OPTIMIZING THE COST-QUALITY TRADEOFF OF AGENTIC THEOREM PROVERS IN LEAN

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models (LLMs) are increasingly used in workflows for generating formal proofs in Lean. These workflows often decompose problems into smaller lemmas, sample many proof attempts, and use compiler feedback to guide search. However, they can be prohibitively expensive, often spending substantial compute on attempts that ultimately fail. In this work, we address this problem with an *action routing agent* that consists of a data plane and a control plane. The data plane generates natural-language lemma decompositions, formalizes them in Lean, and samples proof attempts for the resulting theorem and lemma targets. The control plane observes previous failed Lean attempts, estimates both the likelihood of success and cost of another attempt, and decides whether to continue proving the current target or restart from a new breakdown. On a subset of PutnamBench, our agent decreases the cost by 28.9% over a fixed-step baseline on average, preserving performance while using substantially less compute. These results suggest that failed Lean trajectories provide actionable signals for cost-aware resource allocation in agentic theorem proving.

1 INTRODUCTION

Large language models (LLMs) have made rapid progress in automated theorem proving with Lean (de Moura et al., 2015), with specialized systems obtaining impressive results on challenging benchmarks (Chen et al., 2025c; Varambally et al., 2025; Liu et al., 2026). Compared to natural language reasoning, formal theorem proving offers a unique opportunity for LLMs, since the correctness of generated proofs can be automatically verified by a compiler, removing the need for human experts. At the same time, formal theorem proving is extremely challenging because proofs must be syntactically valid and every small step must be rigorously justified. To overcome this difficulty, recent works built agentic workflows that iteratively decompose problems into lemmas, generate multiple proof attempts for each, and self-correct when necessary (Varambally et al., 2025; Liu et al., 2026).

Agentic theorem provers are expensive. The automatic verifiability of Lean proofs allows these workflows to scale to thousands of attempts per problem, which is critical for achieving high performance on difficult benchmarks. However, they are also prohibitively expensive: Liu et al. (2026) spent $\$50$ per problem to achieve a perfect score on Putnam 2025, and the top performance on PutnamBench reportedly used over $\$40$ per problem (Tsoukalas et al., 2024).

Limitations of fixed-step policies. Existing agents typically scale compute through fixed policies, where every input receives the same pre-specified budgets for decomposition, proof generation, and self-correction. This can waste significant compute on infeasible problems or misformalized statements. The reliance of fixed-step policies thus poses a fundamental efficiency tradeoff: while more attempts improve the probability of success, they also incur high costs that may not be justified by the marginal improvement. Unfortunately, this efficiency perspective has largely been ignored in prior work, which has focused on improving performance rather than optimizing workflow costs.

This work: Routing for improved cost-quality tradeoffs. To mitigate this, we adapt and extend existing works in the routing and cascading literature (Zhuge et al., 2024; Aggarwal et al., 2024; Ramírez et al., 2024) to the setting of Lean theorem proving with agentic workflows. To do so, we propose a novel agent architecture that uses a lightweight router to dynamically evaluate the expected cost, measured as LLM generation compute, and quality of future proof attempts based on

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

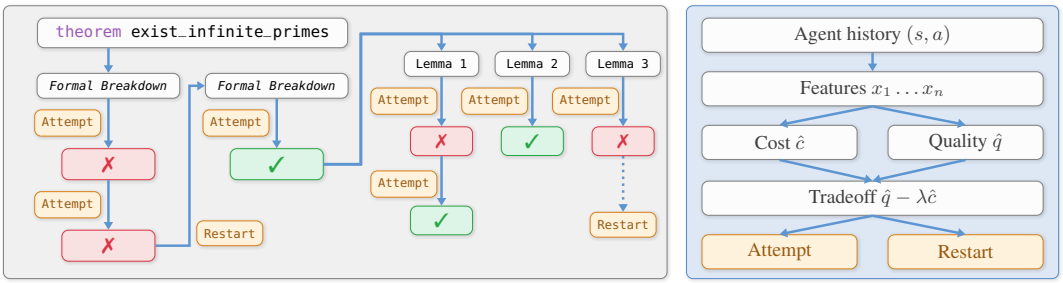

Figure 1: An overview of our approach. As shown on the left, our agent first decomposes the original problem into formal lemmas. It then attempts to prove the theorem, assuming the lemmas are correct. If the proof fails, the agent router (right) observes the trajectory of past attempts, extracts features, and estimates the cost and quality of future attempts. Based on a cost-quality tradeoff, it then decides whether to continue allocating compute to the current target or create a new breakdown. Once a breakdown is accepted, the agent attempts to prove each lemma, with the same router overseeing the proof trajectory and making dynamic routing decisions at each step.

the agent's past trajectory. By learning to identify low-value attempts early, our agent can stop early or restart, significantly reducing unnecessary computation while preserving performance.

Agent architecture. Our architecture conceptually separates the agent into a proof-generation *data plane* (Figure 1, left) and a cost-quality *control plane* (Figure 1, right). Inspired by Chen et al. (2025c), the data plane generates lemma-style Lean proofs by decomposing the original problem into natural-language lemmas, formalizing those lemmas in Lean, and generating proof attempts for the resulting theorem and lemma targets. Meanwhile, the control plane observes the prior proof trajectory and uses a lightweight model to estimate both the success probability and the cost of the next attempt. Based on a target cost-quality tradeoff, it then dynamically determines whether allocating further compute is worthwhile.

Experimental results. We evaluate our routing approach on an 85-problem subset of PutnamBench. We find that our dynamic approach significantly outperforms the fixed-step baseline across multiple cost-quality metrics. On average, our agent reduces computational cost by 28.9% and improves solve rate by 7.9% compared with the fixed-step baseline. These results demonstrate that fixed-step policies leave significant room for efficiency improvements and that more research is needed to lower the cost of agentic theorem proving.

Key contributions. Our key contributions are:

- We identify that fixed-step scaling in Lean agents is critically inefficient, and that reducing unnecessary attempts is key to achieving a better cost-quality tradeoff.
- We develop a lemma-style Lean prover equipped with a cost-quality adaptive router (§3).
- We empirically demonstrate that our proposed router successfully decreases wasted compute while maintaining solve rate (§4).

2 Related Work

Lean benchmarks To evaluate theorem provers in Lean, researchers have developed several benchmarks. MiniF2F (Zheng et al., 2021) and PutnamBench (Tsoukalas et al., 2024) are two widely used evaluation suites for Lean theorem proving, with PutnamBench focusing on problems from the Putnam mathematics competition. More recent benchmarks have expanded the scope of evaluation to include research-level problems (Dekoninck et al., 2026; Jiang et al., 2025).

Lean prover models. To improve capabilities, several efforts have trained specialized models for Lean theorem proving (Polu et al., 2023; Lin et al., 2025a; Wang et al., 2025; Ren et al., 2025; Lin et al., 2025b). These models, including Goedel-Prover-V2 (Lin et al., 2025b), use reinforcement learning techniques on large datasets of proof trajectories to improve proof generation and instill reasoning capabilities. They perform very well on easier benchmarks like MiniF2F, but still struggle with harder or more advanced benchmarks.

Agentic prover systems. To push the performance barrier, recent systems have adopted agentic workflows that make direct use of the exact verifiability of Lean to scale inference compute significantly. These agents recursively decompose problems into smaller lemmas, sample many proof attempts, and use Lean feedback to guide search (Zhou et al., 2025; Chen et al., 2025c; Varambally et al., 2025; Liu et al., 2026; Hubert et al., 2026; Chen et al., 2025a; Mistral AI, 2026; Requena Pozo et al., 2026). They obtain significantly improved solve rates on harder benchmarks, but they also shift the bottleneck toward inference-time compute: many require hundreds of expensive LLM calls per problem, making them prohibitively expensive to run at scale.

Routing and cascading. Routing and cascading are standard approaches for reducing the cost of LLM systems. Routing selects a model or tool for a query according to a learned policy or predictor (Chuang et al., 2025; Ding et al., 2024; Hari & Thomson, 2023). Usually, a query is routed to a specialized model for a specific topic, or to a more expensive model if the query is predicted to be difficult. Instead, cascading runs a sequence of increasingly expensive LLMs and stops once a response passes a quality criterion (Chen et al., 2024; Ramírez et al., 2024). In open-ended natural language tasks, these quality criteria often rely on proxy signals such as confidence, agreement, or response variance (Aggarwal et al., 2024; Jitkrittum et al., 2023).

Routing in agentic environments. Recent work extends routing and cascading to multi-agent LLM systems (Zhuge et al., 2024; Chen et al., 2025b; Yue et al., 2025; Su et al., 2026; Kumaresan, 2026). These methods typically optimize query-level model selection, agent structure, or task allocation across collaborating agents. Our setting differs because we apply routing to the unique environment of agentic theorem proving in Lean, where the router observes structured proof trajectories in an agentic process. To our knowledge, no prior work has explored routing in this setting.

3 Action Routing Lean Agent

Our primary goal is to replace fixed policies with dynamically predicted routing decisions, thereby achieving a better cost-quality tradeoff in Lean agents. This section describes the implementation of our proposed architecture. §3.1 presents the proof-generation data plane, which operates the lemma-style prover and records the trajectories consumed by the control plane. §3.2 describes the cost-quality control plane, which evaluates these trajectories to decide dynamically whether the agent should retry the current target or abandon it.

3.1 Proof-Generation Data Plane

The design of the data plane is based on prior work (Varambally et al., 2025; Chen et al., 2025c). In particular, it is responsible for decomposing the original problem into tractable subgoals, generating proof attempts for those subgoals, and reconstructing the final proof. In addition, it produces the attempt trajectories used by the control plane to make routing decisions. The data plane consists of four modules, illustrated in Figure 2. Detailed prompts of each module are provided in App. D.

Breakdown module. We prompt a general-purpose model (GPT-OSS-20B (Agarwal et al., 2025), with high reasoning effort) to first solve the original problem in natural language. The format of the output breakdown is shown in the breakdown prompt in App. D. The model organizes the proof into a sequence of lemmas, accompanied by concise proof sketches of each lemma and the main theorem. Lemmas may depend on other lemmas in the same breakdown, so each proof sketch specifies which lemma should be used to prove the current target. If the modules described next fail to produce a full proof for a breakdown, the breakdown is discarded, and the agent generates a new one, up to at most n_b repetitions. If all n_b breakdowns are exhausted, the agent terminates.

Formalizer. Next, the agent attempts to formalize the breakdown in Lean. For every natural-language lemma, we sample n_f candidate formalizations from Goedel-Formalizer-V2-8B (Lin et al., 2025b), compile each candidate against Mathlib, and discard any candidate that produces syntax or type errors. If multiple formalizations of the same lemma compile successfully, we pass them to a general-purpose model (GPT-OSS-20B, with low reasoning effort), together with the original natural-language statement and proof sketch. The model then selects the best formalization.

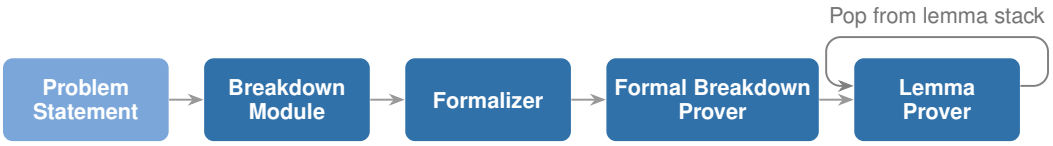

Figure 2: Overview of the architecture of the data plane, illustrating the core modular components alongside the recursive lemma dependency resolution.

Formal breakdown prover. During proof generation, the agent first attempts to prove the top-level theorem using the formalized lemmas as Lean `axioms`, which can be used directly to prove the statement. The natural-language proof sketch is included in the prompt to guide the prover. We use GOEDEL-PROVER-V2-8B (Lin et al., 2025b) to generate proof attempts for this formal breakdown. In Lean, a proof is successful only if it compiles without errors. If a proof attempt fails to compile, the control plane determines whether to sample another attempt, up to a maximum limit, or to terminate the current breakdown. If the control plane terminates the process, the current breakdown is discarded. If the proof compiles successfully, we extract the subset of lemmas invoked in the proof and push them onto a lemma stack for subsequent proof generation.

Lemma prover. Once the formal breakdown has been proven, the agent pops one lemma at a time from the stack and attempts to prove it using the same GOEDEL-PROVER-V2-8B model and control plane logic. To avoid spending compute on unused lemmas, the agent only attempts to prove lemmas that are invoked by an accepted proof of the formal breakdown or by other proven lemmas. The input to the prover includes the formal lemma statement, its natural-language proof sketch, and any dependencies extracted from the sketch, which are provided as axioms. If the attempt limit is reached or the control plane decides to stop, the breakdown is discarded. If the lemma is proven successfully, any new lemmas invoked in its proof are pushed onto the lemma stack. This process continues recursively until the stack is empty. Once it is, the problem has been successfully solved.

3.2 COST-QUALITY CONTROL PLANE

The cost-quality control plane decides whether the agent should continue to prove the current target or terminate the trajectory and move to another candidate breakdown. Its role is to replace a fixed attempt budget with a routing policy that accounts for both the expected benefit and the computational cost of additional proof attempts. In this section, we first formulate the trajectory-level decision problem (§3.2.1), then define the cost-quality objective used by the router (§3.2.2), and finally describe the cost and quality estimators used to implement this objective (§3.2.3).

3.2.1 TRAJECTORY AND ACTION SPACE

We define the object currently being proved as a target, which is either the formal breakdown of the original theorem or an individual lemma produced during decomposition. For a given target, a trajectory consists of the sequence of all failed proof attempts observed so far. Each attempt produces a candidate Lean proof, its compilation result, and auxiliary information such as output length, error messages, and proof structure. The control plane uses this trajectory-level information to decide whether further attempts should be made for the current target or whether the agent should move on to another breakdown. At the beginning of a trajectory, to provide non-trivial information, the agent always samples two proof attempts before the first decision is made.

We denote the information available to the router at a decision point by an abstract state s. This state summarizes the current failed-attempt trajectory and contains whatever quantities are needed by the cost and quality estimators. The concrete information used by two estimators is described in §3.2.3.

We define the action space of the control plane as $A = \{\text{ATTEMPT}, \text{TERMINATE}\}$. The action ATTEMPT samples one additional proof for the current target. The action TERMINATE stops the current trajectory and marks the breakdown as failed.

3.2.2 COST-QUALITY TRADEOFF AND ROUTING

Intuitively, the router should continue attempting a target when the next proof attempt has a sufficiently high chance of success, and terminate when the trajectory is unlikely to yield a valid proof.

At each decision point, the router receives the state s. Let $\hat{q}(s)$ denote the predicted probability that one additional attempt will solve the current target, which we will refer to as quality, and let $\hat{c}(s)$ denote the predicted cost of generating that attempt. Following the cost-quality routing formulation of Dekoninck et al. (2025), we define the utility of one additional attempt as

$$\tau(s) = \hat{q}(s) - \lambda \hat{c}(s),$$

where $\lambda > 0$ controls the relative penalty assigned to computational cost. A positive value of $\tau(s)$ indicates that the expected gain in success probability outweighs the cost of another attempt under the chosen budget preference.

The resulting routing policy is

$$\pi(s) = \begin{cases} \textsc{Attempt}, & \tau(s) > 0, \\ \textsc{Terminate}, & \text{otherwise.} \end{cases}$$

Thus, the router continues proving the current target only when the predicted value of an additional attempt is positive. The hyperparameter λ traces different operating points along the cost-quality frontier. Smaller values of λ make the router more willing to incur high costs, while larger values encourage earlier termination and lower total cost.

3.2.3 Estimating Cost and Quality

To instantiate the policy above, the control plane requires two estimates for each decision: the expected cost $\hat{c}(s)$ of another attempt and the success probability $\hat{q}(s)$ of that attempt. We estimate these quantities separately.

Cost estimator. We use the number of output tokens as a proxy for the computational cost of each attempt. We measure cost as LLM generation compute and treat the cost of Lean compilation as negligible. Within a trajectory, the agent generates proof attempts using the same model and the same target. We therefore assume that attempt costs are independent and identically distributed. Under this assumption, the average cost of previous attempts is an unbiased estimator of the cost of the next attempt. Thus, the estimated cost $\hat{c}(s)$ is computed as the average cost of all previous attempts in the trajectory.

Quality estimator. Estimating $\hat{q}(s)$ is more challenging because the value of an additional attempt depends on both the inherent difficulty of the target and the history of previous failures. We design features based on previous failures that may signal whether the prover is making progress or is stuck in a local failure mode. The design of these features is discussed in §3.2.4. Using these features, along with the number of attempts made so far, we train a logistic regression model to predict the success probability of the next attempt.

Training the quality estimator. To train the estimator, we construct random trajectories on a designated training set of problems using a fixed attempt budget, collecting features at every step from the previous history. Each trajectory terminates either when it reaches the first successful attempt or when the attempt budget is exhausted. The quality estimator is a logistic regression model trained on these trajectories, with the attempt success as the signal and the trajectory features and the number of previous attempts as inputs. The training details are listed in App. A.

3.2.4 Features for Quality Estimation

The quality estimator relies on features extracted from the trajectory of previous proof attempts. These features are designed to capture signals of target difficulty and repeated failure modes. In addition to the number of (failed) attempts in the current trajectory, we use the following features:

Proof similarity. We measure how similar the generated proofs are across attempts. To compute this feature, we first normalize each proof by extracting the proof body, renaming variables into a standard format, and removing comments and redundant whitespace. We then compute the average pairwise fuzzy string ratio among all normalized proofs in the trajectory. A low similarity score suggests that the prover is exploring different proof states and failing in different ways, which might indicate that the model is confused and has low intrinsic confidence.

Error diversity. We compute the ratio of unique Lean compilation error messages to the total number of compilation errors observed across all attempts. A low diversity ratio indicates that the prover repeatedly encounters the same error, suggesting a persistent failure mode. In contrast, a high diversity ratio suggests that the prover is exploring different strategies and failing in different ways.

3.3 Extension to Other Lean Agents

Although we instantiate routing in our own open-source prover, the same control plane principle can be applied to other Lean agents (Varambally et al., 2025; Chen et al., 2025c). The key idea is to separate the agent's proof-generation mechanism from the routing policy that decides which action is optimal. This separation makes the framework extensible to agents with richer action spaces, multiple prover models, or additional self-correction. We do not explore these extensions experimentally due to the high cost of evaluating these agents.

Action space. The action space need not be restricted to ATTEMPT and TERMINATE. For example, an agent could include an action that switches to a stronger prover model when the current model appears to be struggling. Another possible action is self-correction, where the agent uses Lean error messages to repair a failed proof attempt. Such an action may be especially useful when the errors suggest syntactic misuse of Lean rather than a genuine logical gap. If the agent supports recursive decomposition, the action space could also include an action that breaks the current target into smaller subgoals instead of abandoning the breakdown entirely.

Optimal policy. With a larger action space, the same cost-quality principle can be generalized by assigning each candidate action an estimated quality gain and cost. The policy then selects the action with the highest predicted utility:

$$\pi(s) = \arg\max_{a \in A} \hat{q}(a, s) - \lambda \hat{c}(a, s).$$

This formulation preserves the same routing behavior while allowing the router to choose among multiple proof-generation strategies.

Cost and quality estimation. The cost and quality estimators can also be generalized by adapting their inputs to the structure of the agent. When multiple prover models are available, cost can be standardized according to the parameter count. Similarly, the quality estimator can incorporate additional trajectory features that reflect the behavior of new modules. These changes affect only the estimator design and do not change the underlying logic of the control plane.

4 Experiments

We evaluate our agent on a subset of PutnamBench, focusing on its ability to improve the cost-quality tradeoff of Lean proof generation. Specifically, we test whether our agent can reduce unnecessary proof attempts without substantially lowering the solve rate, compare its performance with a fixed-step agent and an oracle agent that has access to ground-truth success probabilities.

4.1 Experimental Setup

We now explain our experimental setup, with additional details provided in App. A.

Dataset. We evaluate on the subset of 85 PutnamBench problems reportedly solved by GOEDEL-PROVER-V2-32B with pass@192.[1] These 85 problems are randomly split into training and test sets, containing 42 and 43 problems, respectively. The router's quality estimator is trained on the training set, while results reported in this section are evaluated on the test set.

Fixed-step baseline. The fixed-step baseline uses the same data plane pipeline, but replaces adaptive routing with a static policy. Specifically, it allocates a fixed maximum of k proof attempts to each active target before terminating the current breakdown and moving to the next one.

[1] GOEDEL-PROVER-V2-32B reportedly solves 86 problems (Lin et al., 2025b). However, only 85 are listed in the PutnamBench leaderboard list of solved problems. See the full list here.

Oracle agent baseline. The goal of our agent is to terminate hard targets early, before spending many failed proof attempts on targets that are unlikely to be solved. Consequently, the gains available from routing depend on the distribution of easy and difficult targets in the evaluation set. We contextualize our results by including an oracle agent that has access to the ground-truth success probability of the current target. This success probability replaces the trajectory features used by our agent, while the router still uses the same logistic quality estimator with the number of previous attempts as an input. The oracle is a privileged information baseline that measures how much our adaptive routing strategy could improve with perfect target-difficulty information.

Evaluation metrics. We measure computational cost in scaled floating point operations (SFLOPs), defined as the product of the model's effective parameter count and the number of generated output tokens. We trace cost-solve rate curves by varying λ from $2{\cdot}10^{-8}$ to $4{\cdot}10^{-6}$ for our agent and the oracle, and k over $\{1, \ldots, 64\}$ for the fixed-step baseline, with at most eight breakdowns per problem. We summarize these curves using two area-under-the-curve (AUC) metrics: *cost AUC*, which integrates cost over solve rate (lower is better), and *solve rate AUC*, which integrates solve rate over cost (higher is better). To compute these metrics, we include the origin in every curve and add the $\lambda = 0$ operating points for our agent and the oracle. We then interpolate linearly between points and integrate over the range shared by the methods being compared. We report improvements relative to the fixed-step baseline for routing experiments and whole-proof generation for the data-plane comparison (§4.3.3). For each metric, let A and A_{ref} denote the AUCs of the method and its reference, respectively. The relative reduction in cost AUC, $1 - A/A_{\text{ref}}$, measures the savings in average cost at matched solve rates. The relative increase in solve rate AUC, $A/A_{\text{ref}} - 1$, measures the gain in average solve rate at matched costs.

4.2 Main Results

Comparison to baselines. Figure 3 compares our agent with the fixed-step baseline and the oracle router. Our agent consistently improves the cost-quality tradeoff relative to the fixed-step baseline, reducing cost AUC by 28.9% and increasing solve rate AUC by 7.9%.

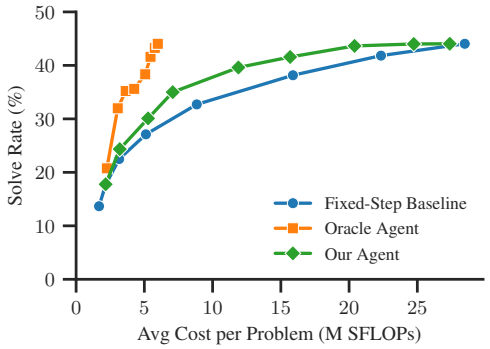

Figure 3: Cost-quality curve of our agent compared with the fixed-step baseline and the zero-noise oracle router.

Point comparisons. To achieve a high solve rate, the fixed-step baseline solves 44.0% of problems using 28.5 M SFLOPs, whereas our agent solves 43.6% using 20.4 M SFLOPs. This corresponds to a 28.4% cost reduction with only a 0.4% decrease in solve rate. In a medium-budget regime, the fixed-step baseline solves 41.8% of problems using 22.3 M SFLOPs, while our agent solves 41.6% of problems using only 15.7 M SFLOPs. Relative to these points, our agent reduces cost by 29.6% with only a 0.2% decrease in solve rate.

Contextualizing results. The oracle agent achieves a substantially stronger cost-quality tradeoff than both the fixed-step baseline and our agent. This result shows that accurately identifying hard targets is highly valuable for adaptive routing. The gap between our agent and the oracle reference suggests that better trajectory features or stronger quality estimators could further improve the router's ability to terminate low-value targets early. Overall, these results show that fixed-step policies leave significant room for efficiency improvements, and that the routing strategy can eliminate unnecessary proof attempts while preserving most of the solve-rate performance.

4.3 Ablation Studies

We now present a series of ablation studies to further analyze the performance of the routing strategy and its components. We first evaluate noisy oracle variants to understand the sensitivity of adaptive

Table 1: Cost AUC reduction relative to the fixed-step baseline across varying levels of artificial noise.

Model Configuration	Cost AUC Reduction
0% noise (oracle signal)	62.0%
1% noise	57.5%
2% noise	42.9%
3% noise	35.7%
4% noise	32.4%
Our agent	28.9%
5% noise	28.2%
10% noise	17.0%

Table 2: Cost AUC reduction relative to the fixed-step baseline for different inputs to the quality estimator.

Model Configuration	Cost AUC Reduction
All three inputs (our agent)	28.9%
Proof similarity + error diversity	23.6%
Error diversity + attempt count	20.3%
Proof similarity + attempt count	19.7%
Attempt count only	13.1%

routing to quality-estimation accuracy (§4.3.1). We then analyze the contribution of different features in the quality estimation model (§4.3.2). At the end of this section, we isolate the contribution of the data plane architecture by comparing it against a whole-proof generation approach (§4.3.3).

4.3.1 Noisy Oracle Agent

We evaluate noisy oracle variants by perturbing the ground-truth success probability with different levels of noise. These variants allow us to study how routing performance degrades as the target-difficulty signal becomes less reliable, and to assess how effectively our agent recovers difficulty information from proof trajectories. We perturb the ground-truth success probability of each proof target using different levels of Gaussian or uniform noise. Given a ground-truth success probability of p, we refer to a Gaussian noise level of 10% as $p_n = p + \varepsilon$, where $\varepsilon \sim \mathcal{N}(0, 0.1)$, and clamp the value to $[10^{-6}, 1 - 10^{-6}]$. We map the perturbed probability p_n with a logit mapping ($\text{logit}(p_n) = \log p_n/(1 - p_n)$) and feed it as a feature to the quality estimator. Table 1 reports the cost AUC reduction of each noisy oracle relative to the fixed-step baseline.

The zero-noise oracle, which represents the theoretical ceiling for this estimator family, yields a 62.0% reduction in cost AUC relative to the fixed-step baseline. As the noise level increases, performance gradually decreases. Our agent achieves a 28.9% reduction in cost AUC, placing it between the 5% noise oracle, which achieves 28.2%, and the 4% noise oracle, which achieves 32.4%. This suggests that the real proof-trajectory features recover a meaningful fraction of the information available to an oracle target-difficulty estimator.

We provide a further analysis of the feature coefficients of the quality estimators in App. B.

4.3.2 Quality Estimator Ablation

We next analyze the contribution of different features used by the quality estimator. We compare the full feature set against removing different features. Table 2 reports the cost AUC reduction of each model relative to the fixed-step baseline. As can be clearly seen in Table 2, all inputs significantly aid our agent in improving the cost-quality tradeoff, with error diversity being the most important feature.

4.3.3 Data Plane Performance

Finally, we isolate the contribution of the data plane architecture. In addition to the control plane's dynamic routing decisions, the data plane itself substantially improves over a whole-proof generation approach, in which a specialized prover model directly generates complete proofs without the agentic decomposition architecture. In the whole-proof generation baseline, we prompt the Goedel-Prover-V2 models to generate entire proofs in a single step, report pass@k, with k ranging from 1 to 256. We provide a detailed description of the whole-proof baseline in App. C.

As shown in Figure 4, when using the same prover model GOEDEL-PROVER-V2-8B, the data plane pipeline achieves a substantially higher solve rate: its performance is close to that of the stronger GOEDEL-PROVER-V2-32B model. Concretely, the data plane pipeline increases solve rate AUC by 88.2% over whole-proof generation with GOEDEL-PROVER-V2-8B, while remaining 23.2% below whole-proof generation with GOEDEL-PROVER-V2-32B. At the high-budget end, the data plane solves 48 of the 85 problems in our PutnamBench subset with GOEDEL-PROVER-V2-8B as the prover model, up from the 23 solved by the same model under whole-proof generation at pass@256 (App. C). These results demonstrate the effectiveness of the data plane design.

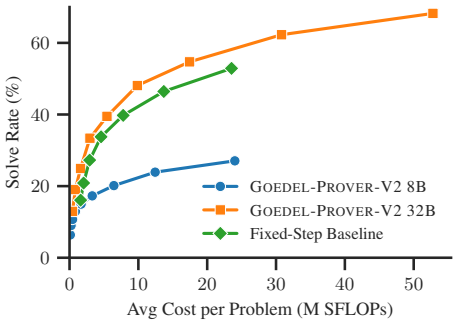

Figure 4: Comparison between the fixed-step data plane pipeline and whole-proof generation.

5 LIMITATIONS AND FUTURE WORK

This work studies adaptive routing as a first step toward cost-aware control for agentic formal mathematical proving. In this section, we discuss several limitations of the current system and open directions for future work.

Small benchmark size. Our evaluation is limited to a subset of PutnamBench because running the full agent on larger benchmarks is computationally infeasible. We estimate our runs on the 85 problem subset to have cost around 1600-2000 GPU hours on NVIDIA GH200, meaning that if we were to scale to the full PutnamBench, the cost would be roughly eightfold, around 12000-16000 GPU hours. However, this subset is well matched to the capability of the underlying prover and thus provides a useful setting for studying routing behavior.

Gap to oracle routing. The proposed routing system also leaves a substantial gap to the oracle router. This gap suggests that the current cost and quality estimators capture only part of the information needed for effective routing. New reward formulations, better trajectory features, and stronger estimator architectures may further improve the prediction.

Limited action space. The current implementation is also restricted to a binary action space: the router decides whether to sample another proof attempt for the current target or terminate it. As discussed in §3.3, the same framework could be extended to larger action spaces and more general agentic architectures. In such settings, however, routing becomes more difficult because the agent must decide not only whether to continue, but also which action to take next. Future work could study how to effectively implement adaptive routing for more sophisticated Lean agents with actions such as model switching, self-correction, and recursive decomposition.

6 CONCLUSIONS

In this work, we introduced an action routing agent that uses failed proof trajectories to estimate the cost and marginal value of an additional proof attempt. The agent then uses these estimates to decide whether to continue proving the current target or terminate the current trajectory and restart from a new breakdown. On a subset of PutnamBench, our agent reduces computational cost by 28.9% and improves solve rate by 7.9% on average compared with the fixed-step baseline. Overall, these results suggest that cost-aware control policies are a promising approach for making formal theorem proving more efficient. Instead of optimizing efficiency of the underlying prover or the number of tokens used by each model call, we show that optimizing on the action level can also lead to significant improvements.

AI USE STATEMENT

In this work, we used generative AI tools to aid and polish writing, to support qualitative data analysis, and to interpret results. We have not used generative AI tools for any other required disclosure task, and the rest of the required disclosure tasks are not applicable to this work. Additionally, we used generative AI tools for code generation and to polish writing.

We have extensively reviewed all AI-assisted work. This includes reviewing and editing all text generated by AI and verifying the correctness of all code generated by two authors through extensive testing and manual verification of code and model outputs. Qualitative conclusions extracted by AI were manually verified and checked for correctness. No AI was used to create any of the main ideas or scientific contributions. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This research does not involve human subjects or sensitive data. The work focuses on improving the efficiency and effectiveness of theorem proving in Lean, which is a mathematical proof assistant. The methods and results presented in this paper are intended to advance the field of automated theorem proving and do not raise any ethical concerns related to privacy, consent, or potential harm to individuals or communities.

REPRODUCIBILITY STATEMENT

We describe our method in detail in this paper, and provide the reviewers with access to our code for reproducing our results. This repository also includes detailed README instructions for running the experiments, along with a dashboard to view and analyze the results. As the creators of PutnamBench explicitly prohibit distribution of formal proofs of the problems, we do not include the proofs generated by our agent in the repository.

REFERENCES

Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K. Arora, Yu Bai, Bowen Baker, Haiming Bao, Boaz Barak, Ally Bennett, Tyler Bertao, Nivedita Brett, Eugene Brevdo, Greg Brockman, Sébastien Bubeck, Che Chang, Kai Chen, Mark Chen, Enoch Cheung, Aidan Clark, Dan Cook, Marat Dukhan, Casey Dvorak, Kevin Fives, Vlad Fomenko, Timur Garipov, Kristian Georgiev, Mia Glaese, Tarun Gogineni, Adam P. Goucher, Lukas Gross, Katia Gil Guzman, John Hallman, Jackie Hehir, Johannes Heidecke, Alec Helyar, Haitang Hu, Romain Huet, Jacob Huh, Saachi Jain, Zach Johnson, Chris Koch, Irina Kofman, Dominik Kundel, Jason Kwon, Volodymyr Kyrylov, Elaine Ya Le, Guillaume Leclerc, James Park Lennon, Scott Lessans, Mario Lezcano Casado, Yuanzhi Li, Zhuohan Li, Ji Lin, Jordan Liss, Lily Liu, Jiancheng Liu, Kevin Lu, Chris Lu, Zoran Martinovic, Lindsay McCallum, Josh McGrath, Scott McKinney, Aidan McLaughlin, Song Mei, Steve Mostovoy, Tong Mu, Gideon Myles, Alexander Neitz, Alex Nichol, Jakub Pachocki, Alex Paino, Dana Palmie, Ashley Pantuliano, Giambattista Parascandolo, Jongsoo Park, Leher Pathak, Carolina Paz, Ludovic Peran, Dmitry Pimenov, Michelle Pokrass, Elizabeth Proehl, Huida Qiu, Gaby Raila, Filippo Raso, Hongyu Ren, Kimmy Richardson, David Robinson, Bob Rotsted, Hadi Salman, Suvansh Sanjeev, Max Schwarzer, D. Sculley, Harshit Sikchi, Kendal Simon, Karan Singhal, and Yang Song. gpt-oss-120b & gpt-oss-20b model card. *CoRR*, abs/2508.10925, 2025. doi: 10.48550/ARXIV.2508.10925. URL https://doi.org/10.48550/arXiv.2508.10925.

Pranjal Aggarwal, Aman Madaan, Ankit Anand, Srividya Pranavi Potharaju, Swaroop Mishra, Pei Zhou, Aditya Gupta, Dheeraj Rajagopal, Karthik Kappaganthu, Yiming Yang, Shyam Upadhyay, Manaal Faruqui, and Mausam. Automix: Automatically mixing language models. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/ecda225cb187b40ea8edc1f46b03ffda-Abstract-Conference.html.

Jiangjie Chen, Wenxiang Chen, Jiacheng Du, Jinyi Hu, Zhicheng Jiang, Allan Jie, Xiaoran Jin, Xing Jin, Chenggang Li, Wenlei Shi, Zhihong Wang, Mingxuan Wang, Chenrui Wei, Shufa Wei, Huajian Xin, Fan Yang, Weihao Gao, Zheng Yuan, Tianyang Zhan, Zeyu Zheng, Tianxi Zhou, and Thomas Hanwen Zhu. Seed-prover 1.5: Mastering undergraduate-level theorem proving via learning from experience. *CoRR*, abs/2512.17260, 2025a. doi: 10.48550/ARXIV.2512.17260. URL https://doi.org/10.48550/arXiv.2512.17260.

Lingjiao Chen, Matei Zaharia, and James Zou. Frugalgpt: How to use large language models while reducing cost and improving performance. *Transactions on Machine Learning Research*, 2024.

Lingjiao Chen, Jared Quincy Davis, Boris Hanin, Peter Bailis, Matei Zaharia, James Zou, and Ion Stoica. Optimizing model selection for compound AI systems. *CoRR*, abs/2502.14815, 2025b. doi: 10.48550/ARXIV.2502.14815. URL https://doi.org/10.48550/arXiv.2502.14815.

Luoxin Chen, Jinming Gu, Liankai Huang, Wenhao Huang, Zhicheng Jiang, Allan Jie, Xiaoran Jin, Xing Jin, Chenggang Li, Kaijing Ma, Cheng Ren, Jiawei Shen, Wenlei Shi, Tong Sun, He Sun, Jiahui Wang, Siran Wang, Zhihong Wang, Chenrui Wei, Shufa Wei, Yonghui Wu, Yuchen Wu, Yihang Xia, Huajian Xin, Fan Yang, Huaiyuan Ying, Hongyi Yuan, Zheng Yuan, Tianyang Zhan, Chi Zhang, Yue Zhang, Ge Zhang, Tianyun Zhao, Jianqiu Zhao, Yichi Zhou, and Thomas Hanwen Zhu. Seed-prover: Deep and broad reasoning for automated theorem proving. *CoRR*, abs/2507.23726, 2025c. doi: 10.48550/ARXIV.2507.23726. URL https://doi.org/10.48550/arXiv.2507.23726.

Yu-Neng Chuang, Prathusha Kameswara Sarma, Parikshit Gopalan, John Boccio, Sara Bolouki, Xia Hu, and Helen Zhou. Learning to route llms with confidence tokens. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, Proceedings of Machine Learning Research. PMLR / OpenReview.net, 2025. URL https://proceedings.mlr.press/v267/chuang25b.html.

Leonardo de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn, and Jakob von Raumer. The lean theorem prover (system description). In Amy P. Felty and Aart Middeldorp (eds.), *Automated*

Deduction - CADE-25, pp. 378–388, Cham, 2015. Springer International Publishing. ISBN 978-3-319-21401-6.

Jasper Dekoninck, Maximilian Baader, and Martin T. Vechev. A unified approach to routing and cascading for llms. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, Proceedings of Machine Learning Research. PMLR / OpenReview.net, 2025. URL https://proceedings.mlr.press/v267/dekoninck25a.html.

Jasper Dekoninck, Nikola Jovanović, Tim Gehrunger, Kári Rögnvaldsson, Ivo Petrov, Chenhao Sun, and Martin Vechev. Beyond benchmarks: Matharena as an evaluation platform for mathematics with llms. 2026. URL https://arxiv.org/abs/2605.00674.

Dujian Ding, Ankur Mallick, Chi Wang, Robert Sim, Subhabrata Mukherjee, Victor Rühle, Laks V. S. Lakshmanan, and Ahmed Hassan Awadallah. Hybrid LLM: cost-efficient and quality-aware query routing. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=02f3mUtqnM.

Surya Narayanan Hari and Matt Thomson. Tryage: Real-time, intelligent routing of user prompts to large language models. *CoRR*, abs/2308.11601, 2023. doi: 10.48550/ARXIV.2308.11601. URL https://doi.org/10.48550/arXiv.2308.11601.

Thomas Hubert, Rishi Mehta, Laurent Sartran, et al. Olympiad-level formal mathematical reasoning with reinforcement learning. *Nature*, 651:607–613, 2026. doi: 10.1038/s41586-025-09833-y. URL https://doi.org/10.1038/s41586-025-09833-y.

Jiedong Jiang, Wanyi He, Yuefeng Wang, Guoxiong Gao, Yongle Hu, Jingting Wang, Nailin Guan, Peihao Wu, Chunbo Dai, Liang Xiao, and Bin Dong. FATE: A formal benchmark series for frontier algebra of multiple difficulty levels. *CoRR*, abs/2511.02872, 2025. doi: 10.48550/ARXIV.2511.02872. URL https://doi.org/10.48550/arXiv.2511.02872.

Wittawat Jitkrittum, Neha Gupta, Aditya Krishna Menon, Harikrishna Narasimhan, Ankit Singh Rawat, and Sanjiv Kumar. When does confidence-based cascade deferral suffice? In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/1f09e1ee5035a4c3fe38a5681cae5815-Abstract-Conference.html.

Ramchand Kumaresan. ACAR: adaptive complexity routing for multi-model ensembles with auditable decision traces. *CoRR*, abs/2602.21231, 2026. doi: 10.48550/ARXIV.2602.21231. URL https://doi.org/10.48550/arXiv.2602.21231.

Yong Lin, Shange Tang, Bohan Lyu, Jiayun Wu, Hongzhou Lin, Kaiyu Yang, Jia Li, Mengzhou Xia, Danqi Chen, Sanjeev Arora, and Chi Jin. Goedel-prover: A frontier model for open-source automated theorem proving. *CoRR*, abs/2502.07640, 2025a. doi: 10.48550/ARXIV.2502.07640. URL https://doi.org/10.48550/arXiv.2502.07640.

Yong Lin, Shange Tang, Bohan Lyu, Ziran Yang, Jui-Hui Chung, Haoyu Zhao, Lai Jiang, Yihan Geng, Jiawei Ge, Jingruo Sun, Jiayun Wu, Jiri Gesi, Ximing Lu, David Acuna, Kaiyu Yang, Hongzhou Lin, Yejin Choi, Danqi Chen, Sanjeev Arora, and Chi Jin. Goedel-prover-v2: Scaling formal theorem proving with scaffolded data synthesis and self-correction. *CoRR*, abs/2508.03613, 2025b. doi: 10.48550/ARXIV.2508.03613. URL https://doi.org/10.48550/arXiv.2508.03613.

Junqi Liu, Zihao Zhou, Zekai Zhu, Marco Dos Santos, Weikun He, Jiawei Liu, Ran Wang, Yunzhou Xie, Junqiao Zhao, Qiufeng Wang, Lihong Zhi, Jia Li, and Wenda Li. Numina-lean-agent: An open and general agentic reasoning system for formal mathematics. *arXiv preprint arXiv:2601.14027*, 2026.

Mistral AI. Leanstral: Open-source foundation for trustworthy vibe-coding, March 2026. URL https://mistral.ai/news/leanstral.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Andreas Müller, Joel Nothman, Gilles Louppe, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in python, 2018. URL https://arxiv.org/abs/1201.0490.

Stanislas Polu, Jesse Michael Han, Kunhao Zheng, Mantas Baksys, Igor Babuschkin, and Ilya Sutskever. Formal mathematics statement curriculum learning. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/forum?id=-P7G-8dmSh4.

Guillem Ramírez, Alexandra Birch, and Ivan Titov. Optimising calls to large language models with uncertainty-based two-tier selection. *CoRR*, abs/2405.02134, 2024. doi: 10.48550/ARXIV.2405.02134. URL https://doi.org/10.48550/arXiv.2405.02134.

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. Deepseek-prover-v2: Advancing formal mathematical reasoning via reinforcement learning for subgoal decomposition. *CoRR*, abs/2504.21801, 2025. doi: 10.48550/ARXIV.2504.21801. URL https://doi.org/10.48550/arXiv.2504.21801.

Borja Requena Pozo, Austin Letson, Krystian Nowakowski, Izan Beltran Ferreiro, and Leopoldo Sarra. A minimal agent for automated theorem proving. 2026. URL https://arxiv.org/abs/2602.24273.

Jinwei Su, Qizhen Lan, Yinghui Xia, Lifan Sun, Weiyou Tian, Tianyu Shi, and Lewei He. Difficulty-aware agentic orchestration for query-specific multi-agent workflows. In Hakim Hacid, Yoelle Maarek, Francesco Bonchi, Ido Guy, and Emine Yilmaz (eds.), *Proceedings of the ACM Web Conference 2026, WWW 2026, Dubai, United Arab Emirates, originally scheduled for April 13-17, 2026, rescheduled for June 29 - July 3, 2026*, pp. 2060–2070. ACM, 2026. doi: 10.1145/3774904.3792240. URL https://doi.org/10.1145/3774904.3792240.

George Tsoukalas, Jasper Lee, John Jennings, Jimmy Xin, Michelle Ding, Michael Jennings, Amitayush Thakur, and Swarat Chaudhuri. Putnambench: Evaluating neural theorem-provers on the putnam mathematical competition. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/1582eaf9e0cf349e1e5a6ee453100aa1-Abstract-Datasets_and_Benchmarks_Track.html.

Sumanth Varambally, Thomas Voice, Yanchao Sun, Zhifeng Chen, Rose Yu, and Ke Ye. Hilbert: Recursively building formal proofs with informal reasoning. *arXiv preprint arXiv:2509.22819*, 2025.

Haiming Wang, Mert Unsal, Xiaohan Lin, Mantas Baksys, Junqi Liu, Marco Dos Santos, Flood Sung, Marina Vinyes, Zhenzhe Ying, Zekai Zhu, Jianqiao Lu, Hugues de Saxcé, Bolton Bailey, Chendong Song, Chenjun Xiao, Dehao Zhang, Ebony Zhang, Frederick Pu, Han Zhu, Jiawei Liu, Jonas Bayer, Julien Michel, Longhui Yu, Léo Dreyfus-Schmidt, Lewis Tunstall, Luigi Pagani, Moreira Machado, Pauline Bourigault, Ran Wang, Stanislas Polu, Thibaut Barroyer, Wen-Ding Li, Yazhe Niu, Yann Fleureau, Yangyang Hu, Zhouliang Yu, Zihan Wang, Zhilin Yang, Zhengying Liu, and Jia Li. Kimina-prover preview: Towards large formal reasoning models with reinforcement learning. 2025. URL http://arxiv.org/abs/2504.11354.

Yanwei Yue, Guibin Zhang, Boyang Liu, Guancheng Wan, Kun Wang, Dawei Cheng, and Yiyan Qi. Masrouter: Learning to route llms for multi-agent systems. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2025, Vienna,*

Austria, July 27 - August 1, 2025, pp. 15549–15572. Association for Computational Linguistics, 2025. URL https://aclanthology.org/2025.acl-long.757/.

Kunhao Zheng, Jesse Michael Han, and Stanislas Polu. Minif2f: a cross-system benchmark for formal olympiad-level mathematics. *arXiv preprint arXiv:2109.00110*, 2021.

Yichi Zhou, Jianqiu Zhao, Yongxin Zhang, Bohan Wang, Siran Wang, Luoxin Chen, Jiahui Wang, Haowei Chen, Allan Jie, Xinbo Zhang, Haocheng Wang, Luong Quoc Trung, Rong Ye, Phan Nhat Hoang, Huishuai Zhang, Peng Sun, and Hang Li. Solving formal math problems by decomposition and iterative reflection. *CoRR*, abs/2507.15225, 2025. doi: 10.48550/ARXIV.2507.15225. URL https://doi.org/10.48550/arXiv.2507.15225.

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. Gptswarm: Language agents as optimizable graphs. In Ruslan Salakhutdinov, Zico Kolter, Katherine A. Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, Proceedings of Machine Learning Research, pp. 62743–62767. PMLR / OpenReview.net, 2024. URL https://proceedings.mlr.press/v235/zhuge24a.html.

A Experimental Details

Training data. To ensure our experiments are non-contaminated, we split the 85 problem subset of PutnamBench (see §4.1) into a training set of 42 problems and a test set of 43 problems. The training set is used to train the quality estimator, while all results reported in §4 are evaluated on the test set. The train-test split is randomly generated, and the split can be found in the supplementary materials.

Quality estimator. The quality estimator is a single logistic regression model trained on trajectories with attempt success as the binary target. Its input consists of two parts: the trajectory features described in §3.2.4, and an attempt-count feature $1/m$, where m is the number of previous attempts on the current target. We standardize all inputs to zero mean and unit variance with `scikit-learn`'s `StandardScaler` (Pedregosa et al., 2018) and use its logistic regression implementation with default hyperparameters.

Monte Carlo simulation. To obtain a robust estimate of the cost-quality curve for a given λ, we first pre-generate all proof attempts, sampling 64 attempts per target. We then run Monte Carlo simulations over 64 seeds of the routing policy on each problem, allowing at most eight breakdowns per problem. In each simulation, the environment reshuffles the order of the breakdowns and of the pre-generated attempts, so that the routing policy is evaluated over a diverse set of trajectories. We apply the same procedure to the fixed-step baseline, replacing the cost penalty λ with the per-target attempt budget k. For the oracle router, the ground-truth success rate of a target is the fraction of successful attempts among its 64 generated attempts.

Routing hyperparameters. We sweep the cost penalty λ (§3.2.2) over 64 seeds with at most eight breakdowns per problem over

$$\{2\cdot10^{-8},\ 4\cdot10^{-8},\ 6\cdot10^{-8},\ 8\cdot10^{-8},\ 1\cdot10^{-7},\ 1.5\cdot10^{-7},\ 2\cdot10^{-7},\ 4\cdot10^{-7},\ 4\cdot10^{-6}\},$$

where $\lambda = 0$ always continues the current target. The fixed-step baseline sweeps the per-target attempt budget $k \in \{1, 2, 4, 8, 16, 32, 64\}$ with at most eight breakdowns per problem.

Language models. The prover Goedel-Prover-V2-8B, formalizer Goedel-Formalizer-V2-8B, and general-purpose model GPT-OSS-20B all use temperature 1.0. The two 8B models additionally use top-$p = 0.95$. We set the maximum generation length to 40960 tokens for the 8B models and 81920 for GPT-OSS-20B.

B Quality Estimator Analysis

The learned coefficients of the quality estimator, shown in Table 3, offer some interesting insights. As expected, the model predicts a lower success probability as more attempts accumulate on a target. Both proof similarity and error diversity correlate positively with success: the model favors attempts whose proofs are similar to each other but whose errors are diverse. This seems paradoxical at first. We hypothesize that a shared high-level structure across proofs (high proof similarity) signals that the model is confident in the overall proof strategy, while the diversity of errors reflects variation in low-level details rather than a single recurring mistake. If every attempt fails in the same way, the model is stuck and unlikely to solve the problem.

Table 3: Learned logistic regression coefficients for the quality estimator. Note that the model parameters are scaled to mean 0 and variance 1.

Feature	Our Agent
Intercept	−4.85
Proof Similarity	0.27
Error Diversity	1.03
Attempt Count ($1/x$ mapped)	0.34

C WHOLE-PROOF GENERATION

To analyze and later optimize the cost-quality tradeoff of Lean prover models, it is insightful to first analyze the cost-quality tradeoff of the open-weight GOEDEL-PROVER-V2 models. These results serve as a baseline for our other experiments.

C.1 EXPERIMENTAL SETUP

We set up an experiment on the same subset of PutnamBench as we used in the main experiment, attempting each problem 256 times, and examining the success rate of the 8B and 32B prover models.

To create a robust estimate of the cost-quality curve per model, for each problem, we shuffle the order of the attempts and loop through them sequentially one by one, until we have generated a successful attempt or we have reached the maximum number of attempts. To compute the cost-quality curve of the model, we report $pass@k$ for varying k from 1 to 256, running 64 seeds per k to create stable measurements.

C.2 RESULTS

The cost-quality curve of the two models on PutnamBench can be seen in Figure 5. We see that for the PutnamBench, GOEDEL-PROVER-V2-32B outperforms GOEDEL-PROVER-V2-8B on the entire data set in terms of the cost-quality tradeoff. At 9.66 M SFLOPs per problem, GOEDEL-PROVER-V2-8B solves 20% of problems, while at 8.15 M SFLOPs per problem, GOEDEL-PROVER-V2-32B solves 40%, showing double performance at a lower cost. We note that Lin et al. (2025b) report 25 problems solved by GOEDEL-PROVER-V2-8B at pass@32 under whole-proof generation, compared to the 23 we observe here at pass@256 on our 85-problem subset of PutnamBench.

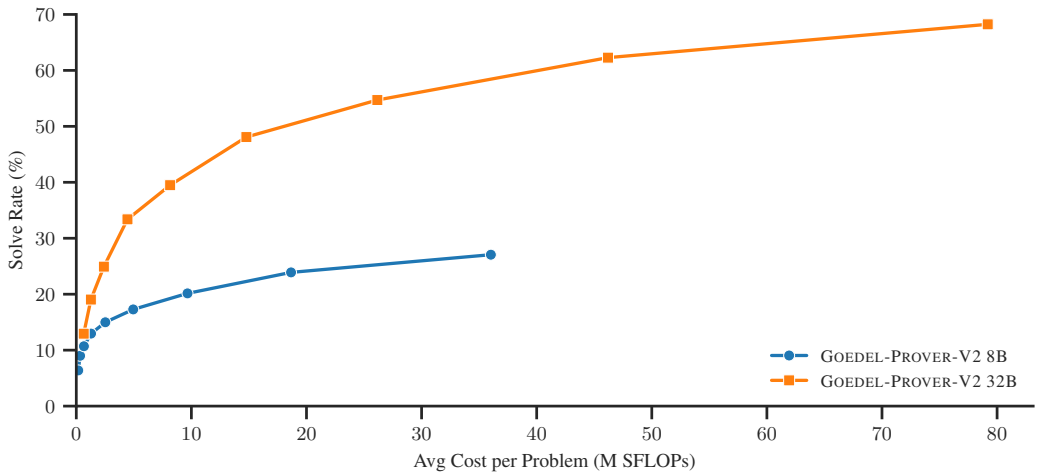

Figure 5: Whole-proof cost-quality curves on PutnamBench.

D DATA PLANE PROMPTS

This section lists the prompts used for the different components of the agentic prover system described in §3.1.

Breakdown Module Prompt

```
Break the following Lean problem down into lemmas, then describe how to combine the lemmas into a full
solution:
```

```
{formal_statement}

**Important points**

1. Make every lemma **mathematically correct** and easy to prove.
2. Do not create redundant lemmas or lemmas that restate the assumptions of the problem
3. In every lemma, also present everything we need to assume to prove the lemma in an isolated manner
4. Make the final solution that combines the lemmas as trivial as possible, i.e. try to put all logic into
the lemmas

The concrete breakdown format should be a valid json. The output format and what you need to include is the
 following:

```json
{{
    "lemmas": [
        {{
            "id": <lemma number>,
            "statement": <lemma statement>,
            "assumption": <state the necessary assumptions for the lemma>,
            "proof": <idea of the proof of the lemma in natural language, if you need to use other lemmas,
specify it in the proof idea>
        }}
    ],
    "theorem": {{
        "statement": <repeat the problem statement>,
        "proof": <idea of the proof, how to combine the lemmas into the final solution>
    }}
}}
```

Important notice: In lemmas' and theorem's proof ideas, explicitly specify them if you need to use
other lemmas' conclusion.

The output can only contain the json above, no other content is allowed.
```

Formalizer Prompt

```
This is an informal breakdown of the math problem we have at hand:

{informal_breakdown}

Your task is to formalize Lemma {i}. To reiterate, here is the statement, the assumptions and the proof
idea of Lemma {i}:

**Statement:** {statement}

**Assumptions:** {assumption}

**Proof Idea:** {proof}

The statement should have the following format. Please only state lemma {i}, DO NOT try to formalize the
final result of the problem.

```lean4
theorem lemma{i} : ... := by sorry
```
```

Formalization Selection Prompt

```
Here is a mathematical problem:

**Problem Statement:** {original_statement}

An LLM has been prompted to formalize the following lemma which is supposed to be a substep of the theorem:

**Statement:** {statement}

**Assumptions:** {assumption}

**Proof Idea:** {proof}

Here are four separate formalizations of the lemma:

```
```
```

```
{lemma_code}
```

Choose the best matching formalization to the lemma according to the statement and the assumptions.

Output ONLY the index number (zero-based) of the best formalization, nothing else. Do not include any explanation after the number.

For example, if formalization 2 is best, output only:
```
2
```
```

**Theorem/Lemma Prover Prompt**

```
Complete the following Lean 4 code:

```lean4
{formal_statement}
```

Before producing the Lean 4 code to formally prove the given theorem, give an overview of the lemmas and how they can contribute to solving the problem. The lemmas mentioned correspond to the axioms and should be assumed proven (they will be proven in subsequent steps so don't worry about them being marked as axioms). Restate the proof idea and possibly develop some underdeveloped steps. Here is the proof idea:

{informal_prefix}

Very Important: The lemmas should be combined in the way the proof idea suggests. Trust the provided solution.

Your answer must contain the whole Lean 4 proof of the problem in a code block in the following format:

```lean4
import Mathlib

{formal_statement}
<Your proof here>
```
```
```