DECOMPOSED ONLINE LEARNING FOR ADAPTIVE LLM INFERENCE IN FORMAL MATHEMATICS

Anonymous authors
Paper under double-blind review

ABSTRACT

Recent advances in agentic mathematical reasoning show that frontier Large Language Models (LLMs) can solve highly challenging mathematical problems. However, these successes often require substantial inference-time compute, making them prohibitively expensive for routine theorem proving. Crucially, subgoals within a proof vary significantly in difficulty, and many can be solved by smaller, more efficient models. We present *METIS*, a budget-aware agentic framework that uses online learning to reduce inference cost while preserving overall capability. Rather than relying on a fixed model, we use feedback from the Lean compiler to (1) decompose the problem into sub-lemmas, (2) adaptively assign budget-efficient, capable models to individual sub-lemmas using a non-stationary online algorithm, and (3) reassemble the complete proof. The decomposition improves sample complexity by generating more tasks for the online learner, and improves predictive power since sub-lemmas are simpler than the original problem. Comprehensive evaluation across four benchmarks demonstrates that *METIS* improves the trade-off between inference cost and proof resolution rate. Specifically, *METIS* achieves an accuracy comparable to the base Claude-Opus-4.8 model while requiring 68.3% less budget on average. Furthermore, *METIS* matches the strongest model selection baseline accuracy while reducing total budget expenditure by an absolute 13.6%. Furthermore, under unstable inference environments, our framework maintains robust performance, with up to twofold lower standard deviation in both inference accuracy and budget than static methods.

1 INTRODUCTION

Large Language Models (LLMs) have demonstrated significant progress in formal theorem proving (Wei et al., 2022; Kojima et al., 2022; Yu et al., 2024; Xu et al., 2025; Moura & Ullrich, 2021), with frontier systems achieving strong performance on international Olympiad-level mathematical problems (Huang & Yang, 2025). More recently, these capabilities have extended to the boundaries of contemporary mathematical research, such as Claude's formalization of Fermat's Last Theorem in Lean (Anthropic, 2026a). Such advances increasingly rely on agentic inference, where capable LLMs iteratively construct, verify, and refine proofs through interaction with formal environments (Ospanov et al., 2025a; Kung et al., 2026; Liu et al., 2026).

Despite their strong capabilities, such agentic workflows can require substantial computational resources; for example, the formalization of Fermat's Last Theorem consumed approximately six billion tokens (Anthropic, 2026a). Meanwhile, decomposing complex problems into smaller, tractable subgoals has proven effective for tackling challenging formal tasks (Ren et al., 2025; Chen et al., 2025; Kung et al., 2026). These subgoals can vary significantly in difficulty, and some can be potentially solved by smaller models (Ren et al., 2025). Using the same high-capacity model for every subgoal can therefore incur unnecessary cost, motivating model allocation at the individual proof obligations.

Recent work has explored model routing as a means of reducing inference cost by selecting among models with different capabilities and costs (Feng et al., 2026). These methods typically rely on a pre-trained or fine-tuned procedure to determine model assignments (Aggarwal et al., 2024; Li, 2025; Feng et al., 2025; Ong et al., 2025). However, they typically operate at the query level, assigning a model before the internal structure of the problem is exposed. This coarse-grained formulation limits their ability to exploit the heterogeneous proof obligations that emerge during theorem proving.

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.

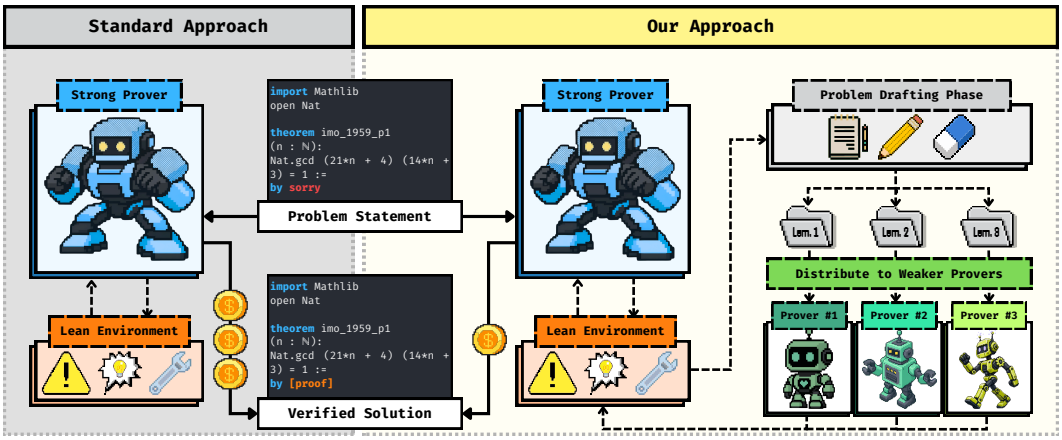


Figure 1: Overview of standard agentic theorem proving and our proposed *METIS* . The standard approach uses a strong prover throughout the proof-generation process. In contrast, *METIS* uses the strong prover to first draft and decompose the proof, then delegates the resulting subgoals to lower-cost models before assembling and verifying the complete proof in Lean.

Furthermore, existing routers typically assume a static inference environment and cannot readily adapt to changes in model cost or availability. In practice, access to closed-weight models is commonly provided through third-party inference platforms such as OpenRouter (OpenRouter, 2026), where model availability, and pricing can vary over time. Existing routers trained on fixed data cannot readily adapt to evolving conditions without repeated retraining or recalibration, which can be impractical.

We address these challenges by introducing *METIS*, a decomposition-based online learning framework for cost-efficient formal theorem proving. Building on the modular structure and verifiability of Lean proofs, *METIS* first uses a strong base model to construct a proof skeleton comprising independently solvable proof obligations. It then applies a cost-aware bandit algorithm (Lattimore & Szepesvári, 2020; Slivkins, 2019) to select models for each obligations using compiler-verified success and cost as feedback. This enables training-free model selection without requiring a fine-tuned router. The resulting subproofs are assembled and verified in Lean. To adapt to changes in model cost and availability, *METIS* employs discounted feedback (Kocsis & Szepesvári, 2006; Garivier & Moulines, 2011), which downweights stale observations and adapts model allocation online. Figure 1 overviews *METIS* in comparison with standard agentic proving workflow.

We evaluate *METIS* on four benchmarks covering diverse domains and difficulty levels, against seven baselines. Compared with the base Claude Opus-4.8 (Anthropic, 2026b), *METIS* reduces inference cost by 68.3% on average while maintaining comparable accuracy. Against existing routing methods, averaged across FATE-M (Shen et al., 2025) and miniF2F-v2s (Ospanov et al., 2025b), *METIS* outperforms the strongest routing baseline, CausalLM (Ong et al., 2025), reducing inference cost by 12.5% while improving accuracy by 0.3 percentage points. Replaying historical OpenRouter uptime traces (Wu, 2026), we further show that *METIS* adapts to deteriorating DeepSeek availability by shifting requests to alternative models while maintaining a stable accuracy–cost trade-off. These results demonstrate the effectiveness of our method for cost-efficient and adaptive formal theorem proving. Our primary contributions are as follows:

- We introduce *METIS*, the first Lean-driven, decomposition-based framework for training-free, cost-aware model selection at the level of individual proof obligations.
- We develop a discounted bandit-based selection strategy that adapts to changes in model cost and availability without retraining, maintaining stable performance under non-stationary model access.
- We construct *METIS-DS*, a dataset of decomposed Lean proof obligations and model trajectories for studying model selection and routing in formal theorem proving.
- We comprehensively evaluate *METIS* across four benchmarks against seven baselines. *METIS* matches Claude Opus-4.8 at 68.3% lower inference cost and, against existing baselines, improves accuracy by 9.8 percentage points while reducing cost by 29.8% on average.

2 Related Works

Automatic Theorem Proving. In recent years, Large Language Models (LLMs) have made remarkable progress in formal reasoning. Goedel Prover v2 (Lin et al., 2026) introduces verifier-guided training over large synthetic corpora. Kimina-Prover Wang et al. (2025) leverages structured reasoning patterns alongside reinforcement learning to improve sample efficiency. DeepSeek-Prover-V2 (Xin et al., 2025) employs sub-goal decomposition-based reasoning, which prioritizes dividing theorems into more tractable sub-problems and discharging them sequentially. Similarly, Seed-Prover (Chen et al., 2025) adopts a sub-lemma proving approach with layered, multi-level inference. Finally, Aristotle (Achim et al., 2025) integrates Lean proof search with dedicated geometry solvers to achieve IMO-level problem solving.

LLM agents for formal mathematics. With the emergence of LLMs equipped with broad world knowledge, numerous agentic frameworks have been developed for automated theorem proving. For instance, APOLLO (Ospanov et al., 2025a) leverages feedback from the Lean compiler to recursively repair faulty proofs rather than re-sampling fresh attempts from scratch. Similarly, LEAP (Kung et al., 2026) focuses on decomposing problems into smaller sub-goals and discharging them through direct interaction with the Lean compiler. Other frameworks integrate specialized coding environments: Numina Lean Agent (Liu et al., 2026) uses Claude Code (Anthropic, 2025) as a base to integrate LLMs with Lean-MCP, LeanDex, an informal proving module, and a discussion partner via separate LLM tools, while ARCHON (Gao et al., 2026) employs agentic loops powered by Claude Code and Codex (Chen et al., 2021) for long-horizon Lean autoformalization. Finally, several works bridge informal and formal reasoning: DSP (Jiang et al., 2023) and DSP+ (Cao et al., 2025) interleave informal proof sketching with formal verification, whereas HERMES (Ospanov et al., 2026) incorporates the Lean compiler into the informal reasoning process to decompose informal proofs into formal sub-steps.

LLM routing. RouteLLM (Ong et al., 2025) trains a router model based on human preference data to perform query routing. LLMRouter (Feng et al., 2026) introduces a collection of LLM- and non-LLM-based routers for query and single agent-turn routing purposes. FrugalGPT (Chen et al., 2023) uses a cascading approach where the query is sent to a weaker model first and then escalated to stronger models. Similarly, AutoMix (Aggarwal et al., 2024) performs an initial self-verification using a weaker model before passing the task to a stronger LLM. Another approach is LLM Shepherding (Dong et al., 2026), in which the authors propose starting the generation with a strong LLM and finishing and cleaning up the proof using weaker models. Finally, Policy-Guided Stepwise Model Routing (Si et al., 2026) combines reinforcement learning with threshold tuning for stepwise math routing.

Additional related work is discussed in Appendix E.

3 Preliminaries

3.1 Online model selection

We formulate model selection as a non-stationary multi-armed bandit problem. Let $\mathcal{G}_1, \ldots, \mathcal{G}_m$ denote the available models, each treated as a black-box arm. Each round t corresponds to an attempt to discharge a decomposed Lean sub-lemma. At round t, the selection policy selects an arm $I_t \in [m]$, corresponding to model $\mathcal{G}_{I_t}$. Let $X_t(i)$ denote the reward associated with arm i at round t, with expected reward $\mu_t(i) = \mathbb{E}[X_t(i)]$. We define the regret over T rounds as

$$\mathrm{Regret}(T) = \sum_{t=1}^{T} \Big(\max_{i \in [m]} \mu_t(i) - \mu_t(I_t) \Big). \tag{1}$$

At each round, $\max_{i\in[m]} \mu_t(i) - \mu_t(I_t)$ measures the expected reward lost relative to selecting the best available model. Thus, maximizing the expected cumulative reward is equivalent to minimizing the regret. Unlike in a stationary setting, the expected rewards $\mu_t(i)$ may vary as the proof obligations, inference costs, and model availability change throughout the proving process (Slivkins, 2019; Lattimore & Szepesvári, 2020).

Cost-aware feedback. Each attempt by model i in round t produces a success indicator $s_t(i) \in \{0,1\}$ and a cost $c_t(i) \geq 0$. We set $s_t(i) = 1$ only if the Lean file compiles and the attempt reduces the number of `sorry` placeholders to zero (compilation with unchanged proof gaps does not count as success). We normalize the cost as $\widetilde{c}_t(i) = \min(c_t(i)/c_{\text{ref}}, 1)$ for a reference cost $c_{\text{ref}} > 0$. To penalize expensive queries, we define the reward for arm i inline as $X_t(i) = s_t(i)\big(1 - \lambda\widetilde{c}_t(i)\big)$, where $\lambda \in [0,1]$. The realized reward for the chosen model I_t is therefore $R_t = X_t(I_t)$. Under this formulation, failures receive zero reward regardless of cost, while cheaper successful attempts yield proportionally higher rewards. Since the realized reward R_t lies in $[0,1]$, we take $B = 1$ below.

3.2 Cost-Aware Discounted UCB

Stationary UCB (Auer et al., 2002) averages all past observations equally and can react slowly after a change in model quality or cost. We instead use Discounted UCB (D-UCB) (Kocsis & Szepesvári, 2006; Garivier & Moulines, 2011), which geometrically downweights old feedback. After every arm has been tried once, the statistics available before round t are

$$N_t(\gamma, i) = \sum_{s=1}^{t-1} \gamma^{t-1-s} \mathbf{1}\{I_s = i\}, \qquad \overline{X}_t(\gamma, i) = \frac{1}{N_t(\gamma, i)} \sum_{s=1}^{t-1} \gamma^{t-1-s} R_s \mathbf{1}\{I_s = i\} \tag{2}$$

where $\gamma \in (0,1]$ is the discount factor. The next model is selected by

$$I_t \in \arg\max_{i \in [m]} \left[\overline{X}_t(\gamma, i) + 2B \sqrt{\frac{\xi \log n_t(\gamma)}{N_t(\gamma, i)}} \right], \qquad n_t(\gamma) = \sum_{i=1}^{m} N_t(\gamma, i), \tag{3}$$

where $\xi > 0$ controls exploration. When $\gamma = 1$, the policy reduces to stationary UCB; when $\gamma < 1$, older observations gradually lose influence, allowing the router to track changes.

We call D-UCB applied to the reward $X_t(i)$ *Cost-Aware Discounted UCB* (CAD-UCB). Cost enters through the reward rather than as an additional term. The preferred arm is therefore the model with the best current trade-off between proof success and the inference cost of its successful attempts.

Proposition 1 (Regret with multiplicative cost)**.** *For each model i and round t, let $p_t(i) = \mathbb{E}[s_t(i)]$ be its success probability and $q_t(i) = \mathbb{E}[s_t(i)\widetilde{c}_t(i)]$ its expected success-weighted normalized cost. Then the mean reward optimized by CAD-UCB is*

$$\mu_t(i) = p_t(i) - \lambda q_t(i) \in \big[(1-\lambda)p_t(i),\, p_t(i)\big] \subseteq [0,1]. \tag{4}$$

Suppose that the rewards $\{X_t(i)\}_{t \geq 1}$ are independent across rounds; the mean vector $(\mu_t(i))_{i=1}^m$ is piecewise constant with $\Upsilon_T \geq 1$ breakpoints before round T; and every positive gap $\mu_t^\star - \mu_t(i)$ is at least $\Delta_{\min} > 0$, where $\mu_t^\star = \max_i \mu_t(i)$. If $\xi > 1/2$, then CAD-UCB with $\gamma = 1 - \frac{1}{4}\sqrt{\frac{\Upsilon_T}{T}}$ satisfies

$$\mathbb{E}[\text{Regret}(T)] \leq C\sqrt{T\Upsilon_T}\log T, \tag{5}$$

The proof is given in Appendix A.

4 Our Approach

In this section, we present *METIS*, a cost-aware framework for formal theorem proving in Lean. *METIS* is designed to be pluggable into modern agent harnesses and consists of two stages: (1) problem-drafting phase and (2) CAD-UCB online-learning phase for model selection. Figure2 illustrates the overall framework. Our implementation builds on the Lean interaction tools provided by Numina-Lean-Agent(Liu et al., 2026) and the Claude Code harness (Anthropic, 2025).

The included Lean tools are LeanExplore, Loogle, REPL (Lean FRO, 2023), and Lean-MCP (Liu et al., 2026). The REPL backend is based on the Kimina-Lean server implementation (Wang et al., 2025). Unless stated otherwise, this is our experimental setup. The Lean compiler returns binary output for a given proof attempt. We refer to the main model driving the harness as the "main" or "base" model, and to smaller models invoked on sub-problems as "weaker" or "sub-" models.

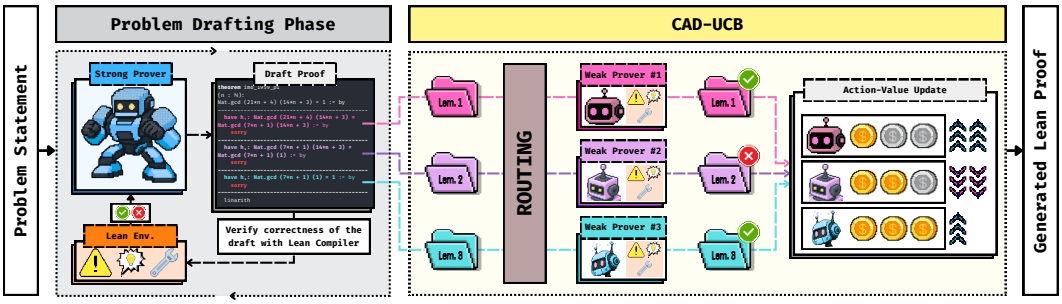


Figure 2: Overview of the *METIS* framework. A strong prover first constructs and verifies a Lean proof skeleton with unresolved proof obligations. CAD-UCB then routes these obligations to lower-cost, capable provers, using compiler-verified success and inference cost as reward to update the model-selection policy. The resulting subproofs are assembled into the final Lean proof.

Problem drafting. In this phase, we prompt the base model to produce a draft solution to the given problem. We observe that the model already knows an approximate successful strategy for solving a problem. Following previous work (Dong et al., 2026), we start generation with a powerful model that simplifies the task and drafts sub-problems to prove. To check this proof skeleton, we verify whether the draft compiles, ignoring pending `sorry` statements for now. Similar to previous agent implementations, we perform targeted repair if the draft fails the compiler test. After the model finalizes the proof strategy, it sends the solution to the next phase: CAD-UCB.

CAD-UCB implementation. At the heart of CAD-UCB are exponential averaging and the UCB bandit algorithm. This choice is motivated by the non-stationarity of the models: most LLMs are now accessed through API and cloud environments across many providers. Each provider has different pricing policies, caching rules, and model uptimes. All of these factors contribute to the non-stationarity of the arms $\mathcal{G}_1, \ldots, \mathcal{G}_m$, so we use a slightly modified version of UCB that maintains discounted counts and means (equation 2) and selects an arm $I_t \in [m]$ via the optimistic index (equation 3). Online learning starts with a warm-up phase in which the algorithm tries all models to determine a routing strategy; the more samples it sees, the more accurate the routing strategy becomes. Since it is a plug-in that does not require extensive compute or extra LLM calls, it incurs no additional cost and can be implemented at various stages of generation. In this work we specifically use it to prove smaller, easier sub-problems proposed by the base agent. As shown in Figure 2, we have several weaker models, each equipped with the same functionality as the base model except for the CAD-UCB routing. These models may use tools and consult the Lean compiler. If a prover closes the goal, its reward R_t is adjusted and recorded. For example, weak prover #1 is the cheapest of the three and was able to close Lemma 1, which contributes to a much larger reward. On the other hand, model #3, which also closed a lemma, gets a smaller reward, since it spent more budget. Prover #2 receives no reward, since it failed to close the goal. Overall, we provide a small sandbox for each prover to maximize the chance of a successful generation. In case of failure, we give one more attempt to a different model, following a cascading approach (Aggarwal et al., 2024).

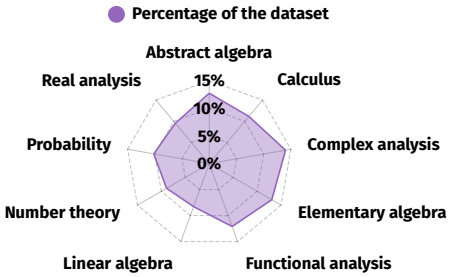


Figure 3: *METIS-DS* spider plot.

***METIS-DS*: A Lean routing dataset.** Alongside *METIS*, we also release *METIS-DS*—a collection of problems from the decomposed DeepSeek-ProverBench (Ren et al., 2025) dataset. The dataset comprises 290 sub-lemmas and four model outputs: DeepSeek-V4-Flash, DeepSeek-V4-Pro (Xu et al., 2026), GPT-5.6-Luna (OpenAI, 2026), and GLM-4.7-Flash (Zeng et al., 2025), with a total of $\sim$13.4K lines of Lean code across nine math fields. We detail the distribution of math topics within *METIS-DS* in Figure 3. This dataset can be used to analyze the outputs of the tested models, tool-use statistics, model behavior, and routing capability. Detailed model performances across topics are in the Appendix.

Table 1: Performance of various *decomposed sub-problem* routing methods on the FATE-M and miniF2F-v2s (IMO subset) datasets. The savings are reported relative to the "No Routing" baseline. Per problem budget was set to $20. The best routing results are **bolded**.

Inference Method	FATE-M			miniF2F-v2s (IMO subset)		
	Accuracy (%)	Cost (USD)	Savings ($\Delta\%$)	Accuracy (%)	Cost (USD))	Savings ($\Delta\%$)
Rule-based baselines						
No Routing	93.3	$250.56	\	75.0[1]	$369.41[1]	\
Cheapest LLM	61.3	$183.35	26.8	75.0	$132.90	64.0
Decomposed sub-problem-level routing baselines						
Random Router	86.0	$104.50	58.3	70.0	$153.09	58.6
kNN Router	82.0	$139.69	44.3	80.0	$164.23	55.5
MF Router	63.3	$183.85	26.6	70.0	$131.01	64.5
Graph Router	60.7	$207.37	17.2	70.0	$158.13	57.2
CausalLM	88.7	$147.93	41.0	**85.0**	$82.46	77.7
Proposed method						
METIS	**89.3**	**$98.24**	**60.8**	**85.0**	**$75.18**	**79.6**

5 EXPERIMENTS

5.1 EXPERIMENTAL SETUP

***METIS* setup.** For evaluation, we use a single-generation setting in which each proving attempt is a single run with a turn budget $K_{\max}$. We set $K_{\max} = 100$ for the base model and $K_{\max} = 40$ for the weaker (sub-) models selected by CAD-UCB to solve drafted sub-lemmas. Online learning begins with a warm-up phase on *METIS-DS*. We use Claude-Opus-4-8 (Anthropic, 2026b) as the base model and DeepSeek-V4-Flash, DeepSeek-V4-Pro, GPT-5.6-Luna, and GLM-4.7-Flash as the weaker models. Reported costs for *METIS* includes drafting and routing phases together. The drafting phase was fixed for every run (i.e., every router faced the same sub-problems) to ensure a fair experimental setup. Additional implementation details are provided in Appendix C.

Baseline methods. We compare our approach with several baselines that route requests from larger to smaller models. In particular, we adopt four methods from the LLMRouter repository (Feng et al., 2026): kNN (Li, 2025), which uses embeddings and k-nearest neighbors to identify a cost-effective model; the MF router, which uses matrix factorization to assign a model based on the dot product between query and model embeddings; GraphRouter (Feng et al., 2025), which represents LLMs as nodes and interactions as edges to match model capabilities to the current query; and CausalLM (Ong et al., 2025), a fine-tuned model that learns routing decisions from past data. We follow the implementation, parameter-tuning, and model-selection procedures described in the original papers. Unlike *METIS*, these learned baselines are *static*: they require a training or construction phase, whereas *METIS* can be deployed without prior observations. We train the static-router baselines on an adapted version of *METIS-DS*. We also include a no-routing baseline that uses the base model, a random router that samples uniformly from the candidate cheaper models, and a rule-based policy that selects the cheapest available LLM.

Evaluation benchmarks. We conduct experiments on four benchmarks: (1) FATE-M (Shen et al., 2025), a Lean benchmark that primarily focuses on algebraic problems; (2) miniF2F-v2s (Ospanov et al., 2025b) (IMO subset), a corrected version of the original miniF2F (Zheng et al., 2021) consisting of only IMO problems; (3) Putnam 2025, a collection of 12 problems from the Putnam competition, for which we use the version of Liu et al. (2026); (4) DeepSeek-ProverBench, a collection of Lean problems covering nine math fields, from which we subsample a small portion, with the rest used as the warm-up and ablation dataset *METIS-DS*.

5.2 ACCURACY AND COST COMPARISON WITH THE BASELINES

Table 1 reports accuracy and cost on FATE-M and miniF2F-v2s with the problem-level routing, in which the router selects a model for each sub-problem. On FATE-M, the base model achieves the highest cost with a 4.0-percentage-point advantage over the next-best method. Across our experiments, *METIS* achieves the highest routing accuracy, improving on the strongest baseline by 7.5

[1]The per-problem budget was set to $50. Please see Appendix F for more details.

Table 2: Performance of various *query level* routing methods on the FATE-M and miniF2F-v2s (IMO subset) datasets. The savings are reported relative to the "No Routing" baseline. Per problem budget was set to $20. The best routing results are **bolded**.

Inference Method	FATE-M			miniF2F-v2s (IMO subset)		
	Accuracy (%)	Cost (USD)	Savings (Δ%)	Accuracy (%)	Cost (USD))	Savings (Δ%)
No Routing	93.3	$250.56	\	75.0[1]	$369.41[1]	\
Query-level routing baselines						
kNN Router	60.0	$30.98	87.6	35.0	$11.91	96.8
MF Router	43.3	$58.87	76.5	15.0	$7.74	97.9
Graph Router	63.3	$8.68	96.5	15.0	$7.99	97.8
CausalLM	78.0	$7.97	96.8	**85.0**	$4.73	98.7
Proposed method						
CAD-UCB	**88.7**	**$7.49**	**97.0**	**85.0**	**$3.96**	**98.9**

Table 3: Performance of various routing methods on the Putnam 2025 dataset. The model distribution shows the fraction of requests routed to smaller models, and savings are reported relative to the "No Routing" baseline. Per problem budget was set to $50. The best results are **bolded**.

Inference Method	Accuracy (out of 12)	Cost (USD)	Savings (Δ%)	Model Distribution (%)			
				DS-V4F	DS-V4P	GLM-4.7F	GPT-5.6L
No Routing	5	$528.23	\	\	\	\	\
Routing Methods							
Cheapest LLM	0	$335.30	36.5	0.0	0.0	100.0	0.0
Random Router	4	$304.34	42.4	20.8	30.2	24.7	24.3
METIS	**7**	**$295.96**	**44.0**	71.1	13.3	1.6	14.0

percentage points. On FATE-M, it also reduces cost by 60.8% relative to no routing. This efficiency largely reflects its preference for DeepSeek-V4-Flash, which offers a favorable balance between Lean capability and inference cost. Appendix F provides a more detailed breakdown of the routing distribution.

We further evaluate *METIS* on the more challenging IMO-level subset of miniF2F-v2s. We set a budget cap of $20 per problem for the routing methods and $50 for the base model. Despite its higher budget, Opus-4.8 solves fewer problems than kNN, CausalLM, and *METIS* because some runs terminate after exhausting the budget. The base model spends a substantial portion of its budget inspecting files, planning proof strategies, and closing sub-goals. On the other hand, due to decomposed problems being easier, cheaper models could find smaller local proofs. On miniF2F-v2s, CausalLM and *METIS* tie for the highest accuracy, with *METIS* using slightly less budget.

To isolate the benefit of routing decomposed sub-problems, we also evaluate the baselines at the query level in Table 2, with the base model excluded from the candidate pool. To adapt CAD-UCB for the query-level baseline, we redefine each round as a single attempt on the complete, undecomposed problem, where the selected model receives a reward based on the final proof compilation and its associated total inference cost. The trained routers remain unchanged; before proving begins, each router assigns a single model to execute the entire agentic pipeline. In this setting, we allow five attempts rather than one to increase the chance of finding a valid proof. Query-level routing uses substantially less budget, but its accuracy is also considerably lower than that of the base model. Drafting and decomposing the proof first simplifies the overarching proof and enables smaller models to close unresolved goals.

Finally, to push the limits of *METIS*, we report results on Putnam 2025 in Table 3. The main bottleneck in this experiment is the number of turns and repairs needed for Putnam problems, and under our budget and compute restrictions, none of the baselines could solve all 12 problems. The base model solved 5 out of 12 problems, while *METIS* solved two more problems at $\sim$56% of the budget. The results are consistent to the miniF2F IMO subset run, where the bigger model hits the budget wall and is unable to continue, leaving sub-goals unproved. Routing baselines rarely hit the budget limit and use more repair self-correction loops to close out the sub-goals. We discuss the behavior of *METIS* trajectories on Putnam problems in Appendix I.

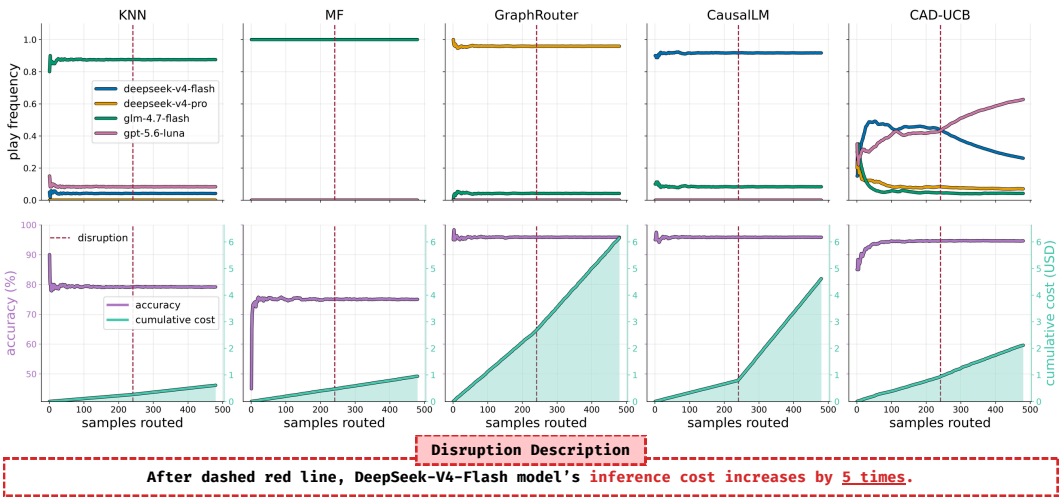


Figure 4: Dynamic-catalog replay for kNN, MF, Graph, CausalLM, and CAD-UCB routing. At the dashed red line, the inference cost of DeepSeek-V4-Flash increases fivefold. The upper plots show arm choice distribution and the lower plot shows changes in accuracy and cumulative cost.

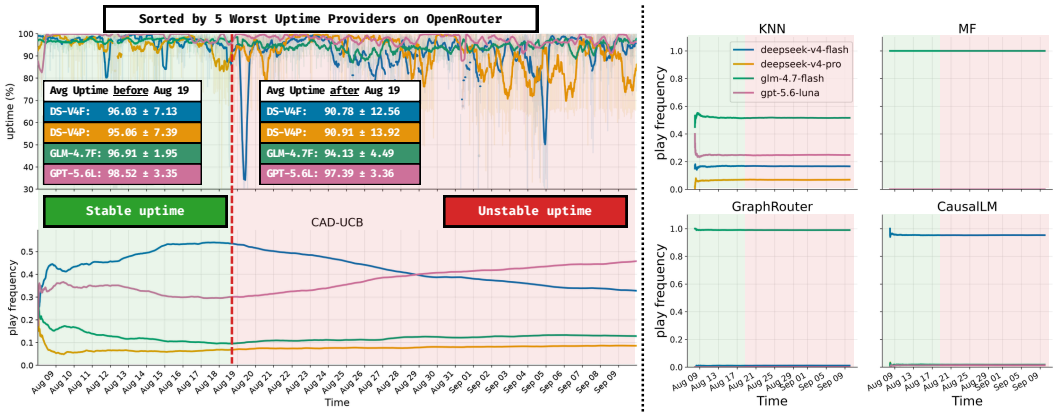
5.3 *METIS* UNDER NON-STATIONARY ENVIRONMENTS

State-of-the-art models are typically accessed through cloud APIs, so we run a series of experiments that probe the non-stationarity faced by *METIS*, and by CAD-UCB in particular. All four models in our study are queried through OpenRouter (OpenRouter, 2026) and differ not only in cost, but also in inference uptime, number of providers, and overall stability. We believe that all these factors play a crucial part when designing a routing algorithm that takes into account the performance-to-cost ratio together with non-stationary nature of the task.

Synthetically crafted disruptions. We first study the adaptability under a controlled synthetic disruption. We pre-compute every model's output on each problem in *METIS-DS*'s test set and replay these data to simulate live routing, giving us control over the factors that affect observed performance. In Figure 4, the routers operate without disruption for the first $\sim$250 timesteps; we then adjust the cost of DeepSeek-V4-Flash to be five times higher, and other models' budgets get noisier. Since static methods cannot adjust their strategy on-the-fly, they either exhibit lower accuracy (kNN and MF) or their cost increases significantly due to over-pulling the DeepSeek-V4-Flash arm. Our CAD-UCB balances the exploitation of different arms and is able to switch strategies as shown by the changing distribution of arm pulls.

Simulated OpenRouter experiment. A controlled synthetic disruption may not capture the full dynamics of API-based model access. We therefore replay historical OpenRouter uptime traces collected by Wu (2026). The protocol parallels the synthetic setting, except that each timestep is aligned with a historical ping recorded every five minutes. At each step, we sample a problem, let the router select a model, choose a provider from that model's list using a simple score given by the product of uptime and inference cost, and apply the provider's uptime and cost to the query. For example, if the selected provider has 80% uptime, the query fails with probability 20%, but the attempt is still billed. Because OpenRouter's implementation and internal routing remain undisclosed, this experiment cannot fully reproduce production non-stationarity; nevertheless, it provides a reproducible approximation based on public traces. To stress-test *METIS*, we restrict attention to the five providers with the lowest uptime. Results for the five most reliable providers and the full provider pool are reported in the Appendix.

Figure 5 illustrates how model serving variations impact theorem proving performance. When uptime for the DeepSeek model family declined after August 19, 2026 (dropping by 5-6 with double the standard deviation), CAD-UCB adapted by decreasing DeepSeek arm pulls in favor of the more stable GPT-5.6-Luna. Static methods, which rely on pretrained routing, fail to adjust to these subtle, day-to-day changes in model access. Because constantly retraining routers for an ever-changing



Routing	KNN	MF	Graph	Causal LM	CAD-UCB
Accuracy (%)	65.25 ± 0.15	61.61 ± 0.26	61.21 ± 0.25	70.78 ± 0.48	**69.96 ± 0.82**
Cost (USD)	31.71 ± 0.77	15.13 ± 0.30	15.27 ± 0.31	29.69 ± 0.38	**20.94 ± 2.31**

Figure 5: OpenRouter uptime simulation for five routing methods: kNN, MF, Graph, CausalLM and CAD-UCB. (Upper left) Model availability across the *five least reliable* OpenRouter providers, sampled at 5-minute intervals mapped against the arm selection frequency of CAD-UCB. (Upper right) Arm selection frequency of the static methods. Bottom table reports mean and standard deviation over 20 repetitions for accuracy and associated costs.

environment is impractical, CAD-UCB's dynamic adaptation proves effective, achieving $\sim$70% accuracy at a lower relative budget.

Table 4: Mean accuracy (%) and cost (USD) of CAD-UCB on *METIS-DS*, averaged across 20 runs.

$\xi \setminus \gamma$	Accuracy (%)					Cost (USD)				
	0.991	0.993	0.995	0.997	0.999	0.991	0.993	0.995	0.997	0.999
0.01	72.02	71.71	72.12	72.66	**72.74**	2.14	2.30	2.23	2.19	2.28
0.05	70.98	71.43	71.57	72.03	71.98	2.29	2.40	2.31	2.29	2.30
0.10	70.69	71.07	70.91	71.10	71.81	2.18	2.21	2.22	2.25	2.30
0.15	70.69	71.17	70.31	70.91	71.22	2.14	2.18	2.13	2.19	2.34
0.20	70.76	70.43	70.29	70.66	70.97	2.08	2.21	2.21	2.05	2.03
0.25	70.28	69.79	70.86	71.64	71.33	**1.98**	2.14	2.16	2.28	2.19

5.4 Ablation of *METIS* parameters

In Table 4 we ablate two essential hyperparameters of CAD-UCB: ξ, responsible for exploration, and γ, responsible for the discount factor, or the effective window size of past observations. We recorded accuracy and cost over 20 independent runs on *METIS-DS*. The lowest exploration factor combined with the largest effective window size yields the best accuracy, while the reverse yields the best cost efficiency. Therefore, when tuning the hyperparameters for CAD-UCB in *METIS*, the evaluator should balance the two. For our experiments, we prioritize accuracy. Full tables with standard deviation are reported in the Appendix K.

6 Conclusion

In this work, we introduced *METIS*, a decomposition-based online learning approach for efficient formal math inference. The framework uses LLM-based drafting together with the Lean compiler to conduct structure checks and decompose a complex proof into smaller sub-problems; then, we invoke a bandit algorithm to route these sub-problems to cheaper, smaller models. Experimental results show that we can reduce the budget expenditure by almost 70% on average while maintaining high accuracy, even on challenging Putnam and IMO competitions. Moreover, we introduced *METIS-DS*, a synthetically generated dataset for routing evaluation specifically designed for Lean. Overall, *METIS* presents a mixed inference approach that takes advantage of Lean's proof modularity to delegate easier tasks to smaller sub-agents for more efficient workload distribution.

AI USE STATEMENT

In this work, we used generative AI tools for generating synthetic data sets. We have not used generative AI tools for designing scientific figures, developing theoretical models or conceptual frameworks, formulating mathematical claims, providing critical ingredients for proving mathematical claims, assisting in the writing of proofs, proposing or refining hypotheses, designing or providing feedback on research methodology or experiments, interpreting results; and the rest of the required disclosure tasks are not applicable to this work.

Additionally, we used generative AI tools for creating or editing software code and editing a research paper to improve readability. We have reviewed all AI-assisted work. We used coding agents to help debugging our implementation and writing tools to catch grammar errors and improve readability of the work. 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 work focuses on optimizing inference costs for automated theorem proving in Lean. The research does not involve human subjects, and the proposed *METIS-DS* dataset contains no personally identifiable information, offensive material, or sensitive data. We do not foresee any direct malicious risks associated with our routing framework or the released dataset.

REPRODUCIBILITY STATEMENT

We are committed to ensuring the reproducibility of the *METIS* framework and the associated experimental results. To facilitate this, we have taken the following steps: (1) We will release *METIS-DS*, comprising 290 sub-lemmas and complete model trajectories ($\sim$13.4K lines of Lean code), to allow the community to study routing behavior and reproduce our warm-up phases; (2) All critical hyperparameters for the CAD-UCB algorithm (cost weight λ, discount γ, exploration coefficient ξ) are explicitly stated in Section 5.1. Extensive ablations are provided in Tables 11 and 12, and exact API token pricing for all models and OpenRouter providers is documented in the Appendix D; (3) The exact system prompts, instructions, and schemas used for the base drafting model, sub-models, and Lean compiler interactions are provided verbatim in Appendix L. All experiments specify the exact Lean version (v4.33.0) and compiler backend (REPL via Kimina-Lean) utilized; (4)The complete source code for the *METIS* framework, including the CAD-UCB routing logic, will be made publicly available under an MIT license upon acceptance.

REFERENCES

Tudor Achim, Alex Best, Alberto Bietti, Kevin Der, Mathïs Fédérico, Sergei Gukov, Daniel Halpern-Leistner, Kirsten Henningsgard, Yury Kudryashov, Alexander Meiburg, et al. Aristotle: Imo-level automated theorem proving. *arXiv preprint arXiv:2510.01346*, 2025.

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 *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=e6WrwIvgzX.

Anthropic. Claude code: Agentic coding tool, 2025. Software tool.

Anthropic. Formalizing Fermat's last theorem. https://www.anthropic.com/research/formalizing-fermats-last-theorem, Sep 2026a.

Anthropic. Introducing claude opus 4.8. https://www.anthropic.com/news/claude-opus-4-8/, May 2026b.

Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2):235–256, 2002.

Chenrui Cao, Liangcheng Song, Zenan Li, Xinyi Le, Xian Zhang, Hui Xue, and Fan Yang. Reviving dsp for advanced theorem proving in the era of reasoning models. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

Lingjiao Chen, Matei Zaharia, and James Zou. Frugalgpt: How to use large language models while reducing cost and improving performance. *arXiv preprint arXiv:2305.05176*, 2023.

Luoxin Chen, Jinming Gu, Liankai Huang, Wenhao Huang, Zhicheng Jiang, Allan Jie, Xiaoran Jin, Xing Jin, Chenggang Li, Kaijing Ma, et al. Seed-prover: Deep and broad reasoning for automated theorem proving. *arXiv preprint arXiv:2507.23726*, 2025.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pp. 337–340. Springer, 2008.

Ziming Dong, Hardik Sharma, Evan O'Toole, Jaya Prakash Champati, and Kui Wu. Pay for hints, not answers: Llm shepherding for cost-efficient inference. *arXiv preprint arXiv:2601.22132*, 2026.

Tao Feng, Yanzhen Shen, and Jiaxuan You. Graphrouter: A graph-based router for llm selections. In *International Conference on Learning Representations*, volume 2025, pp. 26186–26203, 2025.

Tao Feng, Fangxu Yu, Haozhen Zhang, Zhongjie Dai, Liangqi Yuan, Zijie Lei, Weizhi Zhang, Kunlun Zhu, Haodong Yue, Keyang Xuan, et al. Llmrouter: Unified infrastructure for developing, evaluating, and deploying llm routers. *arXiv preprint arXiv:2608.06867*, 2026.

Guoxiong Gao, Bin Wu, Zeming Sun, Jiedong Jiang, Wanyi He, Zichen Wang, Yutong Wang, Peihao Wu, and Bin Dong. Archon: An agentic system for formalizing research-level mathematics, March 2026. URL https://frenzymath.com/blog/archon-firstproof/.

Aurélien Garivier and Eric Moulines. On upper-confidence bound policies for switching bandit problems. In *International conference on algorithmic learning theory*, pp. 174–188. Springer, 2011.

Timnit Gebru, Jamie Morgenstern, Briana Vecchione, Jennifer Wortman Vaughan, Hanna Wallach, Hal Daumé Iii, and Kate Crawford. Datasheets for datasets. *Communications of the ACM*, 64(12): 86–92, 2021.

Xiaoyan Hu, Ho fung Leung, and Farzan Farnia. An optimism-based approach to online evaluation of generative models. In *The 28th International Conference on Artificial Intelligence and Statistics*, 2025. URL https://openreview.net/forum?id=Cg1QUTpTIp.

Yichen Huang and Lin F Yang. Winning gold at imo 2025 with a model-agnostic verification-and-refinement pipeline. *arXiv preprint arXiv:2507.15855*, 2025.

Albert Qiaochu Jiang, Sean Welleck, Jin Peng Zhou, Wenda Li, Jiacheng Liu, Mateja Jamnik, Timoth'ee Lacroix, Yuhuai Wu, and Guillaume Lample. Draft, Sketch, and Prove: Guiding formal theorem provers with informal proofs. In *International Conference on Learning Representations*, 2023. URL https://doi.org/10.48550/arXiv.2210.12283.

Shashank Kirtania, Priyanshu Gupta, and Arjun Radhakrishna. Logic-lm++: Multi-step refinement for symbolic formulations. In *Proceedings of the 2nd Workshop on Natural Language Reasoning and Structured Explanations (@ ACL 2024)*, pp. 56–63, 2024.

Levente Kocsis and Csaba Szepesvári. Discounted ucb. In *2nd PASCAL Challenges Workshop*, volume 2, pp. 51–134, 2006.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35: 22199–22213, 2022.

Po-Nien Kung, Linfeng Song, Dawsen Hwang, Jinsung Yoon, Chun-Liang Li, Simone Severini, Mirek Olšák, Edward Lockhart, Quoc V Le, Burak Gokturk, et al. Leap: Supercharging llms for formal mathematics with agentic frameworks. *arXiv preprint arXiv:2606.03303*, 2026.

Tor Lattimore and Csaba Szepesvári. *Bandit algorithms*. Cambridge University Press, 2020.

Lean FRO. A read-eval-print-loop for Lean 4. https://github.com/leanprover-community/repl, 2023.

Yang Li. Rethinking predictive modeling for llm routing: When simple knn beats complex learned routers. *arXiv preprint arXiv:2505.12601*, 2025.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations*, volume 2024, pp. 39578–39601, 2024.

Haohan Lin, Zhiqing Sun, Sean Welleck, and Yiming Yang. Lean-star: Learning to interleave thinking and proving. In *International Conference on Learning Representations*, volume 2025, pp. 66041–66062, 2025.

Yong Lin, Shange Tang, Bohan Lyu, Ziran Yang, Jui-Hui Chung, Haoyu Zhao, Lai Jiang, Yihan Geng, Jiawei Ge, Jingruo Sun, et al. Goedel-prover-v2: Scaling formal theorem proving with scaffolded data synthesis and self-correction. In *International Conference on Learning Representations*, volume 2026, pp. 11793–11818, 2026.

Chengwu Liu, Ye Yuan, Yichun Yin, Yan Xu, Xin Xu, Zaoyu Chen, Yasheng Wang, Lifeng Shang, Qun Liu, and Ming Zhang. Safe: Enhancing mathematical reasoning in large language models via retrospective step-aware formal verification. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12171–12186, 2025.

Junqi Liu, Zihao Zhou, Zekai Zhu, Marco Dos Santos, Weikun He, jiawei liu, 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. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=0bTEd4LpQr.

Leonardo de Moura and Sebastian Ullrich. The lean 4 theorem prover and programming language. In *International Conference on Automated Deduction*, pp. 625–635. Springer, 2021.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M Waleed Kadous, and Ion Stoica. RouteLLM: Learning to route LLMs from preference data. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=8sSqNntaMr.

OpenAI. Previewing GPT-5.6 Sol: a next-generation model. https://openai.com/index/previewing-gpt-5-6-sol/, Jun 2026. OpenAI Product Release.

OpenRouter. Openrouter: A unified AI inference layer. https://openrouter.ai/, 2026.

Azim Ospanov, Farzan Farnia, and Roozbeh Yousefzadeh. APOLLO: Automated LLM and lean collaboration for advanced formal reasoning. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025a. URL https://openreview.net/forum?id=fxDCgOruk0.

Azim Ospanov, Farzan Farnia, and Roozbeh Yousefzadeh. minif2f-lean revisited: Reviewing limitations and charting a path forward. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025b. URL https://openreview.net/forum?id=KtaHv0YUyh.

Azim Ospanov, Zijin Feng, Jiacheng Sun, Haoli Bai, SHEN XIN, and Farzan Farnia. HERMES: Towards efficient and verifiable mathematical reasoning in LLMs. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=w7BZcUc0fJ.

Liangming Pan, Alon Albalak, Xinyi Wang, and William Wang. Logic-lm: Empowering large language models with symbolic solvers for faithful logical reasoning. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 3806–3824, 2023.

Keiran Paster, Marco Dos Santos, Zhangir Azerbayev, and Jimmy Ba. Openwebmath: An open dataset of high-quality mathematical web text. In *International Conference on Learning Representations*, volume 2024, pp. 20357–20379, 2024.

ZZ Ren, Zhihong Shao, Junxiao Song, Huajian Xin, Haocheng Wang, Wanjia Zhao, Liyue Zhang, Zhe Fu, Qihao Zhu, Dejian Yang, et al. Deepseek-prover-v2: Advancing formal mathematical reasoning via reinforcement learning for subgoal decomposition. *arXiv preprint arXiv:2504.21801*, 2025.

Ziju Shen, Naohao Huang, Fanyi Yang, Yutong Wang, Guoxiong Gao, Tianyi Xu, Jiedong Jiang, Wanyi He, Pu Yang, Mengzhou Sun, et al. Real-prover: Retrieval augmented lean prover for mathematical reasoning. *arXiv preprint arXiv:2505.20613*, 2025.

Wenwen Si, Insup Lee, and Osbert Bastani. Policy-guided stepwise model routing for cost-effective reasoning. *arXiv preprint arXiv:2605.06116*, 2026.

Aleksandrs Slivkins. Introduction to multi-armed bandits. *Foundations and Trends® in Machine Learning*, 12(1-2):1–286, 2019.

Haiming Wang, Mert Unsal, Xiaohan Lin, Mantas Baksys, Junqi Liu, Marco Dos Santos, Flood Sung, Marina Vinyes, Zhenzhe Ying, Zekai Zhu, et al. Kimina-prover preview: Towards large formal reasoning models with reinforcement learning. *arXiv preprint arXiv:2504.11354*, 2025.

Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. Math-shepherd: Verify and reinforce llms step-by-step without human annotations. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9426–9439, 2024.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Wenbin Wu. Openrouter uptime: an independent availability and data-policy registry for openrouter models. `https://github.com/dthinkr/openrouter-uptime`, 2026.

Huajian Xin, Z.Z. Ren, Junxiao Song, Zhihong Shao, Wanjia Zhao, Haocheng Wang, Bo Liu, Liyue Zhang, Xuan Lu, Qiushi Du, Wenjun Gao, Haowei Zhang, Qihao Zhu, Dejian Yang, Zhibin Gou, Z.F. Wu, Fuli Luo, and Chong Ruan. Deepseek-prover-v1.5: Harnessing proof assistant feedback for reinforcement learning and monte-carlo tree search. In *The Thirteenth International Conference on Learning Representations*, 2025. URL `https://openreview.net/forum?id=I4YAIwrsXa`.

Anyi Xu, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chenchen Ling, et al. Deepseek-v4: Towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, 2026.

Xin Xu, Tong Xiao, Zitong Chao, Zhenya Huang, Can Yang, and Yang Wang. Can llms solve longer math word problems better? In *International Conference on Learning Representations*, volume 2025, pp. 29005–29036, 2025.

Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. Metamath: Bootstrap your own mathematical questions for large language models. In *International Conference on Learning Representations*, volume 2024, pp. 45040–45061, 2024.

Aohan Zeng, Xin Lv, Qinkai Zheng, Zhenyu Hou, Bin Chen, Chengxing Xie, Cunxiang Wang, Da Yin, Hao Zeng, Jiajie Zhang, et al. Glm-4.5: Agentic, reasoning, and coding (arc) foundation models. *arXiv preprint arXiv:2508.06471*, 2025.

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

A PROOF OF THE CAD-UCB REGRET BOUND

Proof of Proposition 1. By linearity of expectation and the reward definition in $R_t = s_t\big(1 - \lambda\widetilde{c}_t\big)$, where $\lambda \in [0,1]$,

$$\mu_t(i) = \mathbb{E}\Big[s_t(i)\big(1 - \lambda\widetilde{c}_t(i)\big)\Big] = p_t(i) - \lambda q_t(i). \tag{6}$$

Because $s_t(i) \in \{0,1\}$ and $\widetilde{c}_t(i) \in [0,1]$, we have $0 \le s_t(i)\widetilde{c}_t(i) \le s_t(i)$, hence $0 \le q_t(i) \le p_t(i) \le 1$. With $\lambda \in [0,1]$ this gives $(1-\lambda)p_t(i) \le \mu_t(i) \le p_t(i)$, so $\mu_t(i) \in [0,1]$. No independence between success and cost is needed for this step: $q_t(i)$ is their joint, success-weighted cost.

Define the quantities:

$$\Delta_t(i) = \mu_t^\star - \mu_t(i) \quad \text{and} \quad \widetilde{N}_T(i) = \sum_{t=1}^{T} \mathbf{1}\{I_t = i,\ \Delta_t(i) > 0\}.$$

The bounded reward implies $0 \le \Delta_t(i) \le 1$, and therefore

$$\mathbb{E}[\mathrm{Regret}(T)] = \sum_{i=1}^{m} \mathbb{E}\Big[\sum_{t=1}^{T} \Delta_t(i)\mathbf{1}\{I_t = i\}\Big] \le \sum_{i=1}^{m} \mathbb{E}[\widetilde{N}_T(i)]. \tag{7}$$

We bound $\mathbb{E}[\widetilde{N}_T(i)]$ with the switching-bandit analysis of D-UCB by Garivier & Moulines (2011, Theorem 1). Their setting matches ours: rewards lie in $[0,B]$ with $B = 1$; each reward $X_t(i)$ is independent of the history before round t, which is the condition required by their Hoeffding-type inequality for discounted self-normalized sums; and the index in equation 3 is their D-UCB index, with the statistics in equation 2 indexed so that round t uses only rounds $s < t$. Two details of their statement differ from ours, and neither affects the argument. First, they define breakpoints as changes of the reward distributions, but their proof uses only that the means are constant between breakpoints, together with a concentration inequality for independent, not necessarily identically distributed, bounded rewards; breakpoints of the mean vector therefore suffice. Second, they count rounds with $I_t = i \neq i_t^\star$ for a designated best arm $i_t^\star$ and state the bound in terms of the smallest gap over those rounds, which is zero if arm i is ever tied for best. Their proof applies verbatim to $\widetilde{N}_T(i)$, which counts only rounds with $\Delta_t(i) > 0$, taking $i_t^\star$ to be any optimal arm and replacing the smallest gap by $\Delta_{\min}$: once $N_t(\gamma, i) \ge 16\xi \log n_T(\gamma)/\Delta_{\min}^2$, the exploration bonus of arm i is at most $\Delta_{\min}/2 \le \Delta_t(i)/2$, which is the only property of the gap the proof uses.

For $\xi > 1/2$ and $\gamma \in (0,1)$, this gives for every arm i

$$\mathbb{E}[\widetilde{N}_T(i)] \le b(\gamma)\, T(1-\gamma) \log\Big(\frac{1}{1-\gamma}\Big) + d(\gamma)\, \frac{\Upsilon_T}{1-\gamma} \log\Big(\frac{1}{1-\gamma}\Big), \tag{8}$$

where, with $\eta = 4\sqrt{1 - 1/(2\xi)}$,

$$b(\gamma) = \frac{16\xi}{\gamma^{1/(1-\gamma)}\Delta_{\min}^2} \cdot \frac{\lceil T(1-\gamma)\rceil}{T(1-\gamma)} + \frac{2\Big\lceil \log\Big(\frac{1}{1-\gamma}\Big) \Big/ \log(1+\eta)\Big\rceil}{\log\Big(\frac{1}{1-\gamma}\Big)\Big(1 - \gamma^{1/(1-\gamma)}\Big)},$$

and $d(\gamma)$ is the second factor in their theorem, which depends only on γ, ξ, and m.

Because $1 \le \Upsilon_T \le T$, the choice in $\gamma = 1 - \frac{1}{4}\sqrt{\frac{\Upsilon_T}{T}}$ gives $1 - \gamma = \frac{1}{4}\sqrt{\Upsilon_T/T} \in (0, \frac{1}{4}]$, that is, $\gamma \in [\frac{3}{4}, 1)$. We check that b and d are bounded on this interval uniformly in T and Υ_T. The map $\gamma \mapsto \gamma^{1/(1-\gamma)}$ is increasing, so $\gamma^{1/(1-\gamma)} \in [(3/4)^4, e^{-1})$ is bounded away from 0 and 1. Moreover, $\log(1/(1-\gamma)) \ge \log 4$, and $T(1-\gamma) = \frac{1}{4}\sqrt{T\Upsilon_T} \ge \frac{1}{4}$ gives $\lceil T(1-\gamma)\rceil/(T(1-\gamma)) \le 5$. Hence $b(\gamma) \le \bar{b}$ for a constant $\bar{b}$ that depends only on ξ and $\Delta_{\min}$. The factor $d(\gamma)$ is continuous on $[\frac{3}{4}, 1)$ and tends to 1 as $\gamma \uparrow 1$, as noted by Garivier & Moulines (2011), so $|d(\gamma)| \le \bar{d}$ for a constant $\bar{d}$ that depends only on ξ and m.

Substituting $1-\gamma = \frac{1}{4}\sqrt{\Upsilon_T/T}$ gives

$$T(1-\gamma) = \frac{1}{4}\sqrt{T\Upsilon_T}, \qquad \frac{\Upsilon_T}{1-\gamma} = 4\sqrt{T\Upsilon_T},$$

and

$$\log\Big(\frac{1}{1-\gamma}\Big) = \log 4 + \frac{1}{2}\log\Big(\frac{T}{\Upsilon_T}\Big) \le \log 4 + \frac{1}{2}\log T \le \frac{5}{2}\log T,$$

where the last step uses $T \ge 2$, which holds because a breakpoint requires at least two rounds. Consequently,

$$\mathbb{E}[\widetilde{N}_T(i)] \le \frac{5}{2}\Big(\frac{\bar{b}}{4} + 4\bar{d}\Big)\sqrt{T\Upsilon_T}\log T.$$

Summing over the m arms in Equation 7 proves Equation 5 with $C = \frac{5m}{2}\big(\frac{\bar{b}}{4} + 4\bar{d}\big)$.

Finally, suppose $\Upsilon_T = 0$ and $\gamma = 1$. Then the statistics in Equation 2 are the ordinary counts $N_t(i)$ and empirical means, $n_t(1) = t-1$, and the exploration bonus is $2\sqrt{\xi\log(t-1)/N_t(i)}$. Since the theorem of Garivier & Moulines (2011) is stated for $\gamma < 1$, we use the standard UCB argument (Auer et al., 2002). Fix an arm i with $\Delta_i > 0$ and let $u = \lceil 16\xi\log T/\Delta_i^2\rceil$. If $N_t(i) \ge u$, the bonus of arm i is at most $\Delta_i/2$, so $I_t = i$ requires that either the empirical mean of arm i exceeds $\mu(i)$ by more than its bonus, or the empirical mean of an optimal arm falls below $\mu^\star$ by more than its bonus. In an oblivious environment the centered rewards observed from an arm form a bounded martingale difference sequence, so the Azuma–Hoeffding inequality and a union bound over the at most $t-1$ possible values of the count bound each of these probabilities by $(t-1)^{1-8\xi}$. Since $\xi > 1/2$, the series $\sum_{t\ge 2} 2(t-1)^{1-8\xi}$ converges, and

$$\mathbb{E}[N_T(i)] \le \frac{16\xi\log T}{\Delta_i^2} + O(1),$$

where the $O(1)$ term depends only on ξ. Multiplying by Δ_i and summing over suboptimal arms yields $O(\sum_{i:\Delta_i>0}(\log T)/\Delta_i)$, as claimed.

□

B DATASHEET

Following the framework of Gebru et al. (2021) and Paster et al. (2024), Table 5 documents the motivation, composition, collection, intended uses, distribution, and maintenance of *METIS-DS*. The dataset contains 290 sub-lemmas obtained by decomposing DeepSeek-ProverBench problems, together with candidate trajectories from the four routing-catalog models, Lean-compiler outcomes, and observed inference costs. These fields make each instance directly usable for studying both proof success and cost-aware routing.

Table 5: Datasheet for our dataset *METIS-DS*

Questions	Answers
Motivation	
For what purpose was the dataset created?	The dataset was developed to test the routing performance on Lean tasks, as well as to inspect the agent outputs when paired with smaller models.
Who created the dataset and on behalf of which entity?	The authors of this paper.
Who funded the creation of the dataset?	The company where the authors work.
Any other comment?	None.
Composition	
What do the instances that comprise the dataset represent?	Lemmas collected from DeepSeek ProverBench (Ren et al., 2025).
How many instances are there in total?	290 instances
Does the dataset contain all possible instances or is it a sample of instances from a larger set?	Yes, we present all instances.
What data does each instance consist of?	A problem statement, candidate solution drafted by four LLMs and associated trajectory data.
Is there a label or target associated with each instance?	Only compiler outputs and associated inference costs.
Is any information missing from individual instances?	No.
Are relationships between individual instances made explicit?	Not applicable.
Are there recommended data splits?	No recommended split.
Are there any errors, sources of noise, or redundancies in the dataset?	No, there are no errors in the proposed dataset to the best of our knowledge. Verification was done by the Lean compiler.
Is the dataset self-contained, or does it link to or otherwise rely on external resources?	The dataset is derived from DeepSeek ProverBench, but all instances are unique lemmas derived from the problem statements.
Does the dataset contain data that might be considered confidential?	No.
Does the dataset contain data that, if viewed directly, might be offensive, insulting, threatening, or might otherwise cause anxiety?	No.
Collection Process	

Continued on next page

How was the data associated with each instance acquired?	All 290 instances originate from lemma decomposition process based on DeepSeek Prover-Bench.
What mechanisms or procedures were used to collect the data?	The seed problems were taken from DeepSeek Prover-Bench repository. Then, we used our method to decompose them into sub-problems to prove.
If the dataset is a sample from a larger set, what was the sampling strategy?	No, there is no larger version.
Who was involved in the data collection process and how were they compensated?	Authors of this work are responsible for data collection.
Over what time frame was the data collected?	The dataset was collected over a four-month period, from June 2026 through September 2026.
Were any ethical review processes conducted?	No.
Preprocessing/cleaning/labeling	
Was any preprocessing/cleaning/labeling of the data done?	Data labelling was done by Lean compiler
Was the "raw" data saved in addition to the preprocessed/cleaned/labeled data?	No, all raw data was converted to preprocessed samples.
Is the software that was used to preprocess/clean/label the data available?	Only Lean compiler provided by Wang et al. (2025).
Any other comments?	No.
Uses	
Has the dataset been used for any tasks already?	Yes, we used to draw usage statistic on routing tasks and ablate hyperparameters of CAD-UCB, part of our *METIS* .
Is there a repository that links to any or all papers or systems that use the dataset?	Public GitHub and HuggingFace links will be provided at a later date.
What (other) tasks could the dataset be used for?	The dataset may be used for benchmarking routing tasks and studying model trajectories on Lean tasks.
Is there anything about the composition of the dataset or the way it was collected and preprocessed/cleaned/labeled that might impact future uses?	We do not anticipate future issues emerging from the proposed benchmark dataset.
Are there tasks for which the dataset should not be used?	The data is primarily for benchmarking, so we discourage others from training their models on this dataset.
Any other comments?	No.
Distribution	
Will the dataset be distributed to third parties outside of the entity on behalf of which the dataset was created?	Yes, we will release the dataset on public platforms such as GitHub and HuggingFace at a later date.
How will the dataset will be distributed?	GitHub, HuggingFace.
When will the dataset be distributed?	The dataset will be distributed along with the camera-ready version of the submission.
Will the dataset be distributed under a copyright or other intellectual property license, and/or under applicable terms of use?	Yes, the dataset will be released under the MIT license.

Continued on next page

Have any third parties imposed IP-based or other restrictions on the data associated with the instances?	No.
Do any export controls or other regulatory restrictions apply to the dataset or to individual instances?	No.
Any other comments?	No.
Maintenance	
Who will be supporting/hosting/maintaining the dataset?	The first author of the submission.
How can the owner/curator/manager of the dataset be contacted?	The curator of the dataset may be reached through an email or any other public means, such as GitHub profile.
Is there an erratum?	At this stage there is no associated erratum.
Will the dataset be updated?	Yes, we plan to periodically update the dataset if mistakes are found.
If the dataset relates to people, are there applicable limits on the retention of the data associated with the instances?	Not applicable.
Will older versions of the dataset continue to be supported/hosted/maintained?	Yes.
If others want to extend/augment/build on/contribute to the dataset, is there a mechanism for them to do so?	Yes, since we release all accompanied information, others may use this data for other uses.
Any other comments?	No.

Table 6: Token prices of models used in the experiments (USD per 1M tokens). Prices are taken from the API configuration and include cached-input, uncached-input, and output rates. Claude Opus 4.8 is the base model; the remaining models form the routing catalog.

Model	Role	Input cache hit	Input cache miss	Output
Claude Opus 4.8	Base	0.95	9.50	47.50
DeepSeek-V4-Flash	Sub-model	0.014	0.44	1.32
DeepSeek-V4-Pro	Sub-model	0.044	1.32	3.96
GPT-5.6-Luna	Sub-model	0.02	0.20	1.20
GLM-4.7-Flash	Sub-model	0.01	0.06	0.40

C Additional implementation details

The Lean compiler oracle used to obtain the success bit s_t times out after 60 seconds; shell-tool commands time out after 300 seconds. Unless stated otherwise, the catalog $\mathcal{T}$ is LeanExplore, Loogle, REPL (Lean FRO, 2023), and Lean-MCP (Liu et al., 2026), with the REPL backend based on the Kimina-Lean server (Wang et al., 2025), plugged into the Claude Code (Anthropic, 2025) harness. CAD-UCB was run on the multiplicative reward with cost weight $\lambda = 0.5$, reference cost $c_{\mathrm{ref}} = 0.0577$ (the 95th percentile of warm-up attempt cost on *METIS-DS*), discount $\gamma = 0.99$, and exploration coefficient $\xi = 0.05$. All experiments use Lean `v4.33.0`.

D Model catalog and inference costs

Direct API prices. Table 6 gives the rates used to convert token counts into dollar costs. The price gap between the base model and the routing catalog is substantial: Claude Opus 4.8 costs \$47.50 per million output tokens, whereas the four sub-models range from \$0.40 to \$3.96. The corresponding uncached-input rates are \$9.50 for the base model and \$0.06–\$1.32 for the sub-models.

Provider-specific prices. Table 7 expands the catalog prices by OpenRouter provider. Rates can vary materially even for the same model: for DeepSeek-V4-Flash, for example, output prices range from \$0.16 to \$1.32 per million tokens. Thus, the realized cost of an arm depends not only on the selected model but also on the provider serving it. We use these provider-level rates in the historical-uptime replays described below.

Table 7: OpenRouter provider prices for each routing-catalog model (USD per 1M tokens). We used these pricings for our OpenRouter uptime simulation experiment.

Model	Provider	Input cache hit	Input cache miss	Output
DeepSeek-V4-Flash	Alibaba	0.0352	0.352	1.056
DeepSeek-V4-Flash	Ambient	0.028	0.44	1.32
DeepSeek-V4-Flash	AtlasCloud	0.028	0.44	1.32
DeepSeek-V4-Flash	Azure	0.044	0.44	1.32
DeepSeek-V4-Flash	Baidu	0.014	0.44	1.32
DeepSeek-V4-Flash	Cloudflare	0.014	0.44	1.32
DeepSeek-V4-Flash	CoreWeave	0.07	0.13	0.28
DeepSeek-V4-Flash	DeepInfra	0.015	0.06	0.18
DeepSeek-V4-Flash	DeepSeek	0.014	0.44	1.32
DeepSeek-V4-Flash	DigitalOcean	0.0252	0.08	0.252
DeepSeek-V4-Flash	Fireworks	0.007	0.22	0.66
DeepSeek-V4-Flash	GMICloud	0.0091	0.286	0.858

Continued on next page

Table 7: OpenRouter provider prices (continued)

Model	Provider	Input cache hit	Input cache miss	Output
DeepSeek-V4-Flash	Mancer 2	0.165	0.165	0.50
DeepSeek-V4-Flash	Morph	0.03125	0.1234	0.3475
DeepSeek-V4-Flash	NextBit	0.012	0.352	1.056
DeepSeek-V4-Flash	Novita	0.02604	0.4092	1.228
DeepSeek-V4-Flash	OpenInference	0.013	0.05	0.16
DeepSeek-V4-Flash	Parasail	0.05	0.14	0.28
DeepSeek-V4-Flash	Phala	0.028	0.44	1.32
DeepSeek-V4-Flash	Sail Research	0.02	0.065	0.18
DeepSeek-V4-Flash	SiliconFlow	0.028	0.22	0.66
DeepSeek-V4-Flash	StreamLake	0.00392	0.1232	0.3696
DeepSeek-V4-Flash	Venice	0.035	0.175	0.35
DeepSeek-V4-Flash	Wafer	0.05	0.10	0.25
DeepSeek-V4-Pro	Alibaba	0.05808	0.5808	1.742
DeepSeek-V4-Pro	AtlasCloud	0.132	1.32	3.96
DeepSeek-V4-Pro	Azure	0.132	1.32	3.96
DeepSeek-V4-Pro	Baidu	0.132	1.32	3.96
DeepSeek-V4-Pro	BaseTen	0.132	1.32	3.96
DeepSeek-V4-Pro	Cloudflare	0.44	1.32	3.96
DeepSeek-V4-Pro	CoreWeave	0.132	1.32	3.96
DeepSeek-V4-Pro	DeepInfra	0.10	1.30	2.60
DeepSeek-V4-Pro	DeepSeek	0.022	0.66	1.98
DeepSeek-V4-Pro	DigitalOcean	0.044	1.32	3.96
DeepSeek-V4-Pro	Fireworks	0.044	1.32	3.96
DeepSeek-V4-Pro	GMICloud	0.0374	1.122	3.366
DeepSeek-V4-Pro	Ionstream	0.132	1.32	3.96
DeepSeek-V4-Pro	NextBit	0.132	1.32	3.96
DeepSeek-V4-Pro	Novita	0.132	1.32	3.96
DeepSeek-V4-Pro	Parasail	0.044	1.32	3.96
DeepSeek-V4-Pro	SiliconFlow	0.44	1.32	3.96
DeepSeek-V4-Pro	StreamLake	0.022	0.66	1.98
DeepSeek-V4-Pro	Together	0.13	1.32	3.96
DeepSeek-V4-Pro	Venice	0.132	1.32	3.96
GPT-5.6-Luna	Amazon Bedrock	0.022	0.22	1.32
GPT-5.6-Luna	Azure	0.02	0.20	1.20
GPT-5.6-Luna	OpenAI	0.02	0.20	1.20
GLM-4.7-Flash	Cloudflare	0.0605	0.0605	0.40
GLM-4.7-Flash	DeepInfra	0.01	0.06	0.40
GLM-4.7-Flash	Novita	0.01	0.07	0.40
GLM-4.7-Flash	Venice	0.01	0.06	0.40

E ADDITIONAL RELATED WORKS

Neurosymbolic solvers and LLMs. Beyond theorem proving models, there have been numerous approaches to integrating neurosymbolic solvers with LLMs. LogicLM Pan et al. (2023) and LogicLM++ Kirtania et al. (2024) employ solvers like Z3 (De Moura & Bjørner, 2008) to construct symbolic formulations and iteratively refine them. Lean-STaR Lin et al. (2025) improves formal reasoning by injecting short, informal proof tactic explanations into Lean tactic fields during training.

Additionally, DTV explores the use of formal languages to reduce hallucinations and improve the self-consistency of mathematical claims. Another line of research investigates Outcome Reward Models (ORMs), which assign a score only to the final answer, and Process Reward Models (PRMs), which assign scores to each mathematical step of the proof Lightman et al. (2024); Wang et al. (2024). Notably, Safe Liu et al. (2025) combines reward modeling with Lean during training to achieve enhanced theorem proving capabilities.

Online learning and Bandit algorithms. Multi-armed bandits make decisions under partial feedback: at each round, a policy selects one arm, observes only that arm's reward (in our case it is associated cost and Lean compiler feedback), and must balance exploiting models that have performed well with exploring alternatives whose quality is uncertain (Slivkins, 2019; Lattimore & Szepesvári, 2020). A standard baseline is ϵ-greedy, which selects the empirically best arm with probability $1 - \epsilon$ and explores randomly otherwise. Its simplicity provides persistent exploration, but a fixed ϵ continues spending a constant fraction of decisions on sub-optimal arms. Upper Confidence Bound (UCB) policies instead implement optimism under uncertainty, selecting an arm by combining its empirical reward with a confidence bonus that is largest for under-sampled arms; in stationary stochastic environments, this results in logarithmic regret under standard assumptions (Auer et al., 2002). In this work we use bandit algorithm for an online selection (Hu et al., 2025) of samples to balance budget expenditure with theorem proving accuracy.

Classical ϵ-greedy and UCB estimators aggregate the full observation history and can therefore adapt slowly when reward distributions change. Non-stationary bandits address drifting or abruptly switching rewards using mechanisms such as restarts, sliding windows, or temporal discounting (Slivkins, 2019; Lattimore & Szepesvári, 2020). Discounted UCB geometrically downweights old observations via exponential means, while sliding-window UCB retains only recent feedback; both have been analyzed for switching environments (Kocsis & Szepesvári, 2006; Garivier & Moulines, 2011). This setting is especially relevant to online LLM routing, where the best model can change with proof difficulty, token prices, provider availability, or the changing sub-goals. Our work adopts this online learning via bandit perspective at the level of formal theorem-proving and uses both compiler output and inference cost in computing the reward.

Table 8: Performance of various routing methods on the DeepSeek Prover-Bench dataset. The model distribution shows the fraction of requests routed to smaller models, and savings are reported relative to the "No Routing" baseline. The best results are **bolded**.

Inference Method	Accuracy (%)	Cost (USD)	Savings (Δ%)	Model Distribution (%)			
				DS-V4F	DS-V4P	GLM-4.7F	GPT-5.6L
Rule-based baselines							
No Routing	**88.9**	$37.59	\	\	\	\	\
Cheapest LLM	**88.9**	$15.99	57.5	0.0	0.0	100.0	0.0
Decomposed sub-problem-level routing baselines							
Random Router	**88.9**	$14.68	60.9	29.4	23.5	17.6	29.4
*k*NN Router	**88.9**	$13.39	64.4	20.0	0.0	70.0	10.0
MF Router	**88.9**	**$13.11**	**65.1**	0.0	0.0	100.0	0.0
Graph Router	**88.9**	$19.44	48.3	0.0	0.0	100.0	0.0
CausalLM	**88.9**	$13.76	63.4	100.0	0.0	0.0	0.0
Proposed method							
METIS	**88.9**	$13.41	64.3	92.9	7.1	0.0	0.0

F ADDITIONAL BENCHMARK RESULTS

DeepSeek-ProverBench. Table 8 extends the end-to-end comparison to the DeepSeek-ProverBench evaluation subset. All methods solve 88.9% of the problems, so this setting isolates differences in expenditure rather than proof accuracy. *METIS* spends $13.41, a 64.3% reduction relative to no routing and only $0.30 more than the lowest-cost MF Router, while sending 92.9% of routed requests to DeepSeek-V4-Flash.

FATE-M. We note that the cheapest LLM, the MF router, and GraphRouter all use GLM-4.7-Flash, while achieving varied accuracy and, most importantly, varied cost. Upon inspection of the trajectories, we recorded that a weak model actively tries to write Lean code even if it cannot close the posed goal, so when the whole proof is reassembled, the base model tries to "correct" the mistakes, which contributes to fluctuating associated costs.

miniF2F-v2s (IMO subset). Similar to FATE-M, we also tested the routers and base model on the more challenging miniF2F-v2s IMO subset presented in Table 10. However, due to the higher difficulty of the problems, the models required deeper repair loops with access to the Lean compiler. More importantly, our uniform $20 per problem budget was not enough for Opus 4.8, which would hit the budget limit very quickly and achieve a very low accuracy of 45%. To mitigate this, we increased the budget to $50, following the Liu et al. (2026) guidelines for Putnam problems, while leaving other inference methods capped at $20. This helped us improve the accuracy, but many problems would still hit the cost wall. Instead of further raising the budget, we decided to test the benefits of routing in this scenario: what if the biggest LLM is simply too expensive? We observe that smaller models achieve a high level of performance, and CausalLM with *METIS* even outperforms the baseline. This experiment showcases that routing not only preserves the budget, but can even result in better accuracy under specific inference conditions.

Table 9: Performance of various routing methods on the FATE-M dataset. The model distribution shows the fraction of requests routed to smaller models, and savings are reported relative to the "No Routing" baseline.

Inference Method	Accuracy (%)	Cost (USD)	Savings (Δ%)	Model Distribution (%)			
				DS-V4F	DS-V4P	GLM-4.7F	GPT-5.6L
Rule-based baselines							
No Routing	93.3	$250.56	\	\	\	\	\
Cheapest LLM	61.3	$183.35	26.8	0.0	0.0	100.0	0.0
Baseline routers on the query level							
*k*NN Router	60.0	$30.98	87.6	13.9	3.7	72.7	9.7
MF Router	43.3	$58.87	76.5	0.0	0.0	100.0	0.0
Graph Router	63.3	$8.68	96.5	92.3	7.7	0.0	0.0
CausalLM	78.0	$7.97	96.8	89.7	5.7	0.0	4.6
CAD-UCB (ours)	88.7	$7.49	97.0	61.1	0.0	0.0	38.9
Decomposed sub-problem-level routing baselines							
Random Router	86.0	$104.50	58.3	22.0	22.9	31.3	23.8
*k*NN Router	82.0	$139.69	44.3	19.9	0.6	50.0	29.5
MF Router	63.3	$183.85	26.6	0.0	0.0	100.0	0.0
Graph Router	60.7	$207.37	17.2	0.0	0.0	100.0	0.0
CausalLM	88.7	$147.93	41.0	96.6	3.0	0.0	0.4
Proposed method							
METIS	89.3	$98.24	60.8	88.8	3.6	4.8	2.8

Table 10: Performance of various routing methods on the miniF2F-v2s (IMO subset) dataset. The model distribution shows the fraction of requests routed to smaller models, and savings are reported relative to the "No Routing" baseline. We report two "No Routing" settings with two different per problem budget limits: $20 and $50.

Inference Method	Accuracy (%)	Cost (USD)	Savings (Δ%)	Model Distribution (%)			
				DS-V4F	DS-V4P	GLM-4.7F	GPT-5.6L
Rule-based baselines							
No Routing ($50 per problem)	75.0	$369.41	\	\	\	\	\
No Routing ($20 per problem)	45.0	$243.08	\	\	\	\	\
Cheapest LLM	75.0	$132.90	64.0	0.0	0.0	100.0	0.0
Baseline routers on the query level							
*k*NN Router	35.0	$47.32	87.2	6.4	12.8	69.9	10.9
MF Router	15.0	$31.92	91.4	0.0	0.0	100.0	0.0
Graph Router	15.0	$35.08	90.5	0.0	0.0	100.0	0.0
CausalLM	85.0	$4.73	98.7	100.0	0.0	0.0	0.0
CAD-UCB (ours)	85.0	$3.96	98.9	95.7	4.3	0.0	0.0
Baseline routers on decomposed METIS drafts							
Random Router	70.0	$153.09	58.6	23.4	20.5	27.1	29.0
*k*NN Router	80.0	$164.23	55.5	7.8	7.8	48.0	36.4
MF Router	70.0	$131.01	64.5	0.0	0.0	100.0	0.0
Graph Router	70.0	$158.13	57.2	0.0	0.0	100.0	0.0
CausalLM	85.0	$82.46	77.7	96.5	3.5	0.0	0.0
Proposed method							
METIS	85.0	$75.18	79.6	90.1	3.7	2.5	3.7

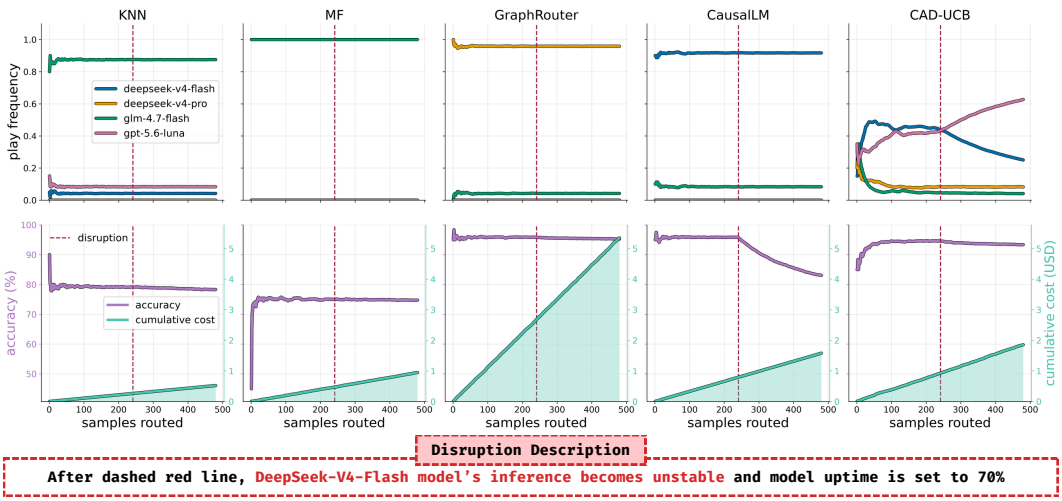


Figure 6: Dynamic-catalog replay for kNN, MF, Graph, CausalLM, and CAD-UCB routing. At the dashed red line, the inference uptime of DeepSeek-V4-Flash is set to 70%. The upper plots show arm choice distribution and the lower plot shows changes in accuracy and cumulative cost.

G Comparing non-stationary behavior against static algorithms

The experiments in this section compare CAD-UCB with the static routers from the main evaluation: kNN, MF, Graph, and CausalLM. Each static router scores a lemma once from its trained parameters and does not update after deployment. We present two more synthetic experiments alongside OpenRouter experiment with the most consistent and across all providers.

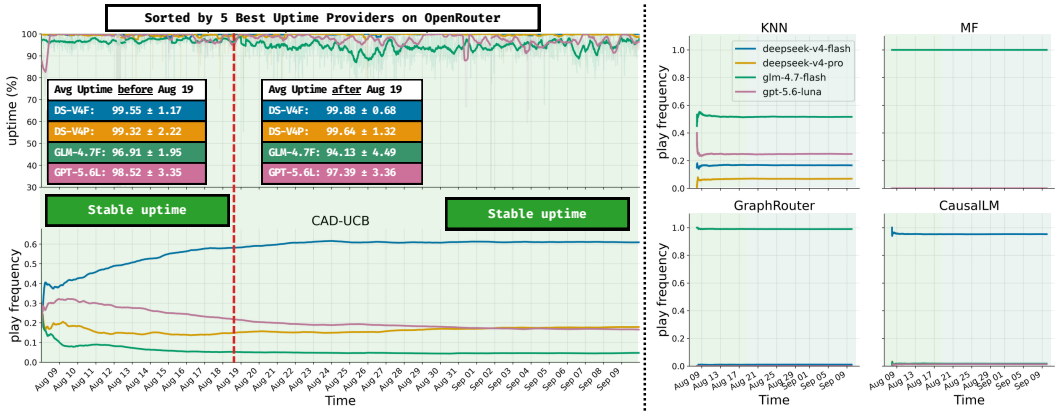
Uptime disruption experiment

Figure 6 holds the catalog fixed and, at the dashed line, lowers the uptime of DeepSeek-V4-Flash to 70%. The static allocations do not move. CausalLM sends about 92% of requests to the degraded arm, so its accuracy falls from 95.83% at the change point to 83.06% by the end, at a mean cost of \$1.58. kNN and MF stay on GLM-4.7-Flash and remain cheap (\$0.52 and \$0.95) but finish at 78.30% and 74.74% accuracy. Graph avoids the disrupted arm by staying on DeepSeek-V4-Pro, which preserves 95.23% accuracy at a much higher cost of \$5.35. CAD-UCB reduces its DeepSeek-V4-Flash share after the change and moves toward GPT-5.6-Luna, ending at 93.35% accuracy and \$1.85 mean cost. Discounted feedback therefore tracks the uptime drop without locking onto either the cheapest weak arm or the most expensive reliable one. This adaptability leads to a high accuracy at a much lower relative budget expenditure.

Additional OpenRouter uptime experiments

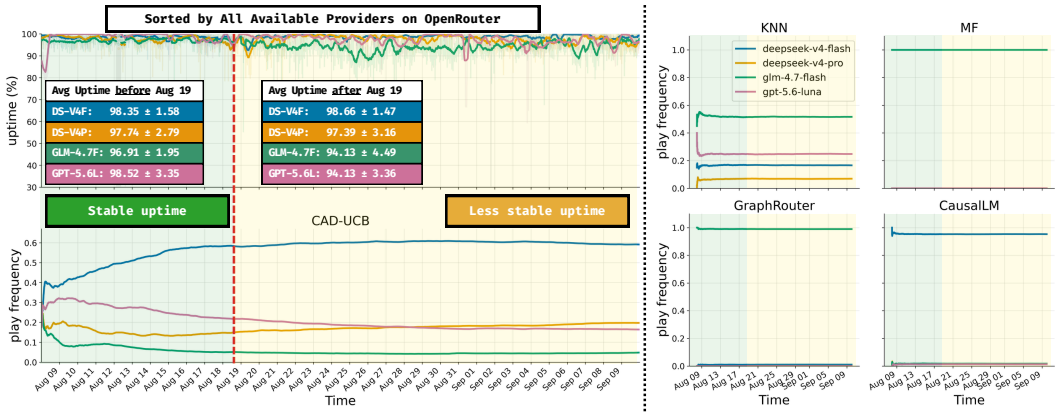
Figure 7 replays the five most reliable OpenRouter providers. Mean uptime stays high on both sides of August 19, and the figure marks both regimes as stable. The static policies are flat across that date: MF assigns every request to GLM-4.7-Flash, Graph does nearly the same, CausalLM stays on DeepSeek-V4-Flash, and kNN keeps a fixed mix led by GLM-4.7-Flash. Under this mild availability trace, kNN, MF and Graph routers remain below 70% accuracy, and CAD-UCB achieves a comparable accuracy to CausalLM while consuming $\sim$ 18% less budget.

Figure 8 repeats the replay over every recorded provider. Uptime is high before August 19 and less stable afterward, but the static arm shares are unchanged from the top-five setting, because those routers do not condition on provider availability. CAD-UCB finishes at 71.87% and \$21.68, slightly below its top-five result, and its allocation still shifts toward DeepSeek-V4-Flash after the date marker. Similar to the Figure 7, CAD-UCB delivers a comparable performance at a smaller cost.



Routing	KNN	MF	Graph	Causal LM	CAD-UCB
Accuracy (%)	66.29 ± 0.07	61.61 ± 0.26	61.25 ± 0.26	74.99 ± 0.10	**72.06 ± 0.63**
Cost (USD)	24.87 ± 0.44	15.13 ± 0.30	15.25 ± 0.30	26.05 ± 0.21	**21.44 ± 1.89**

Figure 7: OpenRouter uptime simulation for five routing methods: kNN, MF, Graph, CausalLM and CAD-UCB. (Upper left) Model availability across the *five most reliable* OpenRouter providers, sampled at 5-minute intervals mapped against the arm selection frequency of CAD-UCB. (Upper right) Arm selection frequency of the static methods. Bottom table reports mean and standard deviation over 20 repetitions for accuracy and associated costs.



Routing	KNN	MF	Graph	Causal LM	CAD-UCB
Accuracy (%)	66.29 ± 0.07	61.61 ± 0.26	61.25 ± 0.26	74.99 ± 0.10	**71.87 ± 0.67**
Cost (USD)	24.97 ± 0.57	15.13 ± 0.30	15.25 ± 0.30	25.85 ± 0.26	**21.67 ± 1.96**

Figure 8: OpenRouter uptime simulation for five routing methods: kNN, MF, Graph, CausalLM and CAD-UCB. (Upper left) Model availability across *all* OpenRouter providers, sampled at 5-minute intervals mapped against the arm selection frequency of CAD-UCB. (Upper right) Arm selection frequency of the static methods. Bottom table reports mean and standard deviation over 20 repetitions for accuracy and associated costs.

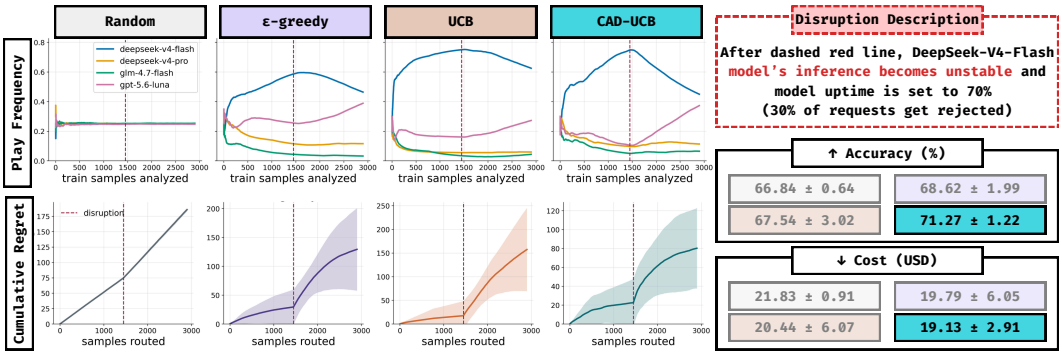


Figure 9: Relative model selection frequencies and cumulative regret for four online inference methods: Random routing, ϵ-greedy, UCB, and CAD-UCB. The upper right corner indicates the type of intervention applied during the inference procedure. Accuracy and color-coded associated costs, along with their standard deviations, are shown on the right.

H Comparing non-stationary behavior across different bandit algorithms

To isolate adaptation from model-generation noise, the two synthetic experiments below replay logged outcomes from *METIS-DS* for ten epochs, and apply one intervention halfway through each run. In Figures 10 and 11, the dashed red line marks this change point; the left panels show how allocation and cumulative regret evolve, while the right panels summarize end-to-end accuracy and cost. Both experiments use the multiplicative reward with $\lambda = 0.5$, $c_{\text{ref}} = 0.05$, $\xi = 0.01$, and $\epsilon = 0.01$ (only relevant to ϵ-greedy algorithm).

Uptime disruption experiment

Figure 9 lowers the uptime of DeepSeek-V4-Flash to 70% at the dashed line, so 30% of requests to that arm are rejected. Before the change, every learned policy concentrates on DeepSeek-V4-Flash. Afterward, CAD-UCB cuts that share from about 75% to 45% and moves the freed mass onto GPT-5.6-Luna, while UCB remains near 62% on the degraded arm and its cumulative regret continues to rise. CAD-UCB finishes at $71.27\% \pm 1.22\%$ accuracy and \$19.13 mean cost (standard deviation \$2.91), compared with $67.54\% \pm 3.02\%$ and \$20.44 for UCB and $68.62\% \pm 1.99\%$ and \$19.79 for ϵ-greedy. Discounting therefore recovers both accuracy and cost stability once the preferred arm becomes unreliable.

Cost-shift experiment

Figure 10 tests whether a router can respond when the initially preferred DeepSeek-V4-Flash arm becomes expensive. CAD-UCB discounts pre-intervention evidence and reallocates requests toward the new success–cost optimum, yielding a mean cost of \$26.01, compared with \$45.20 for ϵ-greedy and \$51.17 for UCB, while maintaining above 70% accuracy. The result illustrates the intended trade-off: UCB attains the highest raw accuracy, but its continued reliance on the formerly attractive arm leads to substantially higher and more variable expenditure.

Dynamic model availability experiment

Figure 11 evaluates adaptation to an expanding model catalog. CAD-UCB incorporates the newly available arms and obtains the highest mean accuracy, $72.43\% \pm 0.95\%$, at a moderate mean cost of \$20.84 (standard deviation \$6.57). Although ϵ-greedy is cheaper at \$17.65, its accuracy is 0.90 percentage points lower; UCB is both less accurate ($71.96\% \pm 1.44\%$) and more variable in cost (\$21.91, standard deviation \$10.65) than CAD-UCB. The allocation trace shows that discounting

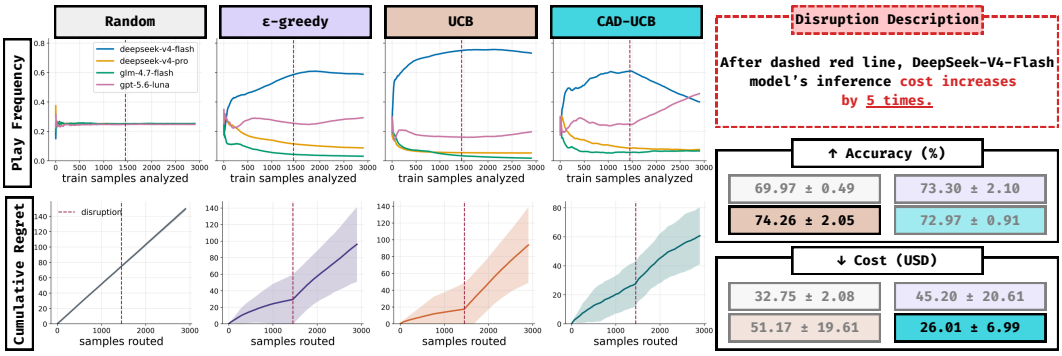


Figure 10: Cost-shift replay for Random routing, ϵ-greedy, UCB, and CAD-UCB. At the dashed red line, the inference cost of DeepSeek-V4-Flash increases fivefold. The left panels show relative model-selection frequencies and cumulative regret; the right panels report accuracy and cost (mean and standard deviation over 20 runs).

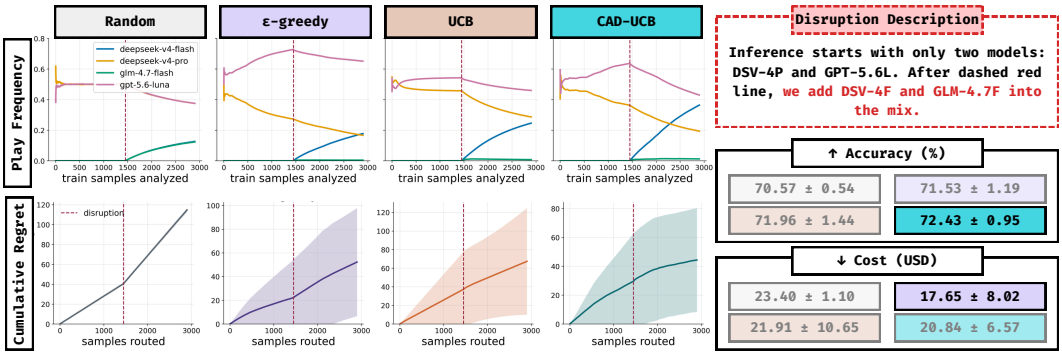


Figure 11: Dynamic-catalog replay for Random routing, ϵ-greedy, UCB, and CAD-UCB. Inference begins with DeepSeek-V4-Pro and GPT-5.6-Luna; at the dashed red line, DeepSeek-V4-Flash and GLM-4.7-Flash become available. The left panels show relative model-selection frequencies and cumulative regret; the right panels report accuracy and cost (mean and standard deviation over 42 runs).

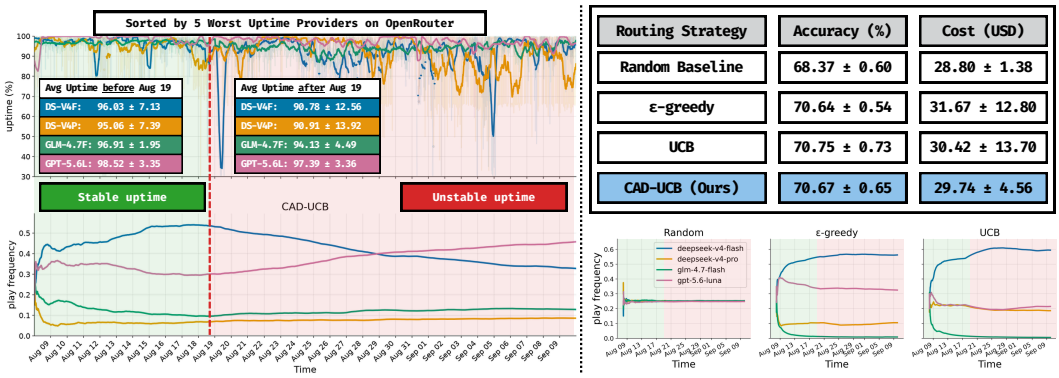


Figure 12: OpenRouter uptime simulation for four online dispatch policies: Random, ϵ-greedy, UCB, and CAD-UCB. (Upper left) Model availability across the *five least reliable* OpenRouter providers, sampled at 5-minute intervals. (Lower left) Arm selection dynamics for CAD-UCB over time. (Upper right) Overall accuracy and associated costs for each evaluated method. (Lower right) Arm selection behavior for the Random, ϵ-greedy, and UCB baselines. Statistic is collected over 20 runs.

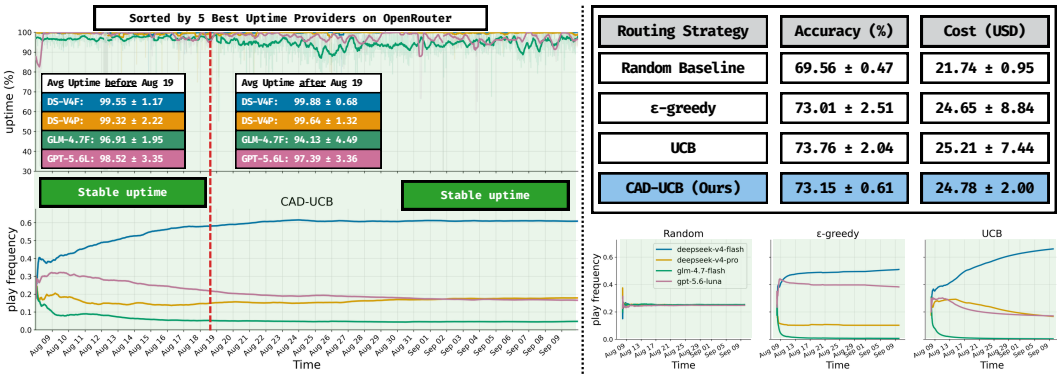


Figure 13: OpenRouter uptime replay for the five most reliable providers. The upper-left panel shows model availability at five-minute intervals; the lower-left panel shows CAD-UCB's arm-selection dynamics. The upper-right panel reports aggregate accuracy and cost, and the lower-right panel shows allocations for Random, ϵ-greedy, and UCB. Statistics are collected over 20 runs.

permits exploration of DeepSeek-V4-Flash after its arrival without discarding the useful evidence already collected for GPT-5.6-Luna.

In all of our experiments, we observe that CAD-UCB attains a smaller standard deviation compared to other tested bandits, which reflects a better performance and cost stability under non-stationary inference environments.

Additional OpenRouter uptime experiments

Figure 12 considers the five providers with the lowest uptime and shows the resulting availability for each weaker model. Around August 19, the average uptime of the DeepSeek models deteriorates considerably. All adaptive bandit methods achieve comparable accuracy; however, the standard deviation of CAD-UCB's cost is approximately half that of the other adaptive bandit methods. Its allocation trace shows reduced reliance on the less stable DeepSeek-V4-Flash and a shift toward the more stable GPT-5.6-Luna model. This result indicates that the proposed learning algorithm provides a more stable accuracy–cost trade-off under non-stationary API availability.

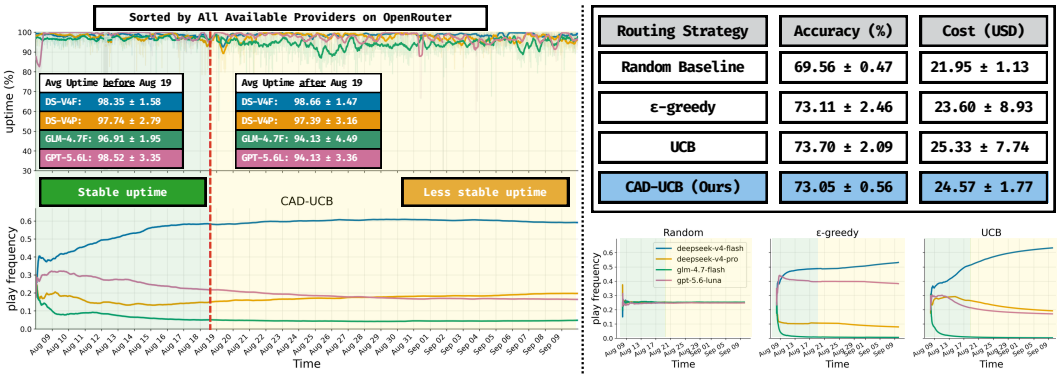


Figure 14: OpenRouter uptime replay over all available providers. The upper-left panel shows model availability at five-minute intervals; the lower-left panel shows CAD-UCB's arm-selection dynamics. The upper-right panel reports aggregate accuracy and cost, and the lower-right panel shows allocations for Random, ϵ-greedy, and UCB. Statistics are collected over 20 runs.

Figure 13 represents the comparatively stable provider regime: model-averaged mean uptime remains above 94% before and after August 19. All three learned routers exceed 73% mean accuracy, with UCB reaching 73.76% and CAD-UCB reaching 73.15%. CAD-UCB has a smaller accuracy and cost dispersion than UCB and ϵ-greedy, showing that the cost-aware discounted policy remains stable even when severe availability shifts are absent.

Figure 14 exposes each router to the full range of provider reliability. UCB achieves the highest mean accuracy at 73.70%, while CAD-UCB remains close at 73.05% and costs slightly less ($24.57 versus $25.33). More importantly, CAD-UCB's standard deviations are 0.56 percentage points for accuracy and $1.77 for cost, compared with 2.09 points and $7.74 for UCB. This result complements the worst-provider stress test in the main text: discounting primarily improves the stability of the accuracy–cost trade-off under heterogeneous availability. These findings motivate our use of an exponential mean instead of standard mean.

I CASE STUDIES: *METIS* VS. THE BASE OPUS-4.8 MODEL

To better understand the effect of *METIS*, we decomposed and studied two Putnam 2025 problems that were solved by *METIS* but not by the base model under a restricted budget. We present two case studies with specific details below.

Case study. Putnam 2025 B2. *METIS* vs. base only.

Outcome on the same three-lemma draft.

Base only: \$50.81, Emitted LIMIT, four `sorry` statements.
METIS : \$43.72, Emitted COMPLETE, zero `sorry` statements and zero compiler errors.
Cost gap: \$7.09 lower for the completed routed proof.

Base-only failure. Opus delegated the planar-centroid lemma twice and then its area subgoal `hvol`. All three Opus workers exhausted the 40-call limit, costing \$16.58, \$16.61, and \$12.86. Together they consumed \$46.05 (90.6% of the run) on one proof branch. The last attempt stalled while showing that the closed region and Mathlib's open `regionBetween` have equal measure:

```
sorry -- measure of graph is zero
```

The run ended before attempting the solid-centroid and ratio lemmas; its last check still had four errors.

***METIS* trajectory.** Its first whole-lemma attempt also hit the 40-call limit. Instead of retrying the same interface with Opus, the coordinator split the proof into seven local lemmas. Cheap workers closed six: two positivity lemmas, the correlation inequality, both planar slice lemmas, and the solid numerator. For the remaining solid-volume lemma, capped workers left a nearly complete disk-slice proof; Opus repaired only the concrete Lean errors (measurability, norm, and `ENNReal` coercions). All routed worker attempts together cost \$1.18.

What this case shows. The base model did not fail for lack of a proof plan; it spent almost its entire run cost repeatedly searching inside one monolithic measure-theory goal. *METIS* succeeded because decomposition turned progress into checked local lemmas and made repeated compiler-guided search cheap enough to reach the remaining branches.

This case isolates a failure mode in which proof-search granularity matters more than nominal model strength. The base-only run devoted 120 expensive worker calls to variants of the same obligation, so reaching a local tool-call limit also stranded most of the accumulated work in an unchecked proof state. *METIS* used more wall-clock time and API calls, but the low marginal cost of its workers allowed the coordinator to explore all proof branches and preserve each successful lemma as a compiler-checked checkpoint. Even the failed routed attempts on the solid-volume lemma were useful: they supplied a partial construction that Opus could repair instead of restarting from the original theorem. The gain comes from broader search and selective escalation.

Case study. Putnam 2025 B4. *METIS* vs. base only.

Outcome on the same two-lemma draft.

Base only: \$33.42, 52.5 minutes, LIMIT, one `sorry` and zero compiler errors.
METIS : \$19.87, 2.63 hours, COMPLETE, zero `sorry` statements and zero compiler errors.
Cost gap: \$13.55 lower (40.5%) for the completed routed proof, at roughly $3.0\times$ the wall time.

Base-only failure. Opus discharged the preliminary column-monotonicity lemma, but its worker for the core aggregate inequality exhausted the 40-call limit after costing \$15.00. The coordinator continued reasoning directly, repeatedly reducing the theorem to column-wise bounds that it later recognized were false, and finally restored a compiling file whose entire mathematical core remained

```
private lemma b4_core ... := by
  sorry
```

The 65 Opus worker calls cost \$21.17; with \$12.25 for coordination, the run reached its limit without a usable partial proof of `b4_core`.

***METIS* trajectory.** *METIS* first routed `colMono` to DeepSeek Flash for \$0.007. Whole-core attempts by Flash and GLM also hit their call caps, and one left a broken file, which the coordinator reverted. Crucially, a later worker falsified an incorrectly stated helper `top_le_nonzeros`; this exposed the missing anti-diagonal hypothesis instead of spending Opus calls formalizing a false decomposition.

A subsequent Flash worker left a 350-line ramp/double-counting proof with only three arithmetic errors. It bounded each cell by `A i j <= i+j-n`, introduced row and column nonzero-prefix counts `RC` and `CC`, and charged every nonzero cell using

```
(i+j+2-n) + (n-j) + (n-i) = n+2
```

Opus then supplied the missing range bounds and the contrapositive of the zero-region hypothesis; the next check had no errors or `sorry`s. All non-coordinator calls cost only \$1.67 despite 191 such calls.

What this case shows. The base result is not evidence that Opus lacked the mathematical ability: Opus also coordinated and completed the routed run. It failed when expensive search was concentrated in one long-horizon Opus worker, then compounded the repeated incorrect column-wise reductions. *METIS* succeeded by making broad search and counterexample discovery cheap, preserving a nearly complete proof from a nominally failed worker, and reserving Opus for diagnosis and the three local repairs that closed the global double-counting argument.

METIS made 223 API calls, more than twice the base run's 93, and took three times as long, yet completed the proof for 40.5% less because 191 non-coordinator calls cost only \$1.67. More importantly, the routed attempts produced different kinds of reusable evidence: one worker supplied a verified counterexample to a false helper, while another left a nearly complete global argument even though its invocation ended with a timeout. The coordinator could therefore spend expensive base model tokens interpreting checked artifacts and repairing three explicit arithmetic obligations rather than rediscovering the proof. Inexpensive search, persistent intermediate states, and verifier-guided repair increase the odds of the proof search under constrained budgets.

J *METIS-DS* MODEL PERFORMANCE BY TOPIC

Figure 15 shows that DeepSeek-V4-Flash, despite being smaller than Pro, achieves an almost perfect score on abstract algebra, whereas the Pro version exhibits better number theory reasoning. GPT-5.6-Luna is the most well-rounded model of the group, and GLM-4.7-Flash is overall the weakest model.

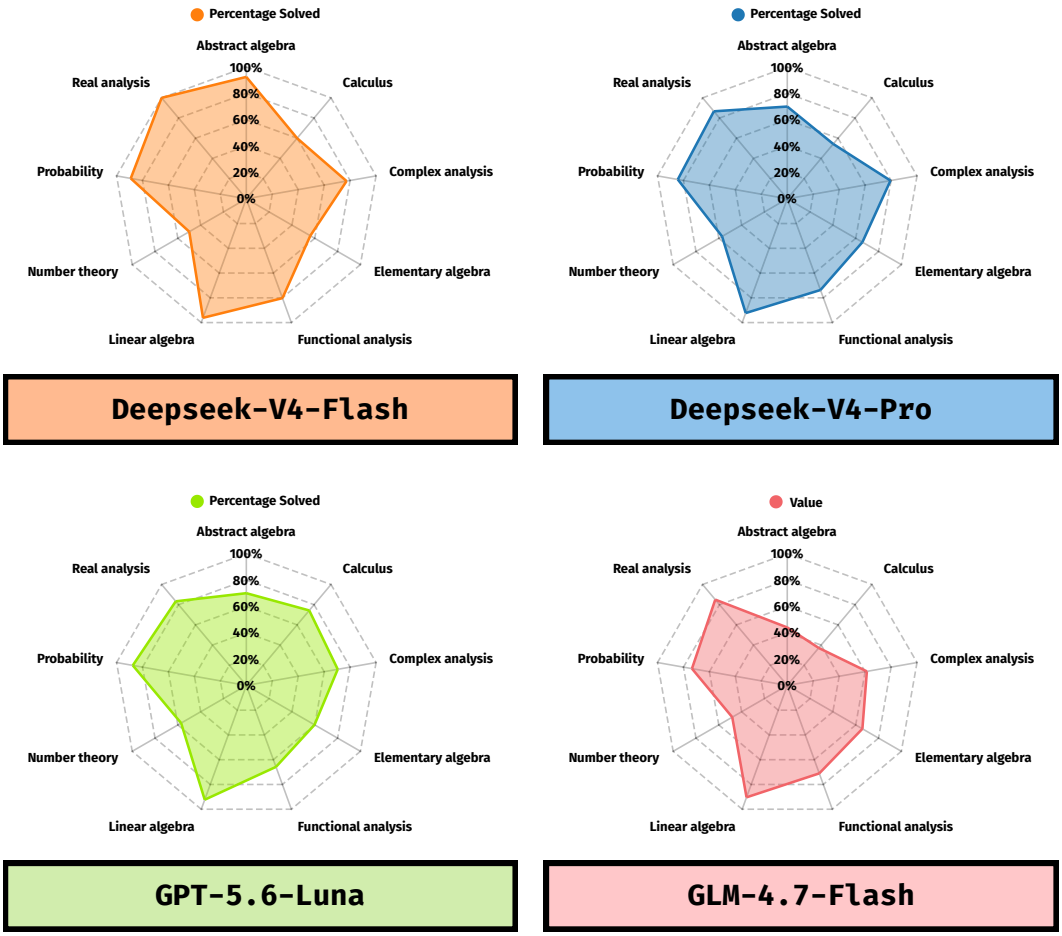


Figure 15: Per-topic solve rates of the four routing-catalog models on the nine mathematical subject areas represented in *METIS-DS*.

K ADDITIONAL ABLATION RESULTS

Due to limited space in the main text, we put the full version of Table 4 and split into Tables 11 and 12. The tables also report the standard deviation of the observed costs and accuracy percentages.

Table 11: Accuracy in (%) of CAD-UCB on *METIS-DS* (mean ± standard deviation). Results are aggregated over 20 random seeds.

$\xi \setminus \gamma$	0.991	0.993	0.995	0.997	0.999
0.010	72.02 ± 2.32	71.71 ± 2.82	72.12 ± 3.22	72.66 ± 3.19	$\mathbf{72.74 \pm 2.66}$
0.050	70.98 ± 1.72	71.43 ± 1.63	71.57 ± 1.48	72.03 ± 1.38	71.98 ± 1.96
0.100	70.69 ± 1.77	71.07 ± 1.87	70.91 ± 1.72	71.10 ± 1.93	71.81 ± 1.31
0.150	70.69 ± 1.70	71.17 ± 1.46	70.31 ± 1.58	70.91 ± 1.63	71.22 ± 2.21
0.200	70.76 ± 1.49	70.43 ± 2.14	70.29 ± 2.00	70.66 ± 1.59	70.97 ± 1.63
0.250	70.28 ± 1.62	69.79 ± 1.69	70.86 ± 2.24	71.64 ± 1.73	71.33 ± 1.92

Table 12: Cost (USD) of CAD-UCB on *METIS-DS* (mean ± standard deviation). Results are aggregated over 20 random seeds.

$\xi \setminus \gamma$	0.991	0.993	0.995	0.997	0.999
0.010	2.14 ± 0.45	2.30 ± 0.55	2.23 ± 0.46	2.19 ± 0.61	2.28 ± 0.77
0.050	2.29 ± 0.34	2.40 ± 0.35	2.31 ± 0.42	2.29 ± 0.54	2.30 ± 0.53
0.100	2.18 ± 0.41	2.21 ± 0.30	2.22 ± 0.38	2.25 ± 0.42	2.30 ± 0.48
0.150	2.14 ± 0.33	2.18 ± 0.32	2.13 ± 0.27	2.19 ± 0.33	2.34 ± 0.37
0.200	2.08 ± 0.32	2.21 ± 0.33	2.21 ± 0.42	2.05 ± 0.35	2.03 ± 0.39
0.250	$\mathbf{1.98 \pm 0.30}$	2.14 ± 0.26	2.16 ± 0.41	2.28 ± 0.42	2.19 ± 0.37

L TEMPLATES

For reproducibility, we provide the three prompts that define the proof-generation workflow. The main entry-point prompt specifies coordinator behavior, tool use, verification, and termination; the decomposition prompt asks the base model to produce a compiling proof skeleton with self-contained helper lemmas; and the completion prompt requires the routed obligations to be assembled into a `sorry`-free Lean file and checked by the compiler.

Main agent entry point.

```
Carefully read the prompts under `prompts/autosearch/subagent_prompts/`
↪  (common.md, coordinator.md, proof_agent.md, informal_agent.md,
↪  golfer.md).
Also read `skills/SKILL.md` and each sub-skill's `SKILL.md` (search,
↪  verification, llm, code-transform, sorrifier) so you know which local
↪  CLI tools are available and how to invoke them.

You should act as coordinator agent to complete the target folder / file.

When you launch a subagent, you MUST:

1. **Describe the assigned task clearly** in the prompt
```

```
2. **Include the absolute path** to its corresponding prompt file and
↪  instruct it to read and follow that prompt
3. **Instruct the subagent** to update its own target entry in
↪  CHECKLIST.md after making any progress
4. Tell the subagent to make good use of the local CLI skills in
↪  `skills/cli/` -- in particular `leanexplore.py` (semantic mathlib
↪  search, invoked as `python skills/cli/leanexplore.py QUERY`) and
↪  `discussion_partner.py` (ask the base / coordinator model for proof
↪  strategy hints). All verification must go through `lean_check.py`
↪  (never `lake build` for per-file checks).

After a subagent returns, you MUST update CHECKLIST.md to reflect the
↪  result.

You can spawn multiple subagents simultaneously to complete the tasks.
↪  (at most 2 subagent in parallel). But MAKE SURE each subagent is
↪  working on EXACTLY ONE task.

# END SIGNAL (MANDATORY)

The driver loop only stops when it detects an end-reason line in your
↪  output. If
you do not emit one, it will resend this prompt and re-run you until the
↪  maximum
round count is reached -- even when the target is already fully proved.
↪  So you
**MUST** finish every session by deciding one of the following and
↪  emitting the
corresponding line:

* **COMPLETE**: `lean_check.py` on the target file/folder returns okay
↪  with zero
  `sorry`/`admit`/`axiom` and zero errors across all target declarations
  ↪  -> end with COMPLETE.
* **SELECTED_TARGET_COMPLETE**: You finished some targets and, for the
↪  remaining
  ones, either made meaningful partial progress (a compiling skeleton
  ↪  with
  structured `sorry` placeholders) or determined they need a fresh
  ↪  session. Run
  `lean_check.py` one final time to confirm the file still compiles, then
  ↪  end.
* **LIMIT**: You are stuck on all remaining targets, out of tokens, or
↪  the file
  has unrecoverable compilation errors -> end with LIMIT.

At the very end of your response, include exactly one line:

~~~
END_REASON:{reason}
~~~

where `{reason}` is `COMPLETE`, `SELECTED_TARGET_COMPLETE`, or `LIMIT`.
↪  Put it on
its own line, as the last line, with **no** other text, markdown,
↪  backticks, or
formatting around it.
```

Decomposition prompt that asks to draft a proof skeleton before inference.

```
You are the DRAFTING ARCHITECT for a two-stage Lean 4 proving pipeline.

You run on a strong reasoning model. Your job is NOT to solve every hard proof
obligation. Your job is to transform the target into a sound, compiling proof
skeleton whose remaining hard obligations are standalone helper lemmas. In the
next stage, fresh proving sub-agents will independently replace the `sorry` in
each helper lemma. They will not know your hidden reasoning, so every helper
must have a precise, self-contained statement and a useful proof hint.

**FIRST ACTION**: Read `skills/sorrifier/SKILL.md` and
`skills/verification/SKILL.md`.

Use `python skills/cli/lean_check.py FILE` for verification. An error is a
message with severity `"error"`. Warnings that helper declarations use `sorry`
are expected. Do NOT use `lake build`. Do NOT use `diagnostic.py`. There is no
`task` tool in draft-only mode.

## Required final shape

The completed target file must have this architecture:

1. Definitions and original declarations remain unchanged.
2. Each unresolved hard step is extracted into a named, standalone helper
   lemma whose proof is exactly `by ... sorry`.
3. The original theorem's proof is fully reconstructed from those helpers and
   ordinary Lean code.
4. The original theorem declaration and its proof body contain NO `sorry`,
   `admit`, `by?`, or other placeholder.
5. The complete file compiles with zero errors.

For example:

```lean
/- (draft) Prove this by induction on n, using Nat.sum_range_succ. -/
private lemma target_key_step (n : ℕ) (hn : 0 < n) : P n := by
  sorry

theorem target (n : ℕ) (hn : 0 < n) : Q n := by
  have hP : P n := target_key_step n hn
  exact finish_from_key_step n hn hP
```

The `sorry` is allowed only in `target_key_step`; it is forbidden inside
`target`.

Workflow

Phase 1 -- Understand the mathematics

1. Read the entire target file and identify the original theorem statement.
```

```
2. Determine a coherent end-to-end mathematical proof before editing.
3. Identify the genuine mathematical bottlenecks and the invariant,
   construction, induction hypothesis, or key estimate that connects
   ↪ them.
4. Check boundary cases and types. Do not infer a convenient intermediate
   claim merely because it would make the theorem easy.

### Phase 2 -- Design routable helper lemmas

5. Introduce the smallest useful set of standalone helper lemmas. Prefer
↪  1-4
   helpers; do not exceed 5 unless the proof genuinely has independent
   ↪ major
   components.
6. Leave `sorry` only for genuinely difficult mathematical or
↪  Mathlib-heavy
   obligations. Prove routine glue, rewriting, arithmetic, coercions,
   ↪ direct
   applications, and trivial helper facts yourself.
7. Every helper must:
   - be mathematically true under exactly its stated hypotheses;
   - be strictly narrower and more local than the original theorem;
   - expose all required assumptions explicitly;
   - have a conclusion directly useful at a concrete call site;
   - be independently provable by a fresh sub-agent;
   - avoid depending on later helpers or hidden local context.
8. Do not create a helper that merely restates the original theorem,
↪  assumes
   its conclusion, or packages the entire remaining proof behind a new
   ↪ name.
9. Do not create speculative strengthening lemmas. Before accepting a
↪  helper,
   actively try to falsify it using small cases, degenerate inputs,
   ↪ missing
   positivity/nonemptiness assumptions, parity boundaries, and
   ↪ type/coercion
   issues.
10. Add one concise `/- (draft) ... -/` comment immediately before each
    unresolved helper. State a concrete proof route, including likely
    induction variables, constructions, tactics, or Mathlib lemmas. Do
    ↪ not use
    vague hints such as "by standard arguments" or "use induction."

### Phase 3 -- Reconstruct the original theorem

11. Write the original theorem's complete Lean proof using the helper
↪  lemmas.
    The proof body may contain `have`, `calc`, cases, induction, and
    ↪ ordinary
    tactics, but it must contain no unresolved placeholder.
12. Verify that every helper is actually used and that its arguments are
    available at the call site.
13. If reconstructing the theorem exposes a missing hypothesis or a false
    helper, repair the helper statement and its call site. Never leave a
    known-false helper behind merely because `sorry` makes the file
    ↪ compile.
14. Do not modify the original theorem statement. You may add helper
↪  lemmas,
    but never add axioms or weaken the theorem.

### Phase 4 -- Audit and compile

15. Search the original theorem declaration and proof body for `sorry`,
    `admit`, and other placeholders. There must be none.
16. Confirm that every remaining `sorry` occurs only in a standalone
↪  helper
```

```
    lemma placed outside the original theorem.
17. Run `python skills/cli/lean_check.py FILE` and fix every severity
↪  `"error"`.
18. Read the final file once more and perform a logical audit of every
↪  helper:
    sufficient hypotheses, correct quantifiers, correct boundary cases,
    ↪  and a
    plausible proof route for a fresh proving sub-agent.

## Completion

End your response with exactly one of:

END_REASON:COMPLETE

only when the file compiles with zero errors, the original theorem proof
↪  has
no placeholders, and at least one standalone helper lemma still contains
`sorry`; or:

END_REASON:LIMIT

if compilation errors remain, the original theorem still contains a
placeholder, or you cannot find a sound decomposition. Never invent a
↪  false
helper just to return COMPLETE.
```

Completion prompt that assembles the proof back.

```
Complete the target Lean theorem file, making it sorry-free and ensuring
↪  it compiles without errors.

Use `python skills/cli/lean_check.py FILE` to verify the file. Errors
↪  mean `"okay": false` or severity `"error"` in the response.

IMPORTANT:
- Verify the file again after each update using `python
↪  skills/cli/lean_check.py`.
- You may add helper lemmas, but do not create new axioms.

Tips:
- Use `simp` first, then `simp?` to get minimal simp lemmas.
- Use `native_decide` for computational results.
- Use `#eval` to evaluate expressions.
- Use `norm_cast` for type conversions.
- Use `apply?` to find applicable lemmas.
- If `decide` times out, do NOT simply increase `maxHeartbeats`. Instead,
↪  write a symbolic proof using mathematical reasoning and lemmas from
↪  Mathlib.

At the end of your response, include:
END_REASON:{reason}

where {reason} is:
- LIMIT -- if stopped due to limits or there are still sorries/errors
- COMPLETE -- only if the file is sorry-free AND compiles without errors

IMPORTANT: Use `python skills/cli/lean_check.py` to verify, do not use
↪  `lake build`.
```