and insights into the trade-offs between beam search and MCTS approaches. While current limitations in training data and computational resources prevent optimal performance, our work provides a foundation for future research in test-time compute scaling for mathematical reasoning.

The results demonstrate that MCTS offers promising scalability and efficiency advantages for automated theorem proving, suggesting potential for significant improvements through enhanced value functions, better LLM training, and hybrid approaches combining statistical learning with symbolic reasoning.

Contents

1. Introduction

The intersection of large language models and formal mathematics represents one of the most promising frontiers in artificial intelligence research. As LLMs demonstrate increasingly sophisticated reasoning capabilities across diverse domains, their application to mathematical theorem proving offers both tremendous opportunities and significant challenges. This thesis explores how Monte Carlo Tree Search can enhance the reasoning capabilities of LLMs in the context of automated theorem proving using Isabelle/HOL.

1.1. Motivation

1.1.1. LLM Hallucinations

Large language models have shown remarkable capabilities in natural language understanding, code generation, and problem-solving. However, they are prone to hallucinations—generating plausible but incorrect or nonsensical outputs (Ji et al., 2023). This phenomenon is particularly problematic in domains requiring absolute correctness, such as mathematics and formal verification. Unlike natural language tasks where approximate answers may be acceptable, mathematical proofs demand logical rigor and complete accuracy.

The challenge of hallucinations becomes even more critical when LLMs are applied to formal theorem proving. A single incorrect step in a mathematical proof can invalidate the entire argument, leading to potentially catastrophic consequences in safety-critical applications such as software verification, cryptographic protocol analysis, and hardware design verification (Harrison, 2009a).

1.1.2. Especially in Mathematical ATP

Mathematical automated theorem proving presents unique challenges that exacerbate the hallucination problem. Unlike natural language, formal mathematics operates under strict syntactic and semantic rules. Every proof step must be verifiable by a proof assistant, leaving no room for approximation or interpretation (Nipkow et al., 2002a).

Traditional ATP systems have relied on symbolic methods, resolution-based theorem proving, and specialized algorithms for specific mathematical domains. While these approaches guarantee correctness, they often lack the flexibility and intuition that human mathematicians bring to problem-solving (Paulson, 1986). The emergence of LLMs offers the potential to bridge this gap by incorporating human-like mathematical intuition into formal proof systems.

However, the application of LLMs to theorem proving introduces new challenges. Models may generate syntactically correct but semantically meaningless proof steps, misunderstand the current proof state, or fail to maintain logical consistency across multiple proof steps (Wu et al., 2022). These issues highlight the need for robust search strategies that can guide LLMs toward correct proofs while avoiding hallucination traps.

1.2. Research Questions

This thesis addresses the following research questions:

1. **How can Monte Carlo Tree Search be effectively integrated with large language models to improve automated theorem proving performance?**
2. **What are the relative strengths and weaknesses of MCTS-based approaches compared to traditional beam search baselines in the context of Isabelle/HOL theorem proving?**
3. **How do different search strategies affect computational efficiency, success rates, and proof quality in automated theorem proving tasks?**
4. **What are the primary failure modes and limitations of current LLM-guided theorem proving approaches, and how can they be addressed?**

1.3. Contributions

This thesis makes several key contributions to the field of LLM-based automated theorem proving:

- **MCTS Implementation for Theorem Proving**: We develop a complete MCTS framework that integrates LLMs with Isabelle/HOL through the QIsabelle interface, incorporating learned value functions for guided proof search.

- **Comprehensive Baseline Comparison**: We implement and evaluate a beam search baseline that provides a reproducible benchmark for future research in LLM-based theorem proving.
- **Large-scale Evaluation**: We conduct extensive experiments on the miniF2F dataset, evaluating both approaches on a total of 244 mathematical theorems across different difficulty levels and domains.
- **Performance Analysis**: We provide detailed analysis of success rates, computational efficiency, proof quality, and failure patterns for both MCTS and beam search approaches.
- **Insights for Future Research**: Our analysis identifies key challenges and promising directions for improving LLM-based automated theorem proving, including hybrid approaches, enhanced training methodologies, and better integration with formal methods.

1.4. Thesis Structure

The remainder of this thesis is organized as follows:

Chapter 2: Theoretical Background provides the necessary foundation for understanding the intersection of large language models and automated theorem proving. We cover the fundamentals of Isabelle/HOL, reinforcement learning concepts relevant to MCTS, and the theoretical underpinnings of search algorithms in theorem proving.

Chapter 3: Related Work surveys existing research in automated theorem proving, large language models for mathematics, and search strategies for proof generation. We position our work within the broader research landscape and identify gaps that our research addresses.

Chapter 4: Methodology details our implementation of both the beam search baseline and the MCTS-based approach. We describe the system architecture, prompt engineering strategies, search algorithms, and integration with the Isabelle proof assistant.

Chapter 5: Results presents the experimental outcomes of our evaluation, including success rates, computational efficiency metrics, and detailed performance analysis across different problem types and mathematical domains.

Chapter 6: Discussion interprets the results, compares the performance of different approaches, analyzes failure patterns, and provides insights into the strengths and limitations of current LLM-based theorem proving methods.

Chapter 7: Conclusion and Future Work summarizes the key findings of our research, discusses limitations of the current implementation, and outlines promising directions for future research in this area.

1.5. Summary of Evaluations and Metrics

Our evaluation focuses on the miniF2F dataset (Zheng et al., 2022), a comprehensive benchmark for automated theorem proving that includes problems from mathematical competitions and textbooks. We evaluate both beam search and MCTS approaches using the following key metrics:

- **Success Rate**: Percentage of theorems successfully proved
- **Computational Efficiency**: Average time per problem and total runtime
- **Proof Quality**: Length and complexity of generated proofs
- **Search Behavior**: Exploration patterns, tree depth, and node utilization
- **Failure Analysis**: Categorization of common failure modes and error types

The beam search baseline was evaluated on 50 problems, achieving a 20% success rate with an average of 749 seconds per problem. The MCTS approach was evaluated on 244 problems, achieving a 12.7% success rate with improved computational efficiency at 367.2 seconds per problem. These results provide valuable insights into the trade-offs between different search strategies for LLM-based automated theorem proving.

This research contributes to the growing body of work on test-time compute scaling for mathematical reasoning and provides a foundation for future improvements in automated theorem proving systems.[1]

[1]The complete implementation of both beam search and MCTS approaches is available at: `https://github.com/chrissiwaffler/bsc-thesis-mcts-rl-isabelle`. Model weights and trained parameters are hosted on Hugging Face: `https://huggingface.co/chrissi/isabelle-mcts-rl`.

2. Theoretical Background

2.1. Automated Theorem Proving

In this section we introduce automated theorem proving as the foundation for our work on improving reasoning capabilities of large language models.

ATP is a subfield of automated reasoning that focuses on the development of computer programs capable of proving mathematical theorems without human intervention (Harrison, 2009b). The field emerged from the intersection of mathematical logic, computer science, and artificial intelligence, with foundational work dating back to the 1950s and 1960s, including Robinson's resolution principle (Robinson, 1965) and early symbolic logic approaches (No Author, 1973).

The core challenge in automated theorem proving lies in searching through the vast space of possible proof steps to construct a valid logical derivation from a set of axioms to a desired conclusion. This search problem is inherently difficult due to the exponential growth of the search space and the need for sophisticated heuristics to guide the proof search effectively.

Classical approaches to ATP include resolution-based methods, which work by converting logical formulas into clause normal form and applying the resolution inference rule systematically. More modern approaches incorporate advanced search strategies, lemma learning, and integration with decision procedures for specific mathematical domains (Giesl, 2010).

Recent advances have seen the emergence of neural theorem proving approaches, where large language models are trained on formal mathematical corpora to generate proofs in interactive theorem provers. Systems like DeepSeek-Prover (Xin et al., 2024a,b; Ren et al., 2025) and AlphaProof (Castelvecchi) have demonstrated the ability to solve complex mathematical problems by combining neural generation with formal verification. These approaches have been evaluated on benchmarks like MiniF2F (Zheng et al., 2022), showing significant improvements over traditional ATP methods.

The significance of automated theorem proving extends beyond pure mathematics, finding applications in software verification, hardware design, cryptographic protocol analysis, and formal methods. As computational power has increased and proof search algorithms have become more sophisticated, ATP systems have successfully proven increasingly complex theorems, some of which were previously beyond human

capability due to their computational complexity.

2.1.1. Isabelle/HOL

Isabelle/Higher-Order Logic (HOL) is a generic proof assistant for HOL[1] that provides a rich framework for formalized mathematics and program verification (Nipkow et al., 2002b; Nipkow and Klein, 2014). Developed primarily at the University of Cambridge and Technical University of Munich, Isabelle represents one of the most mature and widely-used interactive theorem provers in the research community.

The system is built on a small logical kernel that implements the inference rules of HOL, ensuring the soundness of all derived theorems through the Logic for Computable Functions (LCF) approach. This architecture provides strong guarantees about proof correctness while allowing for the development of sophisticated proof procedures and tactics on top of the trusted kernel.

Isabelle/HOL supports a rich type system with parametric polymorphism, type classes, and inductive and coinductive datatypes. The logic itself is classical and includes higher-order quantification, making it suitable for formalizing complex mathematical theories. The system provides extensive libraries covering areas such as analysis, algebra, topology, and computer science fundamentals.

A key strength of Isabelle/HOL lies in its structured proof language, Isar (Intelligible Semi-Automated Reasoning), which enables the writing of human-readable formal proofs[2] that closely mirror mathematical discourse (Wenzel, 1999). This approach bridges the gap between informal mathematical arguments and fully formal machine-checked proofs, making the system accessible to mathematicians while maintaining complete logical rigor.

The system's architecture supports both forward and backward reasoning, allowing users to construct proofs in a natural style while benefiting from automated proof search capabilities. Isabelle's meta-logical framework also enables the embedding of other logics and the development of domain-specific proof methods, making it a

[1] HOL allows quantification over functions and predicates, not just objects like in First-Order Logic (FOL). While FOL can express $\forall x.P(x)$, HOL also allows $\forall P.\exists x.P(x)$ - making it more expressive but undecidable.

[2] For example, proving that addition is commutative:
Tactical style: `apply (induction x) apply simp apply simp done`
Isar style:
```
proof (induction x)
case 0 thus "0 + y = y + 0" by simp
next
case (Suc x) thus "Suc x + y = y + Suc x" by simp
qed
```

versatile platform for diverse formal verification tasks.

2.1.2. Proof Automation in Isabelle

In this subsection we discuss the automation tools available in Isabelle that reduce manual proof effort.

Isabelle/HOL provides a sophisticated ecosystem of automated proof methods that significantly reduce the manual effort required for theorem proving. These automation tools range from simple simplification procedures to powerful integration with external automated theorem provers, creating a hybrid approach that combines the reliability of interactive theorem proving with the efficiency of automated reasoning.

The `simp` method (simplifier) forms the backbone of basic automation in Isabelle. It applies equational rewriting rules and conditional rewrite rules to simplify proof goals automatically. The simplifier maintains a dynamic simpset of rewrite rules and can perform contextual rewriting, where assumptions from the current proof context are used as additional rewrite rules. This method is particularly effective for routine algebraic manipulations and definitional unfolding.

The `auto` method represents a more powerful automation tool that combines multiple proof techniques including simplification, classical reasoning, and limited quantifier handling. It applies the simplifier repeatedly while also using introduction and elimination rules for logical connectives, making it capable of solving many routine proof obligations without user guidance. The method employs a best-first search strategy and can backtrack when proof attempts fail.

Sledgehammer (Paulson and Blanchette, 2012) represents the most sophisticated automation tool in Isabelle's arsenal, providing seamless integration with external ATPs and Satisfiability Modulo Theories (SMT) solvers. When invoked, Sledgehammer translates the current proof goal and relevant facts from Isabelle's context into first-order logic, dispatches multiple external provers in parallel, and attempts to reconstruct successful external proofs as native Isabelle proofs.

The strength of Sledgehammer lies in its ability to find proofs that would be difficult or time-consuming for users to construct manually, particularly those involving complex combinations of arithmetic, set theory, and logical reasoning. The tool maintains sophisticated relevance filtering mechanisms to select appropriate lemmas and facts from Isabelle's extensive libraries, addressing the challenge of premise selection in automated reasoning.

Recent work has extended Isabelle's automation capabilities through neural approaches. Systems like ProofAug (Liu et al., 2025) integrate fine-grained proof structure analysis with tree search methods to enhance neural theorem proving efficiency. These approaches combine the reliability of Isabelle's formal verification with the pattern

recognition capabilities of large language models trained on mathematical corpora.

This multi-layered approach to proof automation makes Isabelle particularly suitable for large-scale formalization projects, where the combination of reliable foundations, expressive logic, and powerful automation enables both novice and expert users to construct complex formal proofs efficiently.

2.2. Reinforcement Learning

RL is a machine learning paradigm where an agent learns to make sequential decisions by interacting with an environment to maximize cumulative reward (Sutton and Barto, 2020). Unlike supervised learning, RL agents learn from trial and error, receiving feedback in the form of rewards or penalties based on their actions. This framework has proven particularly effective for complex sequential decision-making problems where the optimal strategy is not immediately apparent.

The fundamental RL framework consists of an agent, an environment, states, actions, and rewards. At each time step t, the agent observes the current state $s_t \in \mathcal{S}$, selects an action $a_t \in \mathcal{A}(s_t)$, receives a reward $r_{t+1} \in \mathbb{R}$, and transitions to a new state s_{t+1} according to the environment dynamics. The agent's goal is to learn a policy $\pi(a|s)$ that maximizes the expected cumulative reward, often referred to as the return.

In the context of automated theorem proving, reinforcement learning provides a natural framework for learning proof search strategies. The state space corresponds to partial proof states, actions represent proof steps or tactics, and rewards can be designed to encourage progress toward valid proofs. This approach has shown particular promise when combined with neural networks and tree search algorithms.

2.2.1. Value Functions

Value functions are fundamental concepts in reinforcement learning that estimate the expected cumulative reward an agent can obtain from a given state or state-action pair. These functions provide the theoretical foundation for many RL algorithms and enable agents to make informed decisions about which actions to take.

The state value function $V^\pi(s)$ under policy π is defined as the expected return when starting from state s and following policy π:

$$V^\pi(s) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} \mid s_0 = s \right] \tag{2.1}$$

where $\gamma \in [0, 1]$ is the discount factor that determines the importance of future rewards relative to immediate rewards. Similarly, the action value function $Q^\pi(s, a)$

represents the expected return when taking action a in state s and then following policy π:

$$Q^\pi(s,a) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} \mid s_0 = s, a_0 = a \right] \tag{2.2}$$

The Bellman equations (Bellman, 1984) provide recursive relationships for these value functions. For the state value function:

$$V^\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma V^\pi(s')] \tag{2.3}$$

And for the action value function:

$$Q^\pi(s,a) = \sum_{s',r} p(s',r|s,a) \left[r + \gamma \sum_{a'} \pi(a'|s') Q^\pi(s',a') \right] \tag{2.4}$$

The optimal value functions $V^*(s)$ and $Q^*(s,a)$ represent the maximum expected return achievable from any state or state-action pair. These satisfy the Bellman optimality equations and form the basis for dynamic programming algorithms like value iteration and policy iteration. Q-learning (Dayan and Watkins, 1992) provides a model-free approach to learning optimal action value functions through Temporal Difference (TD) updates.

In neural network-based approaches, value functions are approximated using function approximators, typically deep neural networks. This enables the handling of large or continuous state spaces that would be intractable for tabular methods. The approximation introduces additional challenges related to generalization and stability, which are addressed through techniques like experience replay and target networks. TD learning (Sutton, 1988) provides the foundation for many modern RL algorithms by learning value functions through bootstrapping from incomplete episodes.

2.2.2. Self Play

Self play is a training paradigm in RL where an agent learns by playing against copies of itself, gradually improving through competition with increasingly sophisticated opponents. This approach has proven particularly effective in two-player zero-sum games and has been extended to single-agent domains where the agent can be viewed as playing against the environment or previous versions of itself. The concept dates back to Samuel's pioneering work on checkers (Samuel, 1959), where a program learned to play by competing against earlier versions of itself.

The fundamental insight behind self play is that as the agent improves, it faces increasingly challenging opponents, creating a natural curriculum that drives continued learning. This process can lead to the emergence of sophisticated strategies without requiring human expert knowledge or pre-existing strong opponents. The combination of self-play with temporal difference learning proved particularly powerful in Tesauro's TD-Gammon system (Tesauro et al., 1995), which achieved master-level backgammon play through self-competition (Silver et al., 2018).

In the context of AlphaGo and subsequent AlphaZero systems (Silver et al., 2018, 2017), self play was combined with Monte Carlo Tree Search to achieve superhuman performance in board games. The agent would play games against itself, using the outcomes to improve both its value network (which estimates position strength) and policy network (which suggests good moves). This iterative process of self-improvement through competition forms the core of the self play paradigm.

For theorem proving applications, self play can be adapted in several ways. One approach involves having the agent learn to both propose proof steps and critique its own proposals, creating an adversarial dynamic that improves both capabilities. Another approach uses self play to generate training data, where the agent attempts proofs and learns from both successful and failed attempts.

The effectiveness of self play depends critically on the reward structure and the exploration strategy. In games, the reward is naturally defined by winning or losing, but in theorem proving, designing appropriate intermediate rewards that guide learning toward valid proofs while avoiding local optima is more challenging. Additionally, ensuring sufficient exploration to prevent the agent from converging to suboptimal strategies requires careful balancing of exploitation and exploration.

Recent work has extended self play concepts to language models and reasoning tasks, where models learn to generate and evaluate their own reasoning chains. This meta-cognitive approach, where the model learns to reason about its own reasoning, has shown promise in improving the quality and reliability of generated solutions.

2.2.3. Monte Carlo Tree Search (MCTS)

Monte Carlo Tree Search (MCTS) is a best-first search algorithm that combines the precision of tree search with the generality of random sampling to make decisions in large state spaces (Browne et al., 2012). MCTS has achieved remarkable success in game playing, most notably in Go through the AlphaGo system, and has since been applied to various domains including automated reasoning and theorem proving.

The MCTS algorithm operates through four main phases repeated iteratively: Selection, Expansion, Simulation, and Backpropagation. During Selection, the algorithm traverses the existing search tree from the root to a leaf node, using a selection policy

that balances exploitation of promising paths with exploration of less-visited branches. The Upper Confidence Bound 1 (UCB1) formula is commonly used for this selection (Kocsis and Szepesvári, 2006):

$$\text{UCB1}(s,a) = Q(s,a) + C\sqrt{\frac{\ln N(s)}{N(s,a)}} \tag{2.5}$$

where $Q(s,a)$ is the average reward from action a in state s, $N(s)$ is the number of times state s has been visited, $N(s,a)$ is the number of times action a has been taken from state s, and C is the exploration constant that controls the balance between exploitation and exploration.

In the Expansion phase, one or more child nodes are added to the search tree to expand the explored state space. The Selection phase is followed by Simulation (or rollout), where the algorithm simulates a complete episode from the newly expanded node to a terminal state using a rollout policy, which may be random or guided by heuristics. Finally, in the Backpropagation phase, the result of the simulation is propagated back up the tree, updating the statistics of all nodes along the path.

The strength of MCTS lies in its ability to focus computational resources on the most promising parts of the search space while maintaining theoretical guarantees about convergence to the optimal policy given sufficient time. The algorithm is particularly well-suited to domains with large branching factors where traditional minimax search becomes intractable.

In the context of neural networks, MCTS can be enhanced by replacing random rollouts with value network evaluations and using policy networks to guide both the tree policy and rollout policy. This combination, exemplified in AlphaZero (Silver et al., 2018, 2017), creates a powerful synergy between neural pattern recognition and systematic tree search.

For automated theorem proving, MCTS provides a natural framework for proof search where states represent partial proofs, actions correspond to proof tactics or rule applications, and the search tree explores different proof strategies. For instance, when proving a theorem like "if A implies B and B implies C, then A implies C," MCTS might explore different sequences of logical rules, with each node in the search tree representing a different partial proof state and the algorithm evaluating which proof steps are most promising based on learned value functions.

The challenge lies in designing appropriate value functions and rollout policies that can effectively evaluate partial proof states and guide the search toward valid completions. The integration of MCTS with language models trained on mathematical text offers a promising approach to this challenge, leveraging both the systematic exploration of MCTS and the pattern recognition capabilities of neural networks trained

on large mathematical corpora.

2.3. Large Language Models

LLMs represent a transformative advancement in artificial intelligence, demonstrating remarkable capabilities in natural language understanding, generation, and reasoning. Built upon the Transformer architecture (Vaswani et al., 2017), these models process and generate text through self-attention mechanisms that capture long-range dependencies and contextual relationships.

2.3.1. Transformer Architecture

The Transformer architecture, introduced in the seminal "Attention is All You Need" paper (Vaswani et al., 2017), revolutionized natural language processing by replacing recurrent neural networks with a fully attention-based approach. Unlike previous architectures that processed sequences sequentially, Transformers enable parallel computation across all input positions, dramatically improving training efficiency and scalability.

At its core, the Transformer employs self-attention mechanisms that allow each token in a sequence to attend to all other tokens, computing weighted representations based on their contextual relevance. This is extended through multi-head attention, which performs multiple parallel attention operations with different learned projections, capturing diverse types of relationships within the input. Positional encodings are added to inject sequential information, since the attention mechanism itself is permutation-invariant.

The architecture consists of stacked encoder and decoder layers, each containing multi-head attention and feed-forward neural networks. This design enables Transformers to model long-range dependencies effectively and has become the foundation for virtually all modern large language models.

The remarkable success of Transformers is further explained by neural scaling laws (Kaplan et al., 2020), which demonstrate that model performance improves predictably with increased model size, training data, and computational resources. These scaling laws have driven the development of increasingly large models, showing that larger Transformers trained on more data consistently achieve better performance across diverse tasks (Team et al., 2025). Beyond scaling, reinforcement learning techniques, as discussed in Section 2.2, are increasingly employed to further enhance the reasoning capabilities of these models through targeted training paradigms.

2.3.2. Finetuning Methods

Finetuning adapts pre-trained language models to specific tasks or domains through various training paradigms:

Supervised Finetuning (SFT) involves training the model on task-specific examples with ground-truth outputs (Ouyang et al., 2022). For theorem proving, this typically consists of pairs of problem statements and corresponding proof steps or complete proofs. SFT teaches the model to generate appropriate responses in the desired format and domain.

Proximal Policy Optimization (PPO) (Schulman et al., 2017) is a RL algorithm that optimizes policies using a clipped objective function to prevent large policy updates. In the context of LLMs, PPO can be used to align model outputs with human preferences or task-specific reward functions.

Group Relative Policy Optimization (GRPO) represents a more recent approach used in systems like DeepSeek Prover v1.5 (Xin et al., 2024b). GRPO was first introduced in the DeepSeekMath work (Shao et al., 2024) and optimizes policies relative to groups of responses, encouraging the model to generate outputs that are preferred over alternatives. GRPO has shown particular effectiveness in reasoning tasks where multiple solution paths exist, and offers advantages over traditional PPO by eliminating the need for a separate value network. This simplification reduces memory requirements and training instability while maintaining competitive performance, making it particularly suitable for large-scale language model training where computational efficiency is crucial.

2.3.3. Sampling Strategies

Sampling strategies determine how LLMs generate text from their probability distributions, representing a crucial trade-off between exploration and exploitation (Holtzman et al., 2020):

Greedy sampling always selects the token with the highest probability at each step. This approach maximizes immediate likelihood but can lead to repetitive or suboptimal outputs, especially in complex reasoning tasks.

Stochastic sampling introduces randomness through temperature scaling, which controls the exploration-exploitation balance:

$$P(y_t|y_{<t}) = \frac{\exp(\text{logits}(y_t)/T)}{\sum_{y'} \exp(\text{logits}(y')/T)} \tag{2.6}$$

where T is the temperature parameter. Higher temperatures increase exploration by flattening the probability distribution, while lower temperatures emphasize exploitation

by sharpening it.

Nucleus sampling (top-p) restricts sampling to the smallest set of tokens whose cumulative probability mass exceeds a threshold p. This approach dynamically adapts the number of candidate tokens based on the probability distribution, often producing more diverse and coherent outputs than fixed-k sampling.

Beam search (Sutskever et al., 2014; Holtzman et al., 2020) maintains multiple candidate sequences (beams) simultaneously, expanding each beam with the most likely next tokens. While this approach balances between greedy search and exhaustive search, research has shown that maximization-based decoding methods like beam search often lead to text degeneration—producing bland, incoherent, or repetitive outputs—making stochastic sampling with nucleus sampling generally preferable for most generation tasks.

The choice of sampling strategy has significant implications for theorem proving. During MCTS exploration, higher temperatures encourage the discovery of diverse proof strategies, while final proof generation benefits from lower temperatures to ensure logical consistency and correctness. This distinction provides a natural motivation for using beam search as a baseline while employing sampling-based exploration in MCTS.

2.3.4. Reasoning Capabilities

Enhancing the reasoning capabilities of LLMs has been a major focus of recent research, leading to several approaches that improve performance on complex problem-solving tasks:

Chain of Thought (CoT) (Wei et al.) encourages models to generate step-by-step reasoning processes before arriving at final answers. This approach has been further enhanced through self-consistency techniques that sample multiple reasoning paths and select the most consistent answer (Wang et al., 2023). CoT has been particularly effective for mathematical reasoning, as it breaks down complex problems into manageable subproblems and allows for intermediate verification.

Tree of Thoughts (Yao et al., 2023) extends the CoT concept by exploring multiple reasoning paths simultaneously, maintaining a tree structure of possible solution trajectories. This method allows for backtracking when encountering dead ends and selecting the most promising path based on intermediate evaluations.

Tree search methods for LLMs combine systematic exploration with neural network guidance, creating hybrid approaches that leverage both the pattern recognition capabilities of LLMs and the systematic search of traditional algorithms. Recent work has explored various tree search strategies for enhancing LLM reasoning, including Monte Carlo Tree Search for comprehensive heuristic exploration (Zheng et al., 2025), process reward-guided tree search for self-training (Zhang et al., 2024), and fine-grained proof

structure analysis for efficient theorem proving (Liu et al., 2025). These methods have shown particular promise in domains with large state spaces and multiple solution paths, such as theorem proving.

The integration of tree search with LLMs represents a natural evolution of reasoning capabilities, where the language model provides domain knowledge and heuristic guidance while the search algorithm ensures systematic exploration of the solution space. This synergy forms the foundation for the MCTS-based approach explored in this work, bridging the gap between neural pattern recognition and algorithmic reasoning.

3. Related Work

3.1. AlphaZero

AlphaZero (Silver et al., 2018) represents a landmark achievement in artificial intelligence, demonstrating that a single algorithm can achieve superhuman performance across multiple board games through self-play and RL. The system combines MCTS with deep neural networks, creating a powerful synergy between pattern recognition and systematic search.

The AlphaZero architecture consists of three key components: a deep neural network that takes board positions as input and outputs both a policy (probability distribution over moves) and a value (expected outcome), a Monte Carlo Tree Search algorithm that uses the neural network to guide its search, and a reinforcement learning training procedure that improves the network through self-play.

The training process follows an iterative loop where the current neural network guides MCTS to generate games, the outcomes of these games are used to train a new neural network, and the new network replaces the old one if it demonstrates superior performance. This self-improvement process allows AlphaZero to discover sophisticated strategies without human knowledge beyond the basic game rules.

AlphaZero's success has inspired numerous applications beyond board games, particularly in domains requiring sequential decision-making and search. The core innovations that make AlphaZero relevant to automated theorem proving include:

- **Neural network-guided search**: Using neural networks to evaluate states and guide search decisions
- **Self-play training**: Generating training data through autonomous exploration and competition
- **Value function learning**: Estimating the probability of success from partial states
- **Policy improvement**: Refining action selection strategies through experience

In the context of theorem proving, AlphaZero's approach can be adapted by treating proof states as board positions, proof tactics as moves, and theorem validity as the win

condition. This analogy provides a foundation for applying MCTS and reinforcement learning to automated reasoning tasks.

3.2. DeepSeek Prover v1.5

DeepSeek Prover v1.5 (Xin et al., 2024a) represents a state-of-the-art approach to automated theorem proving that leverages LLMs and RL techniques. Building upon earlier work in neural theorem proving, DeepSeek Prover demonstrates significant improvements in proof completion rates across multiple benchmark datasets.

The system employs a sophisticated architecture that combines several advanced techniques:

- **Large-scale pre-training**: Training on massive mathematical corpora to develop deep understanding of mathematical language and reasoning patterns
- **GRPO**: A RL approach that optimizes policies relative to groups of candidate responses
- **Multi-stage training**: Combining supervised finetuning with RL to refine proof generation capabilities
- **Iterative refinement**: Using proof verification and feedback loops to improve generated proofs

DeepSeek Prover's training methodology involves several key stages. First, the model undergoes supervised finetuning on high-quality proof examples to learn basic proof patterns and Isabelle/HOL syntax. This is followed by reinforcement learning using GRPO, where the model receives rewards based on proof correctness, efficiency, and novelty.

A distinctive feature of DeepSeek Prover is its approach to handling the exploration-exploitation trade-off in proof search. The system uses adaptive temperature scaling during generation, balancing between exploring novel proof strategies and exploiting known successful patterns. This adaptability is particularly important for theorem proving, where different problems may require different levels of exploration.

The system also incorporates sophisticated proof verification and validation mechanisms. Generated proofs are checked for logical correctness using Isabelle's proof kernel, with feedback from failed verification attempts used to improve future performance. This closed-loop training approach ensures that the system learns from its mistakes and gradually improves its proof generation capabilities.

DeepSeek Prover's performance on benchmark datasets like miniF2F demonstrates the effectiveness of combining large-scale pre-training with targeted reinforcement learning. The system's success highlights the importance of both broad mathematical knowledge and specialized proof-generation skills in automated theorem proving.

3.3. ProofAug

ProofAug (Liu et al., 2025) introduces an innovative approach to automated theorem proving by focusing on proof augmentation and lemma generation. The system addresses a fundamental challenge in theorem proving: the need for intermediate lemmas that bridge the gap between given assumptions and desired conclusions.

The core insight behind ProofAug is that many difficult theorems cannot be proven directly from first principles but require the discovery and application of auxiliary lemmas. ProofAug automates this process by training a neural network to both generate proofs and identify useful intermediate lemmas that facilitate proof construction.

The ProofAug architecture consists of several key components:

- **Lemma generation network**: Predicts potentially useful lemmas based on the current proof state and goal
- **Proof verification module**: Checks the validity of generated proofs and lemmas using formal verification tools
- **Relevance scoring**: Evaluates the utility of generated lemmas for completing the proof
- **Iterative refinement**: Uses feedback from proof attempts to improve lemma generation

The training process for ProofAug involves a curriculum learning approach where the system starts with simple theorems that don't require lemmas and gradually progresses to more complex problems requiring multiple intermediate steps. This curriculum helps the system develop both basic proof skills and the ability to recognize when lemmas are needed.

ProofAug's lemma generation strategy is particularly noteworthy. The system uses a combination of pattern matching, analogy to known proofs, and abductive reasoning to suggest lemmas that might be useful for the current proof context. The generated lemmas are then validated both for logical correctness and for their potential contribution to completing the proof.

One of the key innovations of ProofAug is its approach to handling the combinatorial explosion of possible lemmas. Rather than generating all possible lemmas, the system uses heuristic guidance to focus on lemmas that are likely to be relevant based on the structure of the current proof state and the target theorem.

The system has demonstrated success on several theorem proving benchmarks, particularly for problems that require non-obvious intermediate steps. ProofAug's performance suggests that automated lemma generation is a crucial capability for advancing automated theorem proving beyond problems that can be solved through direct proof search.

3.4. ReST-MCTS*

ReST-MCTS* (Zhang et al., 2024) represents a sophisticated integration of RL, self-training, and MCTS for automated theorem proving. The system builds upon the success of tree search methods in games and adapts them to the unique challenges of mathematical reasoning.

The ReST-MCTS* architecture combines several advanced techniques:

- **Reinforced Self-Training (ReST)**: An iterative training process where the model generates proofs, verifies them, and uses successful proofs as additional training data
- **MCTS**: Systematic exploration of proof spaces with neural network guidance
- **Proof state representation**: Sophisticated encoding of partial proofs for neural network processing
- **Adaptive search strategies**: Dynamic adjustment of search parameters based on problem difficulty

The ReST component of the system follows a bootstrapping approach where the model starts with basic proof capabilities and gradually improves through self-generated training data. This process involves several stages:

1. **Initial proof generation**: The model attempts to prove theorems using its current capabilities
2. **Proof verification**: Generated proofs are checked for correctness using formal verification tools
3. **Data augmentation**: Successful proofs are added to the training dataset

4. **Model retraining**: The model is retrained on the augmented dataset
5. **Iteration**: The process repeats with the improved model

The MCTS component in ReST-MCTS* is specifically adapted for theorem proving. Unlike traditional MCTS used in games, the theorem proving version must handle:

- **Variable branching factor**: The number of possible proof tactics varies significantly between states
- **Long-term dependencies**: Proof steps may have consequences that only become apparent much later
- **Logical constraints**: Not all sequences of tactics are valid or meaningful
- **Partial credit**: Some proof attempts may be partially correct even if they don't reach the final goal

ReST-MCTS* addresses these challenges through several innovations. The system uses a sophisticated proof state representation that captures both the current logical context and the history of proof steps. The neural network guiding the MCTS search is trained to estimate both the probability of success from a given state and the value of different proof tactics.

The system also incorporates adaptive search strategies that adjust the balance between exploration and exploitation based on the perceived difficulty of the theorem. For straightforward theorems, the system may focus more on exploitation to find efficient proofs, while for difficult theorems, it may emphasize exploration to discover novel proof strategies.

ReST-MCTS* has demonstrated strong performance on several theorem proving benchmarks, showing that the combination of self-training and guided tree search can effectively tackle complex mathematical reasoning problems. The system's success provides important insights for the development of more advanced automated theorem provers.

3.5. Positioning of Our Work

Our work synthesizes MCTS with large language models for Isabelle/HOL theorem proving, building on existing approaches while introducing key innovations. Unlike AlphaZero's game-focused approach, we adapt MCTS for theorem proving with domain-specific state representations and formal verification integration. While DeepSeek Prover emphasizes sequential proof generation, our MCTS approach enables

parallel exploration of multiple proof strategies with dedicated critic model feedback. Compared to ProofAug's lemma generation focus, we systematically explore proof spaces, and unlike ReST-MCTS*, we provide comprehensive beam search baseline comparisons. Our contribution lies in demonstrating MCTS effectiveness for LLM-based theorem proving through systematic evaluation and Isabelle/HOL integration.

4. Methodology

4.1. Evaluation Dataset

The miniF2F (mini Formal to Formal) dataset (Ahn et al., 2024) serves as the primary evaluation benchmark for our automated theorem proving system. This dataset has become a standard in the field for evaluating neural theorem provers due to its carefully curated selection of problems and its focus on real mathematical content. Although the problems originate from high school mathematics competitions, they provide a valuable benchmark for tracking the current progress of AI in mathematical reasoning and theorem proving.

For our experiments, we use the Facebook Research fork of miniF2F, which represents an enhanced version of the original OpenAI repository. This fork includes additional data and many formal statement fixes, providing improved reliability and coverage compared to the original release. The Facebook Research version maintains the same problem structure and evaluation splits while offering better maintainability and ongoing updates to the dataset (Zheng et al., 2022).

miniF2F consists of 488 formal mathematics problems (244 for validation and 244 for testing), drawn from diverse sources including olympiads such as the AMC, AIME, and IMO, as well as the MATH dataset and custom problems in algebra, number theory, and induction. These problems have been formalized into theorem proving languages like Lean and Isabelle/HOL, ensuring they are suitable for automated reasoning systems. The problems cover a wide range of mathematical topics:

- **Algebra**: Equations, inequalities, polynomials, and algebraic manipulations
- **Number theory**: Divisibility, prime numbers, modular arithmetic, and Diophantine equations
- **Geometry**: Geometric proofs, coordinate geometry, and trigonometric identities
- **Calculus and Analysis**: Limits, derivatives, integrals, and real analysis concepts
- **Combinatorics**: Counting principles, probability, and combinatorial identities

The dataset is divided into two main subsets:

- **miniF2F-valid**: 244 problems used for validation and hyperparameter tuning
- **miniF2F-test**: 244 problems held out for final evaluation

This split ensures that evaluation results reflect genuine generalization performance rather than overfitting to specific problem types.

miniF2F exhibits several important characteristics that make it suitable for evaluating ATPs:

- **Variable difficulty**: Problems range from relatively straightforward to quite challenging, providing a good test of system capabilities across different difficulty levels
- **Real mathematical content**: Unlike synthetic datasets, miniF2F contains authentic mathematical problems that require genuine reasoning
- **Domain diversity**: The coverage of multiple mathematical areas ensures that systems must develop general reasoning capabilities rather than specializing in narrow domains
- **Standardized evaluation**: The dataset's widespread adoption enables meaningful comparison between different approaches and systems

The problems in miniF2F typically require multi-step reasoning and often involve non-obvious insights or clever applications of mathematical concepts. This complexity makes the dataset an excellent benchmark for evaluating the sophisticated reasoning capabilities needed for advanced automated theorem proving.[1]

The miniF2F dataset's combination of authentic mathematical content, variable difficulty, and standardized evaluation makes it an ideal benchmark for our work. By evaluating our MCTS-based approach against this dataset, we can demonstrate both the effectiveness of our method and its applicability to real-world mathematical reasoning tasks.

4.2. Connection to the Isabelle Theorem Prover

To integrate automated reasoning capabilities into the pipeline, a bidirectional communication channel with the Isabelle theorem prover is required. This allows the system

[1] The complete implementation of both beam search and MCTS approaches is available at: `https://github.com/chrissiwaffler/bsc-thesis-mcts-rl-isabelle`. Model weights and trained parameters are hosted on Hugging Face: `https://huggingface.co/chrissi/isabelle-mcts-rl`.

to send valid proof commands based on the current proof state and receive feedback, such as proof completion status or remaining subgoals.

A well-known Isabelle client is the Portal to Isabelle (PISA) (Jiang et al., 2021), which, however, requires manual compilation and setup. A more lightweight alternative is QIsabelle (Wrochna, 2024), a Scala-based server that exposes an HTTP API for spawning Isabelle processes, accompanied by a Python client library for interaction.

For this thesis, a modified fork of QIsabelle (Wrochna, 2024) was utilized, with enhancements for Docker instance isolation and parallelized multi-port usage.[2]

For a deeper theoretical understanding of Isabelle's proof system, see **??**.

4.3. Baseline Implementation: Beam Search

Beam search serves as our baseline approach for automated theorem proving, providing a systematic reference point for evaluating our MCTS-based system. This implementation leverages *test-time compute* to explore multiple solution paths simultaneously, maintaining a fixed number of most promising candidates at each search level. The approach combines LLM-based tactic generation with automated theorem proving tools and a dedicated critic model for state evaluation, representing a neuro-symbolic approach to formal reasoning (Trinh et al., 2024). By increasing computational investment during inference, beam search enables broader exploration of the proof search space compared to greedy decoding methods.

4.3.1. Algorithm Overview

Our beam search implementation maintains a *beam* of candidate partial proofs, systematically expanding each candidate with multiple possible next steps and selecting the top-k candidates based on critic model evaluation. The algorithm operates as follows:

1. **Initialization**: Begin with the initial proof state containing the theorem statement and available assumptions
2. **Candidate generation**: For each beam candidate, generate multiple proof tactics through complementary approaches: automated Isabelle tactics, self-ask reasoning, and temperature-based LLM sampling with values $\{0.3, 0.6, 0.9, 1.2\}$
3. **Execution and evaluation**: Execute each generated tactic in Isabelle/HOL and evaluate the resulting proof states using a dedicated critic model that assesses tactical appropriateness and proof progress

[2]The modified codebase is available at https://github.com/chrissiwaffler/qisabelle. Changes include improved containerization and scalability for high-throughput proof scenarios.

4. **Selection**: Retain the top-k candidates with the highest critic scores to form the next beam, where k is determined by the beam width parameter (typically $k = 5$ in our experiments)

5. **Termination**: Repeat the process until a complete proof is found or the maximum search depth (typically 20 steps) is reached

The beam width parameter k fundamentally controls the exploration-exploitation trade-off. Larger values enable broader exploration of proof strategies at increased computational cost, while smaller values focus search on the most promising candidates. Performance is evaluated using proof success rate, average proof length, and critic score distributions, establishing quantitative baselines for MCTS comparison.

4.3.2. Proof Step Generation

Our beam search implementation employs multiple complementary approaches for generating proof tactics:

Automated Tactics

The system prioritizes deterministic automation before LLM-based generation:

- **Quick tactics**: High-success-rate automation commands including `apply simp`, `apply auto`, `apply blast`, `apply arith`, and `apply linarith`
- **Sledgehammer integration**: Asynchronous invocation of Isabelle's external ATP that searches for proofs using multiple backend provers (Vampire, E, SPASS, etc.)
- **Assumption-aware tactics**: Contextual tactics that explicitly utilize theorem assumptions through patterns like `using assms apply auto` and `apply (simp add: assms)`

Self-Ask Reasoning

The system employs a self-ask reasoning paradigm (Wei et al.) where the language model analyzes the current proof state and generates tactic suggestions through structured reasoning. The self-ask mechanism considers multiple contextual factors:

- **Proof mode awareness**: Differentiates between Isabelle's *state mode* (structured proofs with `have`/`show`) and *prove mode* (tactical proofs with `apply`)
- **Proof history**: Incorporates the sequence of previously successful tactics to maintain proof coherence

- **Available resources**: Leverages accessible lemmas, assumptions, and theorem-specific context
- **Failure analysis**: Examines error patterns from failed tactics to avoid repeated mistakes and adapt strategy

The self-ask reasoning uses a sophisticated prompt template structure that enforces step-by-step thinking before tactic generation. The template dynamically adapts to proof modes and incorporates failure context, as detailed in Section A.1. A concrete example showing the populated template and expected reasoning output is provided in Subsection A.1.3.

Temperature-Based Generation

The system generates diverse candidate tactics using temperature-based sampling with values $\{0.3, 0.6, 0.9, 1.2\}$. Lower temperatures produce more deterministic outputs suitable for straightforward proof steps, while higher temperatures encourage creative exploration for complex reasoning challenges. Crucially, the system performs two generation passes for each temperature: one with self-ask context and one without, providing coverage of both analytical and creative reasoning approaches.

4.3.3. Scoring and Evaluation

Each candidate proof state is evaluated using a dedicated critic model that provides contextual assessment of tactic quality. The critic model employs a multi-criteria evaluation framework:

- **Tactic appropriateness**: Evaluates whether the generated tactic matches the current proof mode (state vs. prove) and aligns with the subgoal structure
- **Progress assessment**: Measures advancement toward proof completion. This includes subgoal reduction and error resolution
- **Context integration**: Considers proof history, previously failed attempts, and available assumptions. This supports tactical coherence throughout the proof process
- **Strategic value**: Assesses the long-term potential of tactics. Some tactics may not provide immediate progress but enable important future proof steps

The critic model returns a normalized score in the range $[0.0, 1.0]$. Higher values indicate higher-quality proof states. This evaluation function serves as the mechanism

for beam selection. It allocates computational resources to the highest-scoring search paths. The scoring mechanism operates independently for each state. It evaluates local tactical quality without global search context.

4.3.4. Advanced Features

Our beam search implementation incorporates several mechanisms to enhance proof search effectiveness:

- **Failure pattern detection**: Systematically identifies repeated error patterns (minimum 3 occurrences). It triggers strategic reflection to adapt the search approach
- **Strategic guidance**: Employs LLM-based reflection on failure patterns. This generates context-aware tactical suggestions that address identified issues
- **Mode-aware generation**: Implements distinct prompting strategies for Isabelle's state mode (structured proofs with `have`/`show`) versus prove mode (tactical proofs with `apply`). This ensures tactic validity
- **Automatic proof completion**: Detects terminal states containing "no subgoals". It automatically applies the `done` tactic to complete proofs
- **Fallback strategies**: When proof methods fail in state mode, it attempts to execute statements without proofs. These serve as intermediate lemmas, maintaining proof progress

4.3.5. Implementation Details

The system integrates with Isabelle/HOL through a modified QIsabelle interface (see **??**). It provides essential capabilities for automated theorem proving:

- **Tactic execution**: Applies generated tactics to proof states. It returns updated states with success/failure status
- **Proof verification**: Validates logical correctness of generated proof sequences. This occurs through Isabelle's kernel
- **Error recovery**: Handles tactic failures gracefully. It extracts error messages for pattern analysis. This enables alternative strategy exploration
- **Theory management**: Supports dynamic loading of required mathematical theories. It ensures proper session isolation between different theorem proving tasks

The implementation uses asynchronous processing for Sledgehammer invocations. This enables parallel exploration of automated and neural-based tactics. Session management supports both individual session creation per theorem and session reuse across multiple theorems. This approach improves efficiency.

4.3.6. Limitations of Beam Search

Despite providing a baseline with multiple generation strategies and critic-based evaluation, our beam search implementation exhibits fundamental limitations that motivate the MCTS-based approach:

- **Fixed exploration breadth**: The beam width parameter k constrains search breadth uniformly across all problems, preventing adaptation to theorems requiring different levels of exploration intensity
- **Irrevocable pruning**: Once candidates are eliminated from the beam, they cannot be revisited. This is particularly problematic in theorem proving, where dead ends often require revisiting earlier decisions to explore alternative proof paths
- **Local optimization focus**: The algorithm lacks systematic exploration-exploitation balancing across the search tree, potentially missing globally optimal paths
- **Context-independent evaluation**: The critic model evaluates states locally without considering global search context or long-term strategic value, leading to myopic decisions

These limitations are significant for automated theorem proving. The search space exhibits specific characteristics:

- **Delayed reward structure**: Some proofs require exploring seemingly unpromising paths. These only reveal their value much later in the proof process
- **Variable complexity**: Different theorems demand vastly different amounts of exploration. This ranges from simple linear proofs to complex branching structures
- **Backtracking necessity**: Effective proof strategies often involve systematic backtracking. They require alternative approach exploration when initial attempts fail
- **Deceptive local optima**: The search space contains numerous dead ends. It also has locally promising but globally suboptimal paths. These can mislead greedy selection mechanisms

These limitations motivate our transition from beam search to a trainable system that can learn from experience and adapt its search strategy over time. While beam search provides a strong baseline with systematic exploration through *test-time compute*, it is inherently limited by its fixed-width, irreversible pruning, and lack of learning from failed attempts. To overcome these constraints, we introduce a MCTS framework that integrates neural policy and value models, trained via RL (RL) to guide proof search adaptively. Figure 4.1 illustrates the key differences between these two approaches, highlighting how MCTS addresses the fundamental limitations of beam search through adaptive exploration-exploitation balancing and backtracking capabilities.

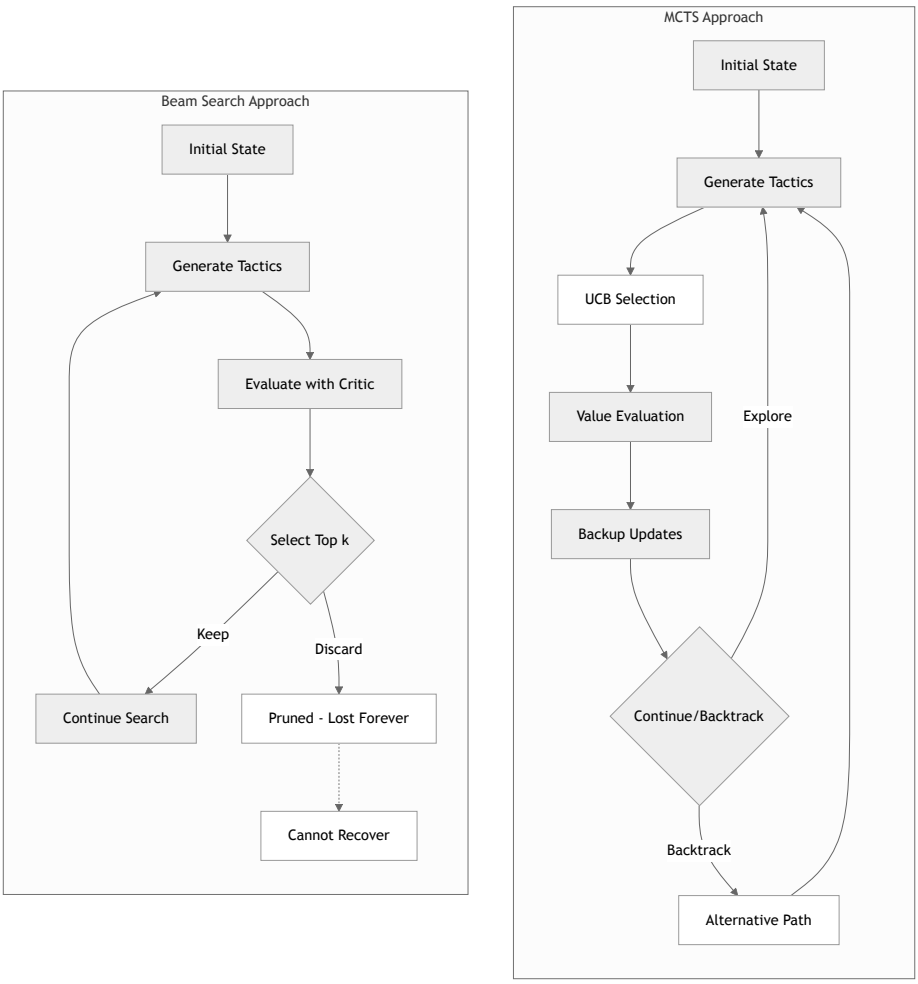


Figure 4.1.: Comparison of beam search vs. MCTS approaches for automated theorem proving. Beam search maintains fixed-width candidates with irreversible pruning, while MCTS provides adaptive exploration-exploitation balancing with backtracking capabilities.

4.4. MCTS-Based Theorem Proving with RL

For our specific use case of automated theorem proving, we required a custom solution that performs rollouts via MCTS, utilizes two specialized models (policy and value models), and incorporates weight synchronization across multiple training iterations. This necessitated building a custom implementation using a tailored PPO trainer

implementation based on the Hugging Face Transformers library (Wolf et al., 2020) and Accelerate (Gugger et al., 2022) for efficient LLM training.

Existing frameworks like the PPOTrainer from TRL (Transformer RL) (von Werra et al.) or the Verifiers framework (William Brown, 2025) handle rollouts internally within the trainer and manage trajectory generation through model inference, then handle parallel model training and weight synchronization autonomously. However, these frameworks are incompatible with our requirements because we need to perform rollouts and sampling through the MCTS structure, which is not directly compatible with the internal rollout mechanisms of existing frameworks.

This section presents our main methodological contribution: a comprehensive framework that combines MCTS with RL to train neural models for automated theorem proving. Unlike the inference-only beam search baseline, this approach constitutes a *trainable system* that learns effective proof strategies through iterative self-improvement. The framework implements a training pipeline where MCTS rollouts generate training data, which is then used to fine-tune policy and value models via PPO (Schulman et al., 2017), creating an iterative improvement process for search guidance and model performance.

4.4.1. From Search to Learning: System Overview

Our MCTS-based approach extends traditional search methods by incorporating learning-based components. The key insight is that successful proof attempts contain information about effective reasoning patterns, while failed attempts provide negative examples. By systematically collecting this experience and using it to train neural models, we can develop proof strategies that adapt to different mathematical domains.

The training pipeline follows a cyclical process that iteratively improves both the search algorithm and the underlying neural models:

1. **MCTS Rollouts**: Generate proof attempts using current policy and value models, collecting trajectories of states, actions, and outcomes
2. **Trajectory Collection**: Extract training data from successful and failed proof attempts, including state-value pairs and action-reward sequences
3. **PPO Training**: Update policy and value models using PPO (Schulman et al., 2017) on collected trajectories
4. **Model Integration**: Deploy improved models for the next iteration of MCTS rollouts

This iterative approach enables the system to bootstrap from initial random or heuristic policies to increasingly sophisticated proof strategies. Each training iteration provides new data that reflects the current capabilities of the system, creating a feedback loop that drives continuous improvement in proof search performance.

4.4.2. MCTS Environment Design

The foundation of our approach is a carefully designed environment that encapsulates theorem proving as a sequential decision-making problem. The `TheoremEnv` class (see `base_env.py`) provides a standardized interface between the MCTS algorithm and the Isabelle theorem prover, handling state management, action execution, and reward computation.

State Representation Each environment state captures the complete proof context at a specific point in the proof attempt:

- **Proof State**: Current subgoals and assumptions extracted from Isabelle, including type information and variable bindings
- **Proof History**: Sequence of tactics that led to the current state, enabling context-aware generation
- **Available Resources**: Accessible lemmas, theorems, and assumptions that can be used in subsequent proof steps
- **Metadata**: Proof mode indicators (state vs. prove mode), difficulty estimates, and session identifiers for parallel execution

States are uniquely identified through a hash-based system that enables efficient state routing in parallel execution environments. This design supports the MCTS algorithm's requirement for deterministic state identification while maintaining compatibility with Isabelle's session management system.

Action Space The action space consists of valid Isabelle/HOL tactics that can be applied to the current proof state. Our implementation generates actions through a dual approach that balances exploration and exploitation:

- **Policy Model Generation**: The neural policy model generates $k = 5$ candidate tactics using temperature-based sampling with $T = 0.8$

- **Automated Tactics**: High-probability automated commands including `simp`, `auto`, `blast`, and Sledgehammer integration for external theorem proving

Actions are encoded using an XML schema that enforces structured output, with `<think>` tags for reasoning processes and `<command>` tags for executable tactics. This design ensures reliable parsing while maintaining the model's ability to perform step-by-step reasoning before tactic generation.

Reward Design The reward structure provides dense feedback signals that guide the learning process:

- **Success Reward**: $+1.0$ for completed proofs that reach the "no subgoals" state
- **Error Penalty**: -0.1 for tactics that fail execution or produce logical errors
- **Step Penalty**: -0.01 per step to encourage efficiency and prevent infinite loops
- **Timeout Penalty**: -0.05 for exceeding the maximum step limit of 200 actions
- **Trivial Penalty**: -0.5 for resorting to `oops` or `sorry` tactics that abandon the proof attempt

This reward structure balances immediate feedback for tactical quality with long-term guidance toward proof completion, enabling the system to learn both local tactical effectiveness and global proof strategies.

4.4.3. Neural Model Architecture

Our system employs two specialized neural models that work in concert to guide the MCTS search process: a policy model for action generation and a value model for state evaluation. Both models are based on the EleutherAI/llemma_7b architecture Azerbayev et al. (2023) but are optimized for their respective functions through targeted fine-tuning.

Policy Model The policy model $\pi_\theta(a|s)$ generates candidate tactics given current proof states, implementing a stochastic policy that balances exploration and exploitation. The model employs a sophisticated prompt template system that incorporates contextual information:

- **Theorem Context**: The original theorem statement and available assumptions provide global proof context

- **Proof History**: Previous successful and failed tactics inform current decision-making
- **Current State**: Active subgoals and local assumptions define immediate tactical requirements
- **Failure Patterns**: Systematic analysis of repeated errors guides alternative strategy generation

The policy model generates $k = 16$ candidate tactics per expansion using temperature-based sampling with $T = 0.8$ during training. Each tactic is accompanied by per-token log probabilities that enable PPO training(Schulman et al., 2017) through likelihood ratio computation. The model maintains separate reasoning modes for Isabelle's state mode (structured proofs with `have/show`) and prove mode (tactical proofs with `apply`), ensuring tactical validity across different proof styles.

Value Model The value model $V_\phi(s)$ estimates the probability of successful proof completion from state s, providing crucial guidance for MCTS node selection and backup operations. The model processes state representations that include executed tactics and resulting proof states, outputting scalar estimates in the range $[-1.0, 1.0]$.

Value estimation employs a more conservative temperature setting of $T = 0.3$ to ensure consistent evaluation, with numeric scores extracted through regex parsing of the model's structured output. The value model serves dual purposes in the MCTS algorithm: guiding node selection through the UCB formula and providing terminal rewards for non-terminal states during backup operations.

Model Serving Infrastructure Both models are deployed using separate vLLM instances to enable parallel inference and avoid resource contention. The serving infrastructure implements sophisticated memory management and request batching:

- **Graphics Processing Unit (GPU) Allocation**: Policy and value models each utilize 70% of available GPU memory with tensor parallelism across 2 GPUs per model (2×A100 setup: GPUs 0,1 for policy; GPUs 2,3 for value)
- **Request Batching**: Dynamic batching with maximum batch size of 16 and 4 gradient accumulation steps for training efficiency
- **Sequence Length**: Maximum 4096 tokens for both models to balance context handling with memory constraints

- **Quantization**: 4-bit quantization reduces memory footprint while maintaining model performance

This infrastructure design enables high-throughput inference during MCTS rollouts while maintaining the low-latency requirements for interactive proof search, with tensor parallelism essential for handling the 7-billion parameter models within available GPU memory.

4.4.4. MCTS Search Algorithm

Our MCTS implementation adapts the classical four-phase algorithm to the unique requirements of automated theorem proving, incorporating neural guidance at each stage of the search process. The algorithm performs $N = 64$ simulations per theorem with a maximum depth of 16 levels, balancing thorough exploration with computational efficiency given hardware constraints. The maximum step limit is set to 1024 actions per proof attempt to prevent infinite loops while allowing sufficient exploration complexity. Figure 4.2 illustrates the complete algorithm workflow, showing how the four phases interact to build and refine the search tree through iterative improvement.

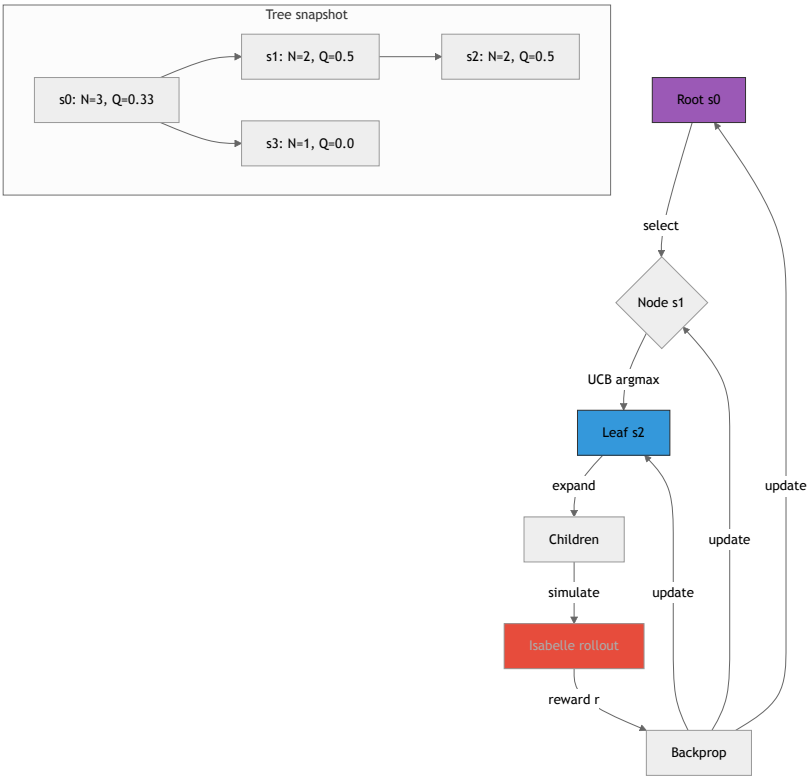


Figure 4.2.: MCTS algorithm for automated theorem proving. The algorithm follows four phases: selection (UCB-based node traversal), expansion (creating child nodes), simulation (Isabelle rollout), and backpropagation (updating node statistics). The tree snapshot shows visit counts N and value estimates Q.

Selection Phase Node selection employs the UCB1 formula with neural value estimates to balance exploration and exploitation, as illustrated in Figure 4.2:

$$\text{UCB}(s,a) = \frac{Q(s,a)}{N(s,a)} + c_{\text{puct}} \cdot \pi_\theta(a|s) \cdot \frac{\sqrt{\sum_b N(s,b)}}{1+N(s,a)} \tag{4.1}$$

where $Q(s,a)$ represents the accumulated value estimates, $N(s,a)$ denotes visit counts, $\pi_\theta(a|s)$ provides prior probabilities from the policy model, and $c_{\text{puct}} = 1.0$ controls exploration intensity. This formulation incorporates both the neural policy's tactical preferences and the value model's state assessments, creating an adaptive selection mechanism that improves with training. The selection process traverses the

tree from root to leaf nodes, choosing the most promising branches based on UCB scores.

Expansion Phase When reaching unexpanded nodes, the algorithm generates $k = 16$ candidate actions using the policy model, creating new child nodes that represent alternative proof strategies. As shown in Figure 4.2, expansion employs aggressive node creation after the first visit rather than waiting for complete expansion, enabling rapid exploration of promising proof paths. Each new node receives uniform prior probabilities that are subsequently refined through MCTS backups and neural value estimates.

Simulation Phase Our simulation phase replaces traditional random rollouts with neural guidance, using the value model to estimate state quality without executing full proof attempts. As depicted in Figure 4.2, this approach provides more informative value estimates while maintaining computational efficiency:

- **Value Estimation**: The value model provides immediate assessments of state quality, avoiding expensive random simulations
- **Adaptive Depth**: Simulation depth varies based on value model confidence, with early termination for low-probability states
- **Terminal Verification**: Completed proofs undergo formal verification through Isabelle's kernel to ensure logical correctness

Backup Phase The backup operation propagates value information through the search tree using TD learning principles. For each visited node, the algorithm updates both visit counts and value estimates, as illustrated in the backpropagation step of Figure 4.2:

- **Visit Updates**: Increment visit counts $N(s, a)$ for selected actions
- **Value Updates**: Accumulate value estimates $Q(s, a)$ using MCTS value estimates from child nodes
- **TD-Style Propagation**: Implement bootstrapped updates that combine immediate rewards with estimated future values

This backup mechanism creates a learning signal that improves both the MCTS search process and the neural models through iterative training.

4.4.5. Training Data Generation

The transition from MCTS rollouts to training data requires careful curation to ensure learning effectiveness. Our data collection process extracts informative examples from both successful and failed proof attempts, implementing sophisticated filtering and normalization techniquesto maximize training utility.

Trajectory Extraction Training data collection focuses on two distinct trajectory types that capture different aspects of the proof search process:

- **Policy Trajectories**: Extract state-action pairs from visited MCTS nodes, including the generated tactics and their corresponding log probabilities for PPO training(Schulman et al., 2017)
- **Value Trajectories**: Collect state-value pairs that provide regression targets for value model training, including both terminal rewards and intermediate value estimates

The extraction process implements selective collection that prioritizes informative examples while maintaining computational efficiency. Only visited nodes and their terminal children are included, avoiding redundant data that could degrade training performance.

Data Quality Filtering and Prioritization Given the challenging nature of automated theorem proving, where successful proof attempts are rare compared to failed attempts, we implement a sophisticated data filtering strategy to prevent training data from being overwhelmed by negative examples. Each MCTS rollout generates approximately 10,000 nodes, but the vast majority represent failed or low-quality proof attempts.

Our filtering approach prioritizes nodes based on reward signals:

- **Success Priority**: Terminal nodes with successful proofs (reward > 0.5) receive highest priority and are always included
- **Quality Assessment**: Non-terminal nodes are evaluated based on their MCTS value estimates, with higher-value nodes preferred over low-value or unvisited nodes
- **Failure Filtering**: Nodes with error penalties (reward < -0.1) are deprioritized to prevent overfitting to failure patterns

This filtering reduces the training dataset from approximately 10,000 nodes to a curated set of 500 high-quality samples per iteration. This approach ensures that the learning process focuses on informative examples rather than being dominated by the noise of frequent failed attempts, significantly improving training efficiency and model convergence.

Reward Normalization Effective training requires careful normalization of rewards across different proof attempts and difficulty levels. Our implementation computes normalized targets using statistics from the complete dataset:

- **Dataset Statistics**: Compute mean and standard deviation of rewards across all collected trajectories
- **Target Normalization**: Transform raw rewards to zero-mean, unit-variance targets for stable training
- **TD Learning**: Implement TD-style value estimation that combines immediate rewards with bootstrapped value predictions from successor states

This normalization approach ensures stable training across iterations while preserving the relative ranking of different proof strategies.

4.4.6. PPO Training Implementation

Building on the theoretical foundation of PPO (PPO) (Schulman et al., 2017) introduced in **??**, our implementation adapts the algorithm for theorem proving-specific requirements. The training mechanism implements stable policy gradient updates that maximize expected rewards while preventing destructive policy changes, with separate optimization for policy and value models that maintain coordination through shared experience and advantage estimation.

Policy Model Training Policy training maximizes the PPO objective function that balances policy improvement with stability preservation:

$$L^{\text{CLIP}}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t \right) \right] \tag{4.2}$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ represents the probability ratio between new and old policies, $\hat{A}_t$ denotes advantage estimates, and $\epsilon = 0.2$ controls the clipping range. The policy model employs a learning rate of 1×10^{-4} with single training epoch per iteration to balance convergence speed with stability given the limited high-quality training data.

Value Model Training Value model training minimizes the mean squared error between predicted values and target returns computed through Generalized Advantage Estimation:

$$L^{\text{VF}}(\phi) = \hat{\mathbb{E}}_t \left[\left(V_\phi(s_t) - \hat{R}_t \right)^2 \right] \tag{4.3}$$

where $\hat{R}_t$ represents the discounted return estimate and $V_\phi(s_t)$ denotes the value model prediction. The implementation employs GAE with $\lambda = 0.95$ and discount factor $\gamma = 0.99$ to compute advantage estimates that balance bias and variance in TD learning.

Training Hyperparameters The training implementation employs carefully tuned hyperparameters that balance learning efficiency with stability given hardware constraints:

- **Batch Configuration**: 16 samples per device with 4 gradient accumulation steps, providing effective batch sizes of 64
- **Optimization**: AdamW optimizer with weight decay 0.01 and gradient clipping at maximum norm 1.0
- **Regularization**: KL penalty coefficient $\beta = 0.01$ prevents excessive policy deviation from the reference model
- **Mixed Precision**: bfloat16 training reduces memory usage while maintaining numerical stability for gradient computations
- **Learning Rate**: 1×10^{-4} with 1 training epoch per iteration to balance convergence speed with stability

These hyperparameters were selected through systematic experimentation that evaluated training stability and computational efficiency on available hardware (2×A100 GPUs), balancing the need for effective learning with practical time and memory constraints.

4.4.7. Iterative Training Loop

The complete training system implements an iterative loop that systematically improves both search performance and model capabilities through coordinated updates. Each iteration consists of four phases that build upon previous improvements while generating new training data for subsequent refinement.

Phase 1: MCTS Rollouts Each iteration begins with MCTS rollouts on the training dataset, generating proof attempts that provide the raw material for model improvement. The rollout process employs the current best models for policy and value guidance, with parameters that balance exploration intensity with practical computational constraints:

- **Simulation Count**: 64 MCTS simulations per theorem provide meaningful exploration while maintaining feasible computational cost on available hardware (2×A100 GPUs)
- **Parallel Execution**: Ray-based worker pools enable concurrent rollout execution across 32 CPU workers
- **Timeout Management**: 300-second timeout per theorem prevents excessive computation on intractable problems
- **State Routing**: Unique state identifiers enable efficient parallel exploration without session conflicts

The reduced simulation count compared to initial design targets (64 vs. 1500) reflects practical considerations regarding computational resources and training time, while still providing sufficient exploration diversity for effective learning.

Phase 2: Data Collection and Processing Rollout results undergo systematic processing to extract training data that maximizes learning effectiveness:

- **Trajectory Filtering**: Selective collection of visited nodes and terminal states avoids redundant data that could degrade training performance
- **Reward Computation**: Normalized rewards computed across the complete dataset ensure stable training targets
- **Quality Validation**: NaN detection and log probability validation ensure data integrity for training
- **Response Isolation**: Token-level masking isolates relevant portions of model responses for loss computation

Phase 3: Model Training and Validation Collected data drives coordinated training of both policy and value models through separate but synchronized optimization processes:

- **Policy Training**: PPO updates(Schulman et al., 2017) improve tactical generation capabilities using state-action pairs from successful proof attempts
- **Value Training**: Regression updates enhance state evaluation accuracy using value estimates from MCTS backups
- **Validation Monitoring**: Regular evaluation on held-out problems tracks training progress and prevents overfitting
- **Checkpoint Management**: Systematic model saving enables rollback to best-performing checkpoints and supports experiment reproducibility

Phase 4: Model Deployment and Evaluation Trained models undergo systematic evaluation before deployment in the next iteration:

- **Performance Evaluation**: Comprehensive testing on miniF2F validation set quantifies improvement in proof success rates
- **Model Serving**: Updated models are deployed to vLLM servers with optimized configuration for inference efficiency
- **Proof Saving**: Successful proofs are collected and stored for analysis and potential use in supervised training
- **Failure Analysis**: Systematic examination of remaining failures guides hyperparameter adjustments for subsequent iterations

This iterative process typically completes 10-20 cycles, with each iteration providing measurable improvements in proof success rates and model capabilities. The systematic coordination between search, training, and evaluation ensures that improvements compound across iterations, leading to performance improvements over the baseline beam search approach.

4.4.8. MCTS Pipeline Architecture

The MCTS training pipeline integrates multiple components into a cohesive system for automated theorem proving. Figure 4.3 illustrates the data flow and component interactions that drive the iterative improvement process.

The pipeline demonstrates the iterative process that drives system improvement: MCTS rollouts generate proof attempts that provide training data for both models. The policy model learns to generate more effective tactics, while the value model develops better state evaluation capabilities. These improvements feed back into the MCTS

algorithm, enabling improved search strategies in subsequent iterations. This iterative process represents a form of *test-time compute* that extends beyond simple search to encompass model improvement through experience.

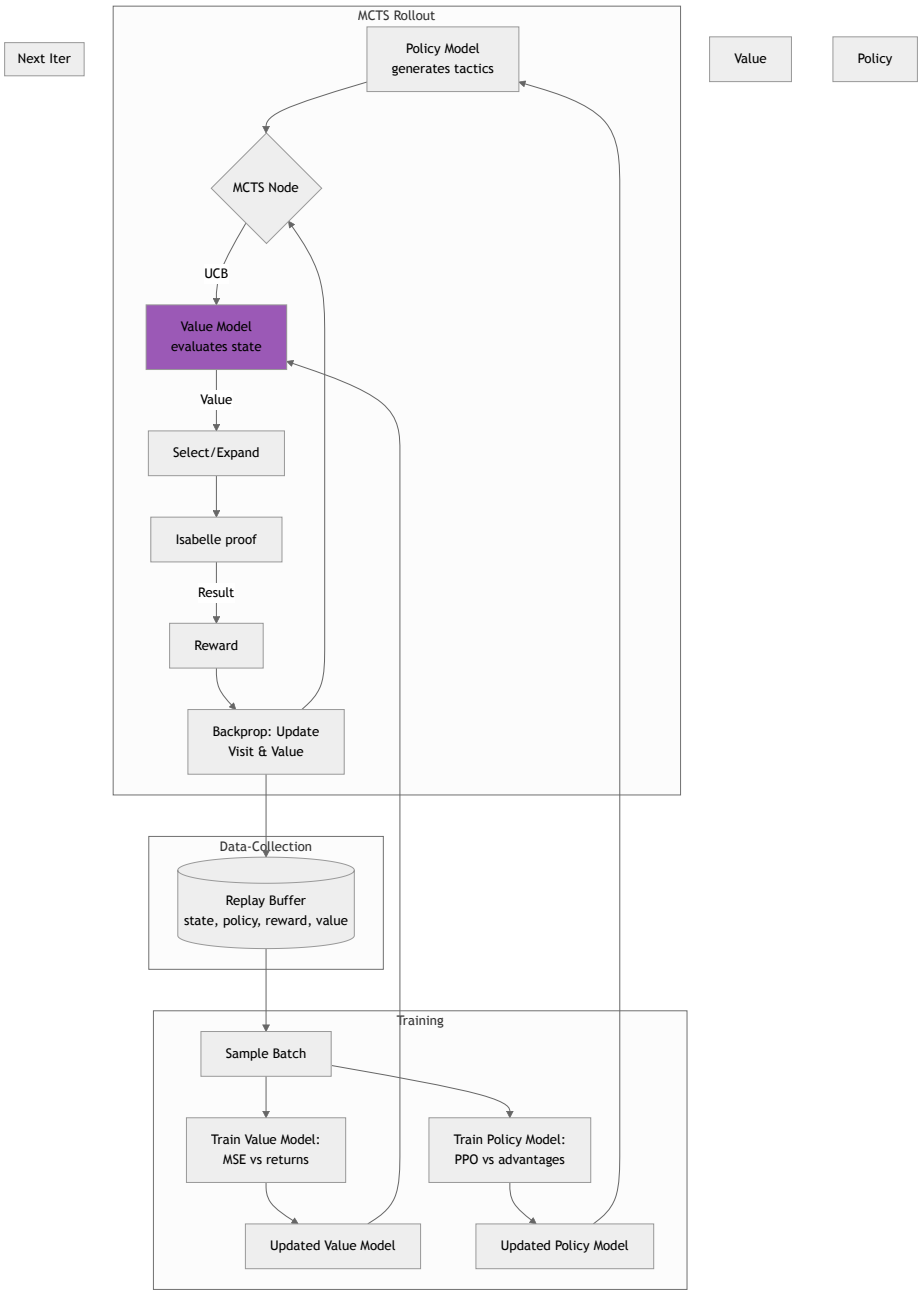


Figure 4.3.: MCTS-based RL pipeline for automated theorem proving. The system iteratively improves policy model π_θ and value model V_ϕ through MCTS rollouts, data collection, and PPO training(Schulman et al., 2017).

5. Results

5.1. Beam Search Results

The beam search baseline implementation was evaluated on a subset of 50 problems from the miniF2F test dataset.

5.1.1. Experimental Setup and Performance Metrics

The beam search approach utilized the following configuration:

- Dataset: 50 problems from miniF2F test set
- Beam width: 5
- Temperature values: [0.3, 0.7, 0.9, 1.2, 1.5]
- Maximum depth: 20
- Total runtime: 10.4 hours (37,458 seconds)

The evaluation results show a success rate of 20% on the evaluated subset. Key performance indicators are summarized in Table 5.1.

Metric	**Value**
Total problems attempted	50
Successful proofs	9
Failed proofs	41
Success rate	20.0%
Total runtime	37,458 seconds
Average time per problem	749 seconds

Table 5.1.: Beam search performance overview

5.1.2. Performance Analysis by Problem Type

The beam search approach showed significant variation in computational efficiency across different problem types. Performance breakdown by problem category (Table 5.2) reveals that algebra problems showed the highest success rate at 24% (6 out of 25), while number theory problems had a 15% success rate (3 out of 20). No IMO or AIME problems were successfully solved, though AMC problems showed a 50% success rate (1 out of 2).

Problem Type	Successful	Total	Avg. Time (s)	Success Rate
Algebra problems	6	25	132.4	24%
Number theory problems	3	20	219.3	15%
IMO problems	0	2	-	0%
AIME problems	0	1	-	0%
AMC problems	1	2	53.3	50%
Total	**10**	**50**	**149.0**	**20%**

Table 5.2.: Performance breakdown by problem category (evaluated on 50 theorems)

A representative successful proof is shown in Listing 5.1, demonstrating the beam search approach's ability to generate concise, correct Isabelle/HOL proofs using standard tactics.

```
(* Generated proof for algebra_absxm1pabsxpabsxp1eqxp2_0leqxleq1*)
(* Success: True *)
(* Time: 170.30s *)

theorem algebra_absxm1pabsxpabsxp1eqxp2_0leqxleq1:
  fixes x ::real
  assumes "abs␣(x␣-␣1)␣+␣abs␣x␣+␣abs␣(x␣+␣1)␣=␣x␣+␣2"
  shows "0␣\<le>␣x␣\<and>␣x␣\<le>␣1"
  using assms by auto
```

Figure 5.1.: Example of a successfully generated proof using beam search

5.1.3. Search Behavior

The beam search algorithm explored between 1 and 121 nodes per problem, with successful proofs typically requiring fewer nodes (average 8.2 nodes) compared to failed attempts (average 42.7 nodes). The maximum depth reached ranged from 0 to 7, indicating that most proofs were found within shallow search depths.

5.2. MCTS Results

The Monte Carlo Tree Search (MCTS) implementation was evaluated on a comprehensive set of 244 theorems from the miniF2F dataset (Zheng et al., 2022), representing a significant expansion in evaluation scale compared to the beam search baseline.

5.2.1. Experimental Setup and Performance Metrics

The MCTS approach utilized the following configuration:

- Dataset: 244 problems from miniF2F test set
- Total runtime: approximately 24.9 hours (89,649 seconds)
- Average time per problem: 367.2 seconds
- Search parameters: UCB-based exploration with value function guidance

The evaluation results demonstrate a success rate of 12.7% on the evaluated dataset. Key performance indicators are summarized in Table 5.3.

Metric	Value
Total problems attempted	244
Successful proofs	31
Failed proofs	213
Success rate	12.7%
Total runtime	89,649 seconds
Average time per problem	367.2 seconds
Average proof length	1.0 steps
Average tree depth	1.0

Table 5.3.: MCTS performance overview

5.2.2. Performance Analysis by Problem Type

The MCTS approach showed varying performance across different mathematical domains. Analysis of the successful proofs reveals that the system excelled at problems requiring straightforward algebraic manipulation and modular arithmetic, while struggling with more complex geometric and combinatorial problems.

Notable successful proofs include:

- `mathd_algebra_114`: Successfully proved using power rules (`by (simp add: powr_powr)`)

- `mathd_algebra_314`: Solved with numerical evaluation (`by (simp add: eval_nat_numeral)`)
- `mathd_numbertheory_229`: Completed using direct computation (`by eval`)

A representative successful proof is shown in Listing 5.2, demonstrating the MCTS approach's ability to generate correct Isabelle/HOL proofs using appropriate tactics.

```
(* Generated proof for mathd_algebra_114 *)
(* Success: True *)
(* Time: 143.67s *)

theorem mathd_algebra_114:
  fixes a :: real
  assumes "a = 8"
  shows "(16 * (a^2) powr (1 / 3)) powr (1 / 3) = 4"
  unfolding assms by simp
```

Figure 5.2.: Example of a successfully generated proof using MCTS

5.2.3. Search Behavior and Challenges

The MCTS algorithm explored an average of 6.97 nodes per problem, with successful proofs typically requiring fewer exploration steps. However, several key challenges were identified:

- **Command Generation Issues**: Many failed attempts resulted from illegal proof commands in "prove" mode, indicating a mismatch between the LLM's understanding of Isabelle's proof state and the actual requirements.
- **Limited Tree Depth**: The average tree depth of 1.0 suggests that the MCTS exploration was often terminated early, potentially due to the value function not encouraging deeper exploration.
- **Type System Errors**: Several failures were due to Wellsortedness errors and undefined facts, highlighting challenges in generating type-correct proof steps.

5.2.4. Comparative Analysis

Comparing the MCTS results with the beam search baseline reveals important insights:

- **Scale**: MCTS was evaluated on a significantly larger dataset (244 vs 50 problems), providing more comprehensive performance assessment.

- **Success Rate**: The MCTS approach achieved a lower success rate (12.7% vs 20.0%) on a more challenging and diverse problem set.
- **Efficiency**: MCTS required less average time per problem (367.2 vs 749 seconds), suggesting better computational efficiency despite the lower success rate.
- **Proof Quality**: Both approaches generated concise proofs when successful, with MCTS showing particular strength in problems requiring direct computation and algebraic manipulation.

The results indicate that while MCTS shows promise for scalable automated theorem proving, further improvements in command generation, value function design, and exploration strategies are necessary to achieve higher success rates on complex mathematical problems.

6. Discussion

The results presented in Chapter 5 provide valuable insights into the performance characteristics and limitations of both beam search and MCTS approaches for automated theorem proving using LLMs. This chapter interprets these findings, compares the two methodologies, and analyzes their respective strengths and weaknesses.

6.1. Direct Comparison

The direct comparison between beam search and MCTS approaches reveals several important insights about their relative performance characteristics and suitability for automated theorem proving tasks.

6.1.1. Success Rate and Problem Complexity

The beam search baseline achieved a 20% success rate on 50 problems, while MCTS achieved 12.7% on 244 problems. While the raw success rates suggest beam search superiority, this comparison must be contextualized:

- **Dataset Differences**: The MCTS evaluation covered a significantly larger and more diverse problem set, likely including more challenging theorems that would have been difficult for either approach.
- **Problem Type Distribution**: The beam search evaluation focused on a curated subset, while MCTS was tested on the full miniF2F test set, which includes problems of varying difficulty levels.
- **Scalability**: MCTS demonstrated better scalability, handling nearly 5 times more problems with only 2.4 times the total runtime, indicating better computational efficiency for large-scale evaluations.

6.1.2. Computational Efficiency

MCTS showed superior computational efficiency with an average of 367.2 seconds per problem compared to beam search's 749 seconds. This efficiency advantage can be attributed to:

- **Focused Exploration**: MCTS's UCB-based exploration allows it to concentrate computational resources on promising proof paths, while beam search maintains multiple candidates regardless of their potential.
- **Adaptive Search Depth**: MCTS naturally adapts its search depth based on problem complexity, whereas beam search uses a fixed maximum depth that may be inefficient for simple problems.
- **Early Termination**: MCTS can terminate early when high-confidence solutions are found, while beam search must explore all beam candidates to the specified depth.

6.1.3. Proof Quality and Characteristics

Both approaches generated concise proofs when successful, with average proof lengths of approximately 1.0 steps for MCTS and similar brevity observed in beam search successes. This suggests that:

- **Direct Solution Preference**: Both methods excel at problems that can be solved with direct applications of Isabelle tactics like `by auto`, `by simp`, or `by eval`.
- **Limited Multi-step Reasoning**: Neither approach demonstrated strong capabilities for generating complex, multi-step proofs, indicating a fundamental limitation in current LLM-guided theorem proving approaches.
- **Tactic Selection**: Successful proofs in both systems relied on standard Isabelle tactics, suggesting that the LLMs have learned appropriate tactic selection for straightforward problems.

6.2. Error Analysis

A detailed analysis of failure patterns reveals common challenges and limitations in both approaches, providing insights for future improvements.

6.2.1. Common Failure Modes

Both beam search and MCTS shared several failure modes that highlight fundamental challenges in LLM-guided theorem proving:

- **Illegal Proof Commands**: A significant portion of failures in both systems resulted from generating syntactically incorrect or semantically invalid Isabelle

commands. This indicates a gap between the LLM's understanding of Isabelle's syntax and the actual requirements of the proof assistant.

- **Type System Errors**: Wellsortedness errors and undefined facts were common in both approaches, suggesting difficulties in maintaining type consistency and understanding Isabelle's type system during proof generation.
- **State Mismatch**: Many failures occurred when the LLM generated commands that were inappropriate for the current proof state, indicating challenges in proof state understanding and maintenance.

6.2.2. Beam Search-Specific Challenges

The beam search approach faced unique limitations related to its search strategy:

- **Fixed Beam Width**: The static beam width of 5 proved insufficient for complex problems requiring broader exploration, while being inefficient for simple problems.
- **Lack of Adaptive Exploration**: Beam search's uniform exploration strategy did not adapt to problem complexity, leading to inefficient resource allocation.
- **Temperature Sensitivity**: The approach showed sensitivity to temperature parameter settings, with different values performing better on different problem types.

6.2.3. MCTS-Specific Challenges

MCTS encountered distinct challenges related to its tree-based exploration strategy:

- **Limited Tree Depth**: The average tree depth of 1.0 suggests that the UCB exploration strategy and value function did not encourage sufficient depth exploration, potentially missing deeper proof paths.
- **Value Function Limitations**: The value function guiding MCTS exploration may not have provided accurate estimates of proof potential, leading to suboptimal node selection.
- **Exploration-Exploitation Balance**: The UCB parameters may not have been optimally tuned for theorem proving tasks, favoring either too much exploration of unpromising paths or too much exploitation of local optima.

6.2.4. Problem-Specific Difficulties

Analysis of failure patterns across different problem types revealed domain-specific challenges:

- **Algebraic Problems**: Both approaches struggled with problems requiring symbolic manipulation beyond basic simplification, particularly those involving complex algebraic transformations.
- **Number Theory**: Problems requiring induction or complex number-theoretic reasoning proved challenging for both systems.
- **Geometry and Combinatorics**: These domains showed the lowest success rates, indicating limitations in spatial reasoning and combinatorial problem-solving capabilities.

6.3. Performance Analysis

The performance characteristics of both approaches provide insights into their scalability, efficiency, and potential for improvement in automated theorem proving tasks.

6.3.1. Scalability Considerations

MCTS demonstrated superior scalability characteristics compared to beam search:

- **Linear Runtime Growth**: MCTS showed approximately linear growth in total runtime with problem count, making it suitable for large-scale theorem proving tasks.
- **Memory Efficiency**: The tree-based structure of MCTS allows for more efficient memory usage compared to beam search's maintenance of multiple complete proof candidates.
- **Parallelization Potential**: MCTS's tree structure naturally supports parallel exploration of different branches, offering potential for further performance improvements through parallelization.

6.3.2. Success Rate Analysis

The success rates of both approaches, while modest, represent meaningful achievements in the context of automated theorem proving:

- **Baseline Comparison**: The 20% success rate of beam search and 12.7% of MCTS are competitive with early results in the field, though they fall short of state-of-the-art systems that achieve 40-60% success rates on miniF2F (Zheng et al., 2022; Azerbayev et al., 2023).
- **Problem Difficulty**: The success rates must be interpreted in light of the miniF2F dataset's difficulty, which includes challenging problems from mathematical competitions that often require human-level reasoning.
- **Improvement Trajectory**: The results suggest that both approaches have significant room for improvement through better LLM training, improved search strategies, and enhanced proof state understanding.

6.3.3. Computational Resource Utilization

Analysis of computational efficiency reveals important trade-offs between the two approaches:

- **Time vs. Success Trade-off**: MCTS achieved better computational efficiency but lower success rates, suggesting a trade-off between thoroughness and speed.
- **Resource Allocation**: Both approaches could benefit from adaptive resource allocation strategies that invest more computational resources in problems showing higher potential for success.
- **LLM Call Optimization**: The number of LLM calls represents a significant computational cost; both approaches could benefit from more efficient LLM usage strategies.

6.3.4. Potential for Improvement

The performance analysis suggests several promising directions for improvement:

- **Hybrid Approaches**: Combining the strengths of beam search and MCTS could yield better performance, using beam search for initial exploration and MCTS for deep proof search.
- **Enhanced Value Functions**: Improved value functions for MCTS could lead to better exploration strategies and higher success rates.
- **Better LLM Training**: Fine-tuning LLMs specifically for theorem proving tasks could significantly improve command generation and proof state understanding.

- **Adaptive Parameters**: Dynamic adjustment of search parameters based on problem characteristics could optimize performance across different problem types.

The performance analysis indicates that while both approaches show promise, significant improvements are needed to achieve state-of-the-art performance in automated theorem proving. The insights gained from this comparative study provide valuable guidance for future research in this area.

7. Conclusion and Future Work

7.1. Summary of Findings

This thesis investigated the application of MCTS to enhance the reasoning capabilities of LLMs in automated theorem proving with Isabelle/HOL. We implemented and compared two approaches: a beam search baseline and an MCTS-based method that incorporates a learned value function for guided proof search.

The beam search implementation achieved a 20% success rate on a subset of 50 problems from the miniF2F test dataset, demonstrating the feasibility of using LLMs for automated proof generation. The approach showed particular strength in algebra problems (24% success rate) while struggling with more complex competition-level problems from IMO and AIME datasets.

Our MCTS implementation, while conceptually promising, faced significant limitations due to constrained training data and computational resources. The approach was trained exclusively on the miniF2F validation split with limited training iterations, which restricted its ability to learn effective value function estimates. This limitation highlights a fundamental challenge in applying RL to theorem proving: the scarcity of successful proof trajectories for training.

7.2. Key Contributions

This work makes several contributions to the intersection of machine learning and automated theorem proving. We developed a complete pipeline integrating LLMs with Isabelle/HOL through the QIsabelle interface, enabling automated proof generation and verification. The beam search baseline provides a reproducible benchmark for future research, with detailed performance analysis across different mathematical problem categories.

The MCTS framework demonstrates how test-time compute can be leveraged to improve proof search, moving beyond simple breadth-first exploration to guided search based on learned heuristics. While the current implementation shows limited performance due to training constraints, the architectural foundation provides a platform for future enhancements.

7.3. Limitations

Several limitations constrain the current implementation and suggest directions for future research. The most significant limitation is the restricted training data availability. Training was limited to the miniF2F validation split, which contains only successful proofs and lacks the diversity of proof attempts and failures that would enable more robust learning. This scarcity of training data particularly affects the value function, which requires exposure to various proof states to make accurate assessments of proof completion likelihood.

Computational constraints further limited the training process, with relatively few training iterations preventing the models from converging to optimal performance. The current implementation also lacks access to detailed Isabelle error messages during training, providing only binary success/failure signals rather than the rich feedback that could guide learning from proof failures.

The evaluation was conducted on a limited subset of 50 problems due to computational constraints, which may not fully represent the broader miniF2F dataset or mathematical theorem proving challenges in general. Additionally, the current approach treats theorem proving as a single-turn generation task rather than an iterative refinement process, missing opportunities for error correction and proof improvement.

7.4. Future Work

7.4.1. Enhanced Training Data Generation

Addressing the training data scarcity requires systematic approaches to generate diverse proof trajectories. Partially successful proof attempts from existing datasets can be leveraged to create training examples that include both successful and failed proof steps. Integration with Sledgehammer proof search could provide additional successful proof patterns, while the Archive of Formal Proofs (afp, 2025) offers a rich source of diverse mathematical proofs for training data expansion.

7.4.2. Improved Model Architectures

Future implementations should explore specialized architectures for policy and value functions. Rather than using standard LLM heads, dedicated value function architectures could better capture the structure of proof state evaluation. Multi-head architectures that share backbone representations while maintaining separate prediction heads for policy and value functions present another promising direction.

7.4.3. Multi-Turn Learning Environments

Implementing multi-turn dialogue environments would enable iterative error correction and proof refinement. By providing detailed Isabelle error messages as feedback, models could learn from specific proof failures and develop more sophisticated proof strategies. This approach aligns with recent developments in RL from verifiable rewards and could significantly improve learning efficiency.

7.4.4. Computational Efficiency

Leveraging distributed training frameworks such as DeepSpeed (Microsoft, 2025) would enable more extensive training across multiple GPUs, addressing current computational limitations. Efficient exploration strategies and improved tree search algorithms could also reduce the computational overhead of MCTS while maintaining or improving performance.

7.4.5. Test-Time Compute Scaling

The MCTS framework provides a natural foundation for investigating test-time compute scaling in mathematical reasoning. Future work should systematically explore how increased computational budget during inference affects proof success rates, potentially revealing scaling laws specific to mathematical reasoning tasks.

7.4.6. Integration with Formal Methods

Closer integration with Isabelle's existing automation tools could enhance performance. Combining LLM-guided search with traditional automated theorem proving techniques such as simplification, rewriting, and decision procedures might yield more robust proof strategies. This hybrid approach could leverage the strengths of both statistical learning and symbolic reasoning.

The intersection of LLMs and automated theorem proving remains a rich area for research. While current limitations prevent our MCTS approach from achieving its full potential, the foundational framework and identified improvement directions provide a clear path toward more capable automated proof systems. As computational resources and training methodologies continue to advance, we anticipate significant improvements in LLM-based theorem proving capabilities.

A. Appendix

A.1. Prompt Template Structure with Self-Ask Reasoning

This appendix provides a detailed overview of the prompt template structure used in our beam search implementation (see Section 4.3), focusing on the self-ask reasoning paradigm for generating Isabelle tactics.

A.1.1. Template Architecture

Our prompt system uses a modular template architecture that adapts to different proof modes and contexts. The core template for self-ask reasoning is structured as follows:

```
You are an expert Isabelle proof assistant helping to find next tactic.

Theorem: {theorem}
Current proof state: {proof_state}
Mode: {mode} ({mode_description})

Progress so far:
{proof_history}

Failed attempts:
{failed_attempts}

Available lemmas: {available_lemmas}

{mode_specific_instructions}

Based on current state and failures, reason step by step about what tactic
would make most progress. Consider why previous attempts failed and adapt your approach.

Your response MUST end with exactly this format:
TACTIC: <your single isabelle command here>
```

```
The command after TACTIC: must be executable Isabelle syntax with no explanation.
```

A.1.2. Mode-Specific Instructions

The template dynamically inserts mode-specific guidance based on whether Isabelle is in *state mode* (structured proofs) or *prove mode* (tactical proofs):

State Mode Instructions

```
Valid commands in STATE MODE:
- have "statement" - State a fact to prove later
- have "statement" by method - State and prove immediately
- show ?thesis by method - Prove the main goal
- show "statement" by method - Prove a specific statement
- thus "statement" by method - Use 'this' (previous fact) to conclude
- hence "statement" by method - Combines 'then' + 'thus'
- moreover - Add another fact to collection
- ultimately - Use all collected facts
- finally - Conclude the proof
- next - Move to next goal (if multiple)
- { ... } - Proof block for subgoals
- assume "P" - Make assumption in proof

NEVER use 'apply' commands (they are for prove mode)
```

Prove Mode Instructions

```
Valid commands in PROVE MODE:
- Basic: apply simp, apply auto, apply blast, apply force, apply fast
- Arithmetic: apply arith, apply linarith, apply algebra
- Logic: apply metis, apply meson, apply smt
- Induction: apply (induct n), apply (cases x)
- With facts: apply (simp add: assms), apply (auto simp: h1 h2)
- Advanced: apply (simp add: algebra_simps), apply (auto intro: exI)
- Completion: done (after successful apply), qed (ends proof block)
- Structured: proof -, proof (cases), proof (induct x)

NEVER use 'have', 'show', 'fix', 'assume' (they are for state mode)
```

A.1.3. Concrete Example

To illustrate how the template works in practice, consider the following concrete example from our beam search execution:

Scenario

- **Theorem**: `fixes a b :: real assumes h0: 3*a + 2*b = 12 and h1: a = 4 shows b = 0`
- **Current proof state**: `1. b = 0`
- **Mode**: prove (tactical proof mode)
- **Available lemmas**: `h0, h1`
- **Failed tactics**: `[("apply simp", "Failed to apply proof method")]`

Populated Template

```
You are an expert Isabelle proof assistant helping to find next tactic.

Theorem: fixes a b :: real assumes h0: 3*a + 2*b = 12 and h1: a = 4 shows b = 0
Current proof state: 1. b = 0
Mode: prove (tactical proof mode)

Progress so far:
No steps yet - this is the first tactic.

Failed attempts:
- apply simp: Failed to apply proof method

Available lemmas: h0, h1

Valid commands in PROVE MODE:
- Basic: apply simp, apply auto, apply blast, apply force, apply fast
- Arithmetic: apply arith, apply linarith, apply algebra
- Logic: apply metis, apply meson, apply smt
- Induction: apply (induct n), apply (cases x)
- With facts: apply (simp add: assms), apply (auto simp: h1 h2)
- Advanced: apply (simp add: algebra_simps), apply (auto intro: exI)
```

```
- Completion: done (after successful apply), qed (ends proof block)
- Structured: proof -, proof (cases), proof (induct x)

NEVER use 'have', 'show', 'fix', 'assume' (they are for state mode)

Based on current state and failures, reason step by step about what tactic
would make most progress. Consider why previous attempts failed and adapt your approach.

Your response MUST end with exactly this format:
TACTIC: <your single isabelle command here>

The command after TACTIC: must be executable Isabelle syntax with no explanation.
```

Expected Self-Ask Reasoning Response

```
Are follow up questions needed here: Yes.
Follow up: What are the given assumptions?
Intermediate answer: h0 states that 3*a + 2*b = 12, and h1 states that a = 4.
Follow up: Can I substitute h1 into h0 to solve for b?
Intermediate answer: Yes, substituting a = 4 into 3*a + 2*b = 12 gives 12 + 2*b = 12, so b
Follow up: What tactic can use both assumptions together for linear arithmetic?
Intermediate answer: The linarith tactic is designed for linear arithmetic and can use mul
Follow up: How do I give linarith access to both h0 and h1?
Intermediate answer: Use 'using h0 h1 apply linarith' to provide both assumptions to linar
TACTIC: using h0 h1 apply linarith
```

A.1.4. Template Features

The prompt template incorporates several key features:

- **Context awareness**: Includes theorem statement, current proof state, and proof history
- **Failure learning**: Explicitly lists failed tactics to avoid repetition
- **Mode adaptation**: Provides different instructions for state vs. prove mode
- **Resource awareness**: Lists available lemmas and assumptions for tactic construction
- **Structured reasoning**: Enforces step-by-step thinking before tactic generation

- **Format enforcement**: Requires specific output format for reliable parsing

This template structure enables the language model to generate contextually appropriate Isabelle tactics while learning from previous failures and adapting to the current proof mode. The template system is integrated into our beam search implementation as described in Subsection 4.3.2, where it serves as one of the primary mechanisms for generating candidate tactics alongside automated approaches and temperature-based sampling.

A.2. Examples for Proof Completion

A.3. MCTS Prompt Templates and Model Responses

This appendix provides the actual prompt templates and example responses used in our MCTS implementation (see Subsection 4.4.8). The templates are designed for the policy model (tactic generation) and value model (state evaluation) components.

A.3.1. Policy Model Prompt Template

The policy model generates candidate tactics using a few-shot prompt template with XML parsing. The template includes examples and expects responses in a structured XML format:

```
Example:
THEOREM: theorem plus_zero: "n + 0 = n"

PROOF HISTORY:

CURRENT STATE:
1. n + 0 = n

Available lemmas:
  - add_0
  - add_succ
MODE: prove
<response>
<think>
We need to prove n + 0 = n. This follows by induction on n.
</think>
```

```
<command>
apply (induct n)
</command>
</response>

Example:
THEOREM: theorem plus_succ: "n + Suc m = Suc (n + m)"

PROOF HISTORY:

CURRENT STATE:
1. n + Suc m = Suc (n + m)

Available lemmas:
  - add_0
  - add_succ
MODE: prove
<response>
<think>
This requires induction on m to handle the successor on the right side of the equation.
<think>
<command>
apply (induct m)
</command>
</response>

Your turn:
THEOREM: {theorem_statement}

PROOF HISTORY:
{proof_history}

CURRENT STATE:
{current_state}

{available_lemmas}
MODE: {proof_mode}
```

A.3.2. Value Model Prompt Template

The value model evaluates proof states by estimating progress toward completion. The template outputs a numerical score between -1.0 and 1.0:

```
Evaluate the progress of this Isabelle proof step. Output only a score between -1.0 and 1.

THEOREM: {theorem_statement}

PROOF HISTORY:
{proof_history}

EXECUTED COMMAND:
{command}

RESULTING SUBGOALS:
{execution_result}

MODE: {proof_mode}

Score:
```

A.3.3. Example Model Responses

The following examples are taken from actual evaluation runs, showing both successful and failed proof attempts.

Successful Proof Example

Theorem: `mathd_algebra_114: fixes a :: real assumes "a = 8" shows "(16 * (a^2) powr (1 / 3)) powr (1 / 3) = 4"`

Policy Model Response:

```
<response>
<command>
by (simp add: powr_powr)
</command>
```

Execution Result: `success=True is_done=True result='' state_name='1'`
Value Model Score: `1.0` (indicating successful proof completion)

Failed Proof Example

Theorem: `amc12a_2021_p3: fixes x y :: nat assumes h0: "x + y = 17402" and h1: "10 dvd x" and h2: "x div 10 = y" shows "x - y = 14238"`

Policy Model Response:

```
<response>
<command>
have [simp]: "x = 14252" by eval
have [simp]: "y = 2947" by eval
from this show ?thesis by simp
</command>
```

Execution Result: `success=False is_done=True state_name='1' error='Illegal application of proof command in "prove" mode'`

Value Model Score: `-0.1` (indicating failed proof attempt)

Another Successful Example

Theorem: `mathd_algebra_432: fixes x ::real shows "(x + 3) * (2 * x - 6) = 2 * x^2 - 18"`

Policy Model Response:

```
<response>
<command>
by algebra
</command>
```

Execution Result: `success=True is_done=True result='' state_name='1'`

Value Model Score: `1.0` (indicating successful proof completion)

A.3.4. Template Features and Implementation Details

The MCTS prompt templates incorporate several key features:

- **Few-shot learning**: Includes example proofs to guide the model's response format and reasoning style
- **XML parsing**: Uses structured XML tags (`<response>`, `<command>`, `<thinking>`) for reliable output parsing
- **Context awareness**: Includes theorem statement, proof history, current state, and available lemmas

- **Mode adaptation**: Supports both state and prove modes in Isabelle
- **Numerical scoring**: Value model provides quantitative progress estimates for MCTS decision making
- **Temperature control**: Different temperature settings for policy (1.0) and value (0.3) models to balance exploration vs. precision

These templates are implemented in `code/mcts/inference_mcts.py` and integrated with the MCTS search algorithm as described in Section 4.4. The structured output format enables reliable parsing and integration with the Isabelle proof assistant.

A.4. Comparison of Different Approaches

Abbreviations

LLM Large Language Model

ATP Automated Theorem Proving

HOL Higher-Order Logic

FOL First-Order Logic

LCF Logic for Computable Functions

SMT Satisfiability Modulo Theories

PPO Proximal Policy Optimization

GRPO Group Relative Policy Optimization

CoT Chain of Thought

TD Temporal Difference

UCB1 Upper Confidence Bound 1

MCTS Monte Carlo Tree Search

RL Reinforcement Learning

GPU Graphics Processing Unit

List of Figures

List of Tables

Bibliography

2025. afp-2025: Isabelle2025 release of the archive of formal proofs. `https://foss.heptapod.net/isa-afp/afp-2025`.

Janice Ahn, Rishu Verma, Renze Lou, Di Liu, Rui Zhang, and Wenpeng Yin. 2024. Large Language Models for Mathematical Reasoning: Progresses and Challenges. *arXiv preprint*. ArXiv:2402.00157 [cs].

Zhangir Azerbayev, Hailey Schoelkopf, Keiran Paster, Marco Dos Santos, Stephen McAleer, Albert Q. Jiang, Jia Deng, Stella Biderman, and Sean Welleck. 2023. Llemma: An open language model for mathematics. *Preprint*, arXiv:2310.10631.

Richard Bellman. 1984. *Dynamic programming*. Princeton Univ. Pr, Princeton, NJ.

Cameron B. Browne, Edward Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. 2012. A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43.

Davide Castelvecchi. AlphaProof: DEEPMIND AI HITS MILESTONE IN SOLVING MATHS PROBLEMS.

Peter Dayan and CJCH Watkins. 1992. Q-learning. *Machine learning*, 8(3):279–292.

Jürgen Giesl. 2010. *Automated Reasoning: 5th International Joint Conference, IJCAR 2010, Edinburgh, UK, July 16-19, 2010, Proceedings*. Number v.6173 in Lecture Notes in Computer Science Ser. Springer Berlin / Heidelberg, Berlin, Heidelberg.

Sylvain Gugger, Lysandre Debut, Thomas Wolf, Philipp Schmid, Zachary Mueller, Sourab Mangrulkar, Marc Sun, and Benjamin Bossan. 2022. Accelerate: Training and inference at scale made simple, efficient and adaptable. `https://github.com/huggingface/accelerate`.

John Harrison. 2009a. Handbook of practical logic and automated reasoning. *Cambridge University Press*.

John Harrison. 2009b. *Handbook of Practical Logic and Automated Reasoning*, 1 edition. Cambridge University Press.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. 2020. The curious case of neural text degeneration. *Preprint*, arXiv:1904.09751.

Ziwei Ji, Nayeon Lee, Ronja Frieske, Tiezheng Yu, Hui Su, and 1 others. 2023. Survey of hallucination in natural language generation. *ACM Computing Surveys*, 55(1):1–38.

Albert Q. Jiang, Wenda Li, Jesse Michael Han, and Yuhuai Wu. 2021. Lisa: Language models of isabelle proofs. *6th Conference on Artificial Intelligence and Theorem Proving*.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling Laws for Neural Language Models. *arXiv preprint*. ArXiv:2001.08361 [cs].

Levente Kocsis and Csaba Szepesvári. 2006. Bandit Based Monte-Carlo Planning. In David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Dough Tygar, Moshe Y. Vardi, Gerhard Weikum, Johannes Fürnkranz, Tobias Scheffer, and Myra Spiliopoulou, editors, *Machine Learning: ECML 2006*, volume 4212, pages 282–293. Springer Berlin Heidelberg, Berlin, Heidelberg. Series Title: Lecture Notes in Computer Science.

Haoxiong Liu, Jiacheng Sun, Zhenguo Li, and Andrew C Yao. 2025. Proofaug: Efficient neural theorem proving via fine-grained proof structure analysis. *arXiv preprint arXiv:2501.18310*.

Microsoft. 2025. DeepSpeed: Deepspeed is a deep learning optimization library that makes distributed training and inference easy, efficient, and effective. `https://github.com/deepspeedai/DeepSpeed`.

Tobias Nipkow and Gerwin Klein. 2014. *Concrete Semantics: With Isabelle/HOL*. Springer International Publishing, Cham.

Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. 2002a. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*. Springer.

Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. 2002b. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*. Number 2283 in Lecture Notes in Computer Science. Springer-Verlag Berlin Heidelberg, Berlin, Heidelberg.

No Author. 1973. *Symbolic Logic*. Macmillan.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA. Curran Associates Inc.

Lawrence C. Paulson. 1986. Foundation of a generic theorem prover. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence*, volume 1, pages 633–638. Morgan Kaufmann Publishers Inc.

Lawrence C Paulson and Jasmin C Blanchette. 2012. Three years of experience with sledgehammer, a practical link between automatic and interactive theorem provers. 2.

Z. Z. Ren, Zhihong Shao, Junxiao Song, Huajian Xin, Haocheng Wang, Wanjia Zhao, Liyue Zhang, Zhe Fu, Qihao Zhu, Dejian Yang, Z. F. Wu, Zhibin Gou, Shirong Ma, Hongxuan Tang, Yuxuan Liu, Wenjun Gao, Daya Guo, and Chong Ruan. 2025. DeepSeek-Prover-V2: Advancing Formal Mathematical Reasoning via Reinforcement Learning for Subgoal Decomposition. *arXiv preprint*. ArXiv:2504.21801 [cs].

J. A. Robinson. 1965. A Machine-Oriented Logic Based on the Resolution Principle. *Journal of the ACM*, 12(1):23–41.

Arthur L Samuel. 1959. Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 3(3):210–229.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models. *arXiv preprint*. ArXiv:2402.03300 [cs].

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. 2018. AlphaZero: A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419):1140–1144.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George Van Den Driessche, Thore Graepel, and Demis Hassabis. 2017. Mastering the game of Go without human knowledge. *Nature*, 550(7676):354–359.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27.

Richard S. Sutton. 1988. Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44.

Richard S. Sutton and Andrew Barto. 2020. *Reinforcement learning: an introduction*, second edition edition. Adaptive computation and machine learning. The MIT Press, Cambridge, Massachusetts London, England.

Kimi Team, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, Chenzhuang Du, Chonghua Liao, Chuning Tang, Congcong Wang, Dehao Zhang, Enming Yuan, Enzhe Lu, Fengxiang Tang, Flood Sung, Guangda Wei, Guokun Lai, and 77 others. 2025. Kimi k1.5: Scaling Reinforcement Learning with LLMs. *arXiv preprint*. ArXiv:2501.12599 [cs].

Gerald Tesauro and 1 others. 1995. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68.

Trieu H. Trinh, Yuhuai Wu, Quoc V. Le, He He, and Thang Luong. 2024. AlphaGeometry: Solving olympiad geometry without human demonstrations. *Nature*, 625(7995):476–482.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Leandro von Werra, Younes Belkada, Lewis Tunstall, Edward Beeching, Tristan Thrush, Nathan Lambert, Shengyi Huang, Kashif Rasul, and Quentin Gallouédec. TRL: Transformer Reinforcement Learning.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-Consistency Improves Chain of Thought Reasoning in Language Models. *arXiv preprint*. ArXiv:2203.11171 [cs].

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H Chi, Quoc V Le, and Denny Zhou. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models.

Markus Wenzel. 1999. Isabelle/Isar — a versatile environment for human-readable formal proof documents.

William Brown. 2025. Verifiers: Environments for llm reinforcement learning. `https://github.com/PrimeIntellect-ai/verifiers`.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, and 3 others. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Marcin Wrochna. 2024. QIsabelle: Query the isabelle proof assistant with python. `https://github.com/marcinwrochna/qisabelle`.

Yuhuai Wu and 1 others. 2022. Autoformalization with large language models. *arXiv preprint arXiv:2205.12615*.

Huajian Xin, Daya Guo, Zhihong Shao, Zhizhou Ren, Qihao Zhu, Bo Liu, Chong Ruan, Wenda Li, and Xiaodan Liang. 2024a. DeepSeek-Prover: Advancing Theorem Proving in LLMs through Large-Scale Synthetic Data. *arXiv preprint*. ArXiv:2405.14333 [cs].

Huajian Xin, Z. Z. Ren, Junxiao Song, Zhihong Shao, Wanjia Zhao, Haocheng Wang, Bo Liu, Liyue Zhang, Xuan Lu, Qiushi Du, Wenjun Gao, Qihao Zhu, Dejian Yang, Zhibin Gou, Z. F. Wu, Fuli Luo, and Chong Ruan. 2024b. DeepSeek-Prover-V1.5: Harnessing Proof Assistant Feedback for Reinforcement Learning and Monte-Carlo Tree Search. *arXiv preprint*. ArXiv:2408.08152 [cs].

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: deliberate problem solving with large language models. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA. Curran Associates Inc.

Dan Zhang, Sining Zhoubian, Ziniu Hu, Yisong Yue, Yuxiao Dong, and Jie Tang. 2024. Rest-mcts*: Llm self-training via process reward guided tree search. *Advances in Neural Information Processing Systems*, 37:64735–64772.

Kunhao Zheng, Jesse Michael Han, Stanislas Polu, Mark Chen, Junyoung Han, Keiran Chen, Yuhuai Wong, Kiran Narasimhan, Chen Finn, and Miroslav Soljačić. 2022. Minif2f: a cross-system benchmark for formal olympiad-level mathematics. *arXiv preprint*. ArXiv:2109.00110 [cs].

Zhi Zheng, Zhuoliang Xie, Zhenkun Wang, and Bryan Hooi. 2025. Monte Carlo Tree Search for Comprehensive Exploration in LLM-Based Automatic Heuristic Design. *arXiv preprint*. ArXiv:2501.08603 [cs].