BUDGET-AWARE PREDICT-AND-VERIFY THEOREM PROVING WITH HETEROGENEOUS LLM SOLVERS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language model (LLM) theorem solvers differ in both proving capability and inference cost. Applying an expensive solver to every subgoal wastes computation when cheaper models are sufficient, while always starting from the cheapest solver wastes attempts on subgoals that require stronger models. We present BudgetPV, an agentic budget-aware predict-and-verify system that allocates computation across unresolved subgoals and heterogeneous LLM solvers. BudgetPV uses a cross-solver predictor to guide solver selection for each subgoal and revises the allocation as verification feedback becomes available.

The allocation problem changes as proving progresses. Early decisions mainly determine which solver should handle an untried subgoal, while later decisions determine which previously attempted subgoals should receive additional computation. We train the predictor with two objectives for these two stages. Class-weighted cross-entropy improves assignment to less frequent stronger solvers during the initial stage, while plain cross-entropy provides better calibrated probabilities for later allocation among previously attempted subgoals.

We evaluate BudgetPV on Lean Workbook, miniF2F, ProofNet, and PutnamBench using three progressively expanded solver settings: a strong but expensive solver alone, adding a medium-cost solver with intermediate solving capability, and further adding a cheap solver with lower solving capability. BudgetPV requires less LLM generation computation than uniform allocation to solve the same number of subgoals across the evaluated settings. The experiments further show the computation savings obtained as medium- and lower-cost solvers are added.

1 INTRODUCTION

Large language models (LLMs) have improved formal theorem proving. Recent systems generate complete proofs, search over proof states, retrieve useful lemmas, learn from proof-assistant feedback, and decompose difficult proofs into smaller steps Polu and Sutskever (2020); Yang et al. (2023); Xin et al. (2024a;b); Ren et al. (2025); Lin et al. (2025; 2026). Many of these gains rely on increased computation at inference time. A common approach generates multiple candidate proofs for the same theorem and submits them to a proof assistant, such as Lean, which mechanically checks formal proofs de Moura and Ullrich (2021). Increasing the number of candidates often raises the probability of finding a valid proof Brown et al. (2024); Snell et al. (2024), but also increases inference and verification costs.

Proof decomposition represents a theorem as a set of smaller subgoals that can be solved separately. The Draft, Sketch, and Prove (DSP) method constructs a formal proof sketch, a partial proof that specifies intermediate subgoals, and proves these subgoals independently Jiang et al. (2023). Recursive and hierarchical proving methods further divide difficult goals into smaller subgoals Wang et al. (2024); Dong et al. (2024); Ren et al. (2025). This decomposition changes the unit of computation from the whole proof to individual subgoals.

However, the difficulty of subgoals can vary widely. Some decomposed subgoals can be solved without LLM generation, while others remain unsolved after repeated proof attempts. A theorem is complete only when every required subgoal is proved. Once a subgoal is solved, further computation on it has no value, while repeated proof generation for a subgoal that continues to fail can consume

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.

computation that could be used elsewhere. Different subgoals should receive different amounts of computation as the proof progresses.

The allocation problem becomes more complex when multiple LLM solvers with different capabilities and inference costs are available. A stronger solver can solve subgoals that weaker models cannot, but using it for every subgoal wastes computation when a cheaper solver is sufficient. Conversely, always starting from the cheapest solver wastes attempts on subgoals that require a stronger model. Efficient allocation must decide both which unresolved subgoal should receive computation and which solver should be used.

We propose BudgetPV, an agentic budget-aware predict-and-verify system that manages the available computation budget across the subgoals of a proof sketch. Each subgoal is first processed using standard proof procedures provided by a proof assistant, such as Lean. If it remains unsolved, LLM solvers generate candidate proofs for the proof assistant to check. Rather than assigning the same number of generations to every unresolved subgoal or using the same solver for all subgoals, BudgetPV uses a cross-solver predictor to guide the allocation of proof attempts across subgoals and solvers according to predicted success and generation cost. These estimates are updated as generated proofs succeed or fail, so the allocation changes as proving progresses.

The allocation problem also changes during the proving process. Early in the process, most unresolved subgoals have not yet been attempted, and the main decision is which solver should handle each subgoal. Later, many unresolved subgoals have already received proof attempts, and the decision shifts toward determining which of them should receive additional computation. We train the same cross-solver predictor with two objectives for these two stages. Class-weighted cross-entropy gives greater weight to the less frequent stronger-solver classes and improves initial solver assignment, while plain cross-entropy provides better calibrated probabilities for later allocation among previously attempted subgoals.

Adaptive budget allocation across subgoals has also appeared in recent theorem solvers. AlphaProof allocates computation within its AND–OR tree search Hubert et al. (2026), while BFS-Prover-V2 allocates solver effort within its planner-based search process Xin et al. (2025). In these systems, budget allocation is part of the proving algorithm itself. BudgetPV instead manages budget allocation on top of the underlying proving process. It observes the current subgoals and proof-attempt history, uses its cross-solver predictor to evaluate available subgoal–solver actions, and decides where additional computation should be spent. BudgetPV acts as an agent over the proving process, repeatedly selecting a subgoal–solver action, observing its verification result, and using the updated state to make the next allocation decision.

We evaluate BudgetPV on miniF2F Zheng et al. (2022), ProofNet Azerbayev et al. (2023), PutnamBench Tsoukalas et al. (2024), and Lean Workbook Ying et al. (2024), using three settings that progressively introduce solvers with different capability and cost: a strong but expensive solver alone, adding a medium-cost solver with intermediate solving capability, and further adding a cheap solver with lower solving capability. These solvers are Goedel-Prover-V2-32B Lin et al. (2026), Goedel-Prover-V2-8B Lin et al. (2026), and Kimina-Prover-Distill-8B Numina and Kimi Team (2025). BudgetPV is tested in both live and replay settings. The live closed-loop evaluation reflects the real proving scenario, where proof candidates are generated during execution, checked by Lean, and the allocation is updated from the observed outcomes. Controlled replay uses the same recorded proof attempts for all compared methods, removing variation from independent generation. Across the evaluated dataset–solver settings, BudgetPV requires less LLM generation computation than uniform allocation to solve the same number of subgoals. The experiments also show the progressive benefit of adding the medium- and lower-cost solvers.

Our contributions are as follows.

1. We introduce BudgetPV, an agentic system for managing computation jointly across the unresolved subgoals of a proof sketch and LLM solvers with different solving capability and inference cost. BudgetPV works on top of the underlying proving process and uses information produced during proving to guide budget allocation.

2. BudgetPV uses a cross-solver predictor to guide the solver assignment for each unresolved subgoal. We distinguish the initial solver-assignment stage from the later repeated-attempt stage and train the predictor with class-weighted and plain cross-entropy objectives for these

two stages, respectively. Verification outcomes update the information used by the agent for subsequent allocation.

3. We evaluate BudgetPV on four theorem-proving datasets using three progressively expanded solver settings: a strong but expensive solver alone, adding a medium-cost solver with intermediate capability, and further adding a cheap solver with lower capability. BudgetPV requires less LLM generation computation than uniform allocation to solve the same number of subgoals, and the experiments show the computation savings obtained by adding medium- and lower-cost solvers.

2 Related Work

Neural theorem proving and proof search. LLM-based theorem provers generate complete proofs, predict proof steps, or search over intermediate proof states Polu and Sutskever (2020); Xin et al. (2024a;b); Lin et al. (2025; 2026). HyperTree Proof Search uses model predictions to guide tree search Lample et al. (2022), while LeanDojo combines proof generation with retrieval of relevant premises from the Lean library Yang et al. (2023). More recent systems also control how computation is distributed during proof search. AlphaProof selects among unresolved subgoals within an AND–OR tree and directs search toward difficult branches Hubert et al. (2026), while BFS-Prover-V2 combines a high-level planner with multiple prover agents to search over planned subgoals Xin et al. (2025). In these systems, computation management is part of the proof-search method. BudgetPV instead provides a separate agentic mechanism for managing computation across subgoals and can be used without changing the internal search algorithm of the prover.

Proof decomposition. A theorem is usually decomposed into smaller subgoals to handle long and difficult proofs. Draft, Sketch, and Prove (DSP) converts an informal proof into a formal proof sketch whose open subgoals are proved separately Jiang et al. (2023). POETRY recursively constructs proof sketches and proves intermediate conjectures at successive levels Wang et al. (2024), while hierarchical proving methods train models to introduce and prove useful intermediate lemmas Dong et al. (2024). DeepSeek-Prover-V2 also uses recursive subgoal decomposition to construct its cold-start training data Ren et al. (2025). BudgetPV starts from the decomposed subgoals and allocates computation across both unresolved subgoals and available LLM solvers.

Adaptive computation and automated proof procedures. A proof assistant can often solve some goals directly using standard automated procedures, without LLM generation. Hammers and related automation have been used in formal theorem proving Paulson and Blanchette (2010), and THOR combines language models with automated reasoning tools Jiang et al. (2022). More broadly, recent work shows that allocating inference according to problem difficulty can be more efficient than applying the same amount of computation to every input Snell et al. (2024); Brown et al. (2024). Related work on input-adaptive computation allocates repeated generations according to predicted per-input success and marginal gain Damani et al. (2025). BudgetPV considers the same general problem in decomposed theorem proving, where computation must be allocated jointly across subgoals and solvers with different capabilities and costs.

Predicting proof success and solver selection. Prediction of model correctness or success has been studied using model confidence Kadavath et al. (2022), calibrated probability estimation Guo et al. (2017), learned correctness models Xiao et al. (2025), hidden-state and activation-based predictors Varshney et al. (2026); Di Cicco (2026), and kNN-based model routing Li (2025). These methods estimate whether a model is likely to succeed on a given input and can be used to guide inference-time computation. In formal theorem proving, each generated candidate is checked by the proof assistant, providing direct feedback from previous attempts. BudgetPV uses a cross-solver predictor rather than independent per-solver predictors. It estimates which solver is appropriate for each unresolved subgoal and updates the allocation as verification feedback becomes available.

3 Method

BudgetPV is an agentic budget manager for formal theorem proving over decomposed proofs. Given a proof sketch, each subgoal is first processed using standard proof procedures provided by the proof

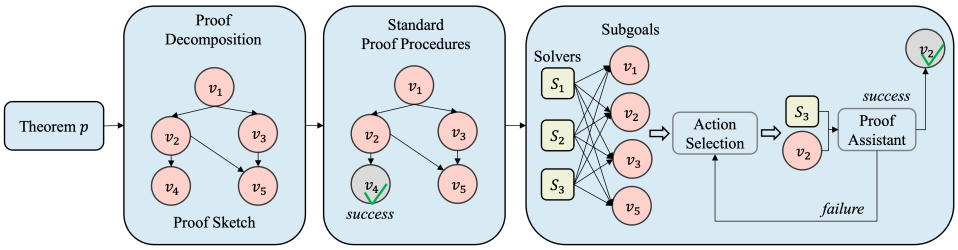

Figure 1: The overall framework of BudgetPV. A theorem p is decomposed into a proof sketch, and standard proof-assistant procedures first process all subgoals. At each decision step, BudgetPV predicts the success probability of each available LLM solver for each unresolved subgoal and selects a subgoal–solver action based on predicted success, computation cost, and the remaining allowed proof attempts. The selected subgoal is processed using the corresponding solver and verified by the proof assistant. The outcome is used to update the predictions for the next allocation decision.

assistant. BudgetPV then manages the computation spent on the unresolved subgoals using the available LLM solvers.

Figure 1 gives an overview of the framework. At each decision step, BudgetPV predicts the probability that each available LLM solver can successfully solve each unresolved subgoal. It then scores the available subgoal–solver actions according to predicted success and generation cost. The selected actions are executed and the generated proofs are checked by the proof assistant. Failed attempts provide additional evidence for the corresponding subgoals, and BudgetPV updates their success estimates before the next allocation decision. This predict-and-verify process continues until the proof is completed or the available computation budget is exhausted.

3.1 Subgoal–Solver Action Space

Given a theorem p, a proof-decomposition method first produces a proof sketch $G_p = (V_p, E_p)$, where each node $v_i \in V_p$ is a formal subgoal and an edge $(v_i, v_j) \in E_p$ indicates that the result of v_i is used in proving v_j. Existing decomposition methods, such as DSP, recursive proving, or hierarchical proving Jiang et al. (2023); Wang et al. (2024); Dong et al. (2024), can be used to generate the proof sketch. For each subgoal, BudgetPV records its current status and the results of previous proof attempts. A theorem is completed when all required subgoals in its proof sketch have been proved.

BudgetPV first applies standard proof procedures provided by the proof assistant to every extracted subgoal. Procedures such as *simp*, *aesop*, *omega*, and *nlinarith* can directly solve some subgoals without LLM inference. Subgoals solved at this stage are removed from further processing, and BudgetPV allocates the LLM computation budget only to the remaining unresolved subgoals.

For these unresolved subgoals, BudgetPV can choose among multiple LLM-based solvers. Let $\mathcal{S} = \{S_1, \ldots, S_M\}$ denote the set of M available LLM solvers. Different solvers may have different proving capabilities, success probabilities, and computation costs for the same subgoal.

For an unresolved subgoal v_i, selecting solver S_j defines a *proving action*

$$a_{i,j} = (v_i, S_j).$$

A *proof attempt* is one execution of a proving action. Several proof attempts may use the same subgoal–solver pair because an LLM solver can generate multiple candidate proofs for the same subgoal. Each generated proof is independently checked by the proof assistant.

At decision step t, let U_t denote the set of remaining unresolved subgoals. BudgetPV considers the candidate proving actions

$$\mathcal{A}_t = \left\{ (v_i, S_j) \mid v_i \in U_t,\ S_j \in \mathcal{S},\ n_{i,j}^{(t)} < K_{i,j} \right\},$$

where $n_{i,j}^{(t)}$ is the number of attempts already made using solver S_j on subgoal v_i, and $K_{i,j}$ is the maximum number of attempts allowed for that pair. The same subgoal may have different success probabilities and costs under different solvers. That is, BudgetPV allocates computation over subgoal–solver pairs rather than over subgoals alone.

3.2 CROSS-SOLVER SUCCESS PREDICTION

BudgetPV uses a single cross-solver predictor to estimate which solvers are suitable for each unresolved subgoal. We index the solver classes $S_1, \ldots, S_M$ in increasing order of generation cost. For subgoal v_i at decision step t, let $H_i^{(t)}$ denote the evidence available from previous proof attempts, including their verification results and errors. The predictor Θ takes v_i and $H_i^{(t)}$ as input and computes a probability distribution over the M solvers and one additional `none` class:

$$q_{i,c}^{(t)} = \Theta(c \mid v_i, H_i^{(t)}), \qquad c \in \{S_1, \ldots, S_M, \text{none}\}.$$

For training, the label is the least expensive available solver that closes the subgoal in the recorded proof-attempt pools. The `none` label is used when none of the available solvers closes the subgoal. Thus, `none` is an additional prediction class, not a solver.

The class probabilities are converted into a success estimate for each solver. Probability assigned to a subgoal being solvable by a less expensive solver also contributes to the success estimate of each more expensive solver. Let $\widehat{p}_{i,j}^{(t)}$ denote the estimated probability that solver S_j closes subgoal v_i at decision step t. Then

$$\widehat{p}_{i,j}^{(t)} = \sum\nolimits_{k=1}^{j} q_{i,S_k}^{(t)}. \tag{1}$$

3.3 TRAINING FOR DIFFERENT STAGES OF PROVING

The allocation decision changes as proving progresses. Early in the process, most unresolved subgoals have not yet been attempted, so the main decision is which solver should be used for each subgoal. Later, many unresolved subgoals have already received proof attempts, and the main decision is which of these subgoals should receive additional computation. We train two instances of the same cross-solver predictor architecture for these two stages.

The predictors are trained using only the formal subgoal, without previous proof attempts or proof-assistant feedback. Each training subgoal v_i is assigned a training label y_i that is the least expensive solver that closes v_i in the recorded proof-attempt pools. If none of the available solvers closes v_i, then $y_i = \texttt{none}$.

For the initial stage, we use class-weighted cross-entropy. Let $q_{i,c}^{A} = \Theta_A(c \mid v_i)$ denote the probability assigned to class c by the class-weighted predictor. The loss is

$$\mathcal{L}_A = -\sum\nolimits_{i=1}^{N} w_{y_i} \log q_{i,y_i}^{A}, \qquad w_c = \frac{N}{(M+1)n_c},$$

where N is the total number of training subgoals and n_c is the number of training subgoals in class c. The weighting gives greater relative weight to the less frequent solver classes and improves initial solver assignment.

For the later stage, we use plain cross-entropy. Let $q_{i,c}^{B} = \Theta_B(c \mid v_i)$ denote the probability assigned to class c by the plain-cross-entropy predictor. The loss is

$$\mathcal{L}_B = -\sum\nolimits_{i=1}^{N} \log q_{i,y_i}^{B}.$$

Plain cross-entropy retains the empirical class distribution and provides better calibrated probabilities for comparing previously attempted subgoals.

At inference time, BudgetPV uses Θ_A during the initial stage and switches to Θ_B once the unresolved set becomes dominated by previously attempted subgoals. The complete proving state is carried across the switch, including previous proof attempts, verification results, attempt counts, and computation already spent.

3.4 UPDATING SUCCESS AFTER FAILED ATTEMPTS

A failed proof attempt provides additional information about which solvers are likely to close the subgoal. BudgetPV uses the training proof-attempt pools to estimate how the success probability of each solver changes after successive failed attempts.

Let $h_{j,c,r}$ denote the measured success rate of the r-th attempt by solver S_j for training subgoals with class c. After the r-th attempt by S_j fails on the subgoal v_i, the class probabilities are updated according to

$$\widetilde{q}_{i,c}^{(t+1)} = q_{i,c}^{(t)}\left(1 - h_{j,c,r}\right), \quad q_{i,c}^{(t+1)} = \frac{\widetilde{q}_{i,c}^{(t+1)}}{\sum_{c'} \widetilde{q}_{i,c'}^{(t+1)}}. \tag{2}$$

A failure reduces the probability of classes under which that solver was expected to succeed. The updated class probabilities are then converted into new per-solver success estimates for the next allocation decision.

3.5 Budget-Aware Solver Allocation

For each available subgoal–solver pair (v_i, S_j), BudgetPV uses the predicted success probability together with the generation cost of the solver. Let d_j denote the generation cost of solver S_j (e.g. measured by GPU-seconds). We compute a score for (v_i, S_j)

$$g^{(t)}(i, S_j) = \widehat{p}_{i,j}^{(t)}/d_j.$$

BudgetPV ranks the available subgoal–solver pairs by $g^{(t)}(i, S_j)$. A pair is ranked higher when it has a higher predicted probability of success or a lower generation cost. The highest-ranked pairs are selected for the next proof attempts, with at most one solver assigned to each subgoal at a decision step.

After an attempt is executed, the generated proof is checked by the proof assistant. If it succeeds, the subgoal is removed from the remaining subgoal set U_t. If it fails, the attempt and its proof-assistant feedback are added to the evidence state. BudgetPV then updates the class probabilities using Eq. 2 and computes new per-solver success estimates using Eq. 1 for the next decision step $t+1$. BudgetPV repeatedly selects subgoal–solver pairs, verifies the generated proofs, and updates the proving state until all required subgoals are proved or the computation budget is exhausted.

4 Experiments

Benchmarks. We evaluate BudgetPV on miniF2F Zheng et al. (2022), ProofNet Azerbayev et al. (2023), PutnamBench Tsoukalas et al. (2024), and Lean Workbook Ying et al. (2024). Standard proof procedures provided by the proof assistant are first applied to every subgoal, and the LLM solvers are evaluated on the unresolved subgoals. Lean Workbook is also used to train the cross-solver predictors. The final Lean Workbook proof-attempt pool contains 4,358 unresolved subgoals. We split the data at the problem level into 3,488 training subgoals and 870 evaluation subgoals, so that subgoals from the same problem do not appear in both sets. For each Lean Workbook training subgoal and each solver, we generate eight proof attempts and check every generated proof with the proof assistant. These recorded attempts are used to determine the training label for the cross-solver predictor.

Solvers. BudgetPV is tested with three LLM solvers of different proving capabilities and generation costs: Goedel-Prover-V2-32B Lin et al. (2026), Goedel-Prover-V2-8B Lin et al. (2026), and Kimina-Prover-Distill-8B Numina and Kimi Team (2025). We consider three progressive solver settings. The first uses Goedel-Prover-V2-32B alone. The second adds Goedel-Prover-V2-8B, and the third further adds Kimina-Prover-Distill-8B. These settings evaluate how BudgetPV uses additional medium- and lower-cost solvers when they become available.

Predictor training. Both predictors use Goedel-Prover-V2-8B as the backbone and are trained with LoRA using rank 32 and scaling factor 64. LoRA is applied to the q, k, v, o, gate, up, and down projections. We train for three epochs with learning rate 10^{-5}, batch size 16, and maximum sequence length 1,536. Predictors Θ_A and Θ_B use the same training set. During evaluation, BudgetPV uses Θ_A initially and switches to Θ_B when the number of solved subgoals reaches 75% of the coverage obtained by Goedel-Prover-V2-32B. The complete proving state is preserved across the switch.

Compared methods. We compare BudgetPV with *Uniform*, which uniformly allocates proof attempts across the available solvers and unresolved subgoals without using the success predictor.

When multiple solvers are available, Uniform rotates among them according to the number of previous attempts, using the lower-cost solver first when the attempt counts are equal.

We additionally report a *Statistics Oracle* reference, which uses the measured success rate of each solver in the complete proof-attempt pools in place of the predicted success probabilities. The oracle reference indicates how computation could be further saved with more accurate success estimation.

Evaluation metric. The main computation metric is the GPU-seconds of LLM proof generation required to reach fixed subgoal-solving targets. For each dataset, we use the subgoals closed by Goedel-Prover-V2-32B alone as the reference set. We report the accumulated generation GPU-seconds required by each method to solve 50%, 60%, 70%, 80%, and 90% of the subgoals in this reference set. The success predictor is evaluated using AUROC Fawcett (2006) in Appendix B.

Live execution and controlled replay. We evaluate BudgetPV in both live execution and controlled replay. The live closed-loop evaluation reflects the real proving scenario, where the solvers generate proofs that are checked by Lean and the allocation is updated from the observed outcomes. Controlled replay uses recorded proof attempts and their measured costs instead of generating new proofs. This removes differences caused by independent proof generation. Multiple replay runs are constructed by presenting the recorded attempts in different orders, allowing us to measure how performance changes when successful attempts are encountered at different points. Replay also provides the complete outcome pools needed for the oracle comparisons.

4.1 Live Closed-Loop Evaluation

Table 1 reports the live closed-loop results. Across all four datasets and all three solver settings, BudgetPV requires less generation computation than Uniform at every reported solve rate. From the live results we can see two complementary sources of computation saving. First, BudgetPV already reduces generation cost when only Goedel-Prover-V2-32B is available. Since there is no solver-selection decision in this 1-solver setting, these gains come from allocating proof attempts more effectively across unresolved subgoals. For example, at the 50% solve rate, BudgetPV reduces generation computation of Uniform from 11,572 to 6,047 GPU-seconds on Lean Workbook, from 4,423 to 1,656 on PutnamBench, from 2,672 to 1,231 on miniF2F, and from 1,086 to 551 on ProofNet.

Second, access to additional lower-cost solvers provides another source of saving. BudgetPV can use these solvers for subgoals that do not require the most expensive model. On Lean Workbook at the 50% solving-rate target, the computation required by BudgetPV decreases from 6,047 GPU-seconds with one solver to 3,402 with two solvers and 3,213 with three solvers. On PutnamBench, the corresponding values are 1,656, 731, and 654 GPU-seconds. These results show that BudgetPV benefits both from deciding where to spend computation and from deciding which solver should receive that computation.

The savings are largest at lower and intermediate solving-rate targets. At these stages, many unresolved subgoals can still be solved without repeatedly calling the strongest solver. As the solve rate increases, the unresolved set becomes increasingly concentrated on harder subgoals, and additional computation from the stronger solver becomes more important. This explains why the computation difference between BudgetPV and Uniform generally decreases toward the highest solve rates.

4.2 Controlled Replay and Oracle Evaluation

Table 2 reports the controlled replay results together with the oracle reference. The same overall pattern can be observed as the live evaluation. BudgetPV requires less generation computation than Uniform at every reported solving-rate target across all four datasets and all three solver settings. The reductions are again generally larger at lower and intermediate solving rates. For example, with three solvers, BudgetPV reduces the 50% computation from 2,275 for Uniform to 997 GPU-seconds on Lean Workbook, from 1,188 to 439 on PutnamBench, from 590 to 517 on ProofNet, and from 341 to 234 on miniF2F. At 90% solve rate, BudgetPV reduces computation from 16,046 to 13,361 GPU-seconds on Lean Workbook, from 14,439 to 11,002 on PutnamBench, from 5,044 to 2,268 on ProofNet, and from 1,870 to 1,610 on miniF2F.

Table 1: Live closed-loop generation GPU-seconds required to reach the corresponding solving-rate target. Results are mean ± standard deviation over five runs. Lower is better.

Dataset	Solvers	Method	50%	60%	70%	80%	90%
miniF2F	1	Uniform	2,672±193	3,248±110	3,580±63	3,949±77	5,056±245
	1	BudgetPV	**1,231±70**	**1,556±18**	**1,955±139**	**2,870±136**	**4,881±649**
	2	Uniform	2,371±144	2,771±144	3,033±311	3,428±311	4,484±188
	2	BudgetPV	**1,249±161**	**1,390±83**	**1,652±145**	**2,282±144**	**4,096±268**
	3	Uniform	2,267±106	2,446±210	2,812±109	2,989±200	4,195±269
	3	BudgetPV	**1,163±136**	**1,375±236**	**2,033±163**	**2,624±645**	**3,818±450**
ProofNet	1	Uniform	1,086±374	1,613±850	2,355±708	3,658±996	6,048±2,305
	1	BudgetPV	**551±208**	**809±200**	**1,063±111**	**1,901±404**	**4,203±705**
	2	Uniform	941±47	1,302±294	2,102±282	3,314±1,259	6,196±1,262
	2	BudgetPV	**804±225**	**1,032±71**	**1,199±142**	**2,466±680**	**3,324±1,381**
	3	Uniform	1,238±4	1,238±4	2,158±705	4,306±893	8,234±2,098
	3	BudgetPV	**711±195**	**1,209±31**	**1,955±898**	**3,763±2,802**	**6,791±2,539**
PutnamBench	1	Uniform	4,423±660	6,047±798	7,347±954	14,371±5,733	33,575±10,280
	1	BudgetPV	**1,656±124**	**4,900±8**	**6,889±467**	**12,090±1,883**	**25,604±6,277**
	2	Uniform	4,353±750	5,865±636	8,765±2,209	14,027±4,369	26,755±16,452
	2	BudgetPV	**731±320**	**1,311±459**	**5,758±2,055**	**11,882±4,103**	**26,684±5,917**
	3	Uniform	2,769±379	4,698±1,179	8,051±1,258	10,828±1,511	19,677±1,522
	3	BudgetPV	**654±216**	**1,934±396**	**3,986±473**	**8,530±3,091**	**18,329±828**
Lean Workbook	1	Uniform	11,572±263	13,163±85	15,368±238	17,845±437	32,113±2,012
	1	BudgetPV	**6,047±158**	**8,268±370**	**10,713±491**	**17,140±478**	**31,582±2,344**
	2	Uniform	8,914±918	10,089±981	11,279±973	14,074±900	22,889±1,223
	2	BudgetPV	**3,402±365**	**4,526±121**	**6,465±411**	**9,585±414**	**19,482±1,235**
	3	Uniform	7,439±337	8,757±90	11,110±768	15,164±1,229	23,941±4,703
	3	BudgetPV	**3,213±273**	**4,774±49**	**6,809±880**	**10,691±275**	**21,882±1,428**

The agreement between live execution and controlled replay is important because replay removes variation caused by independent proof generation. All methods use the same recorded proof-attempt pools and measured generation costs, while differing only in how computation is allocated. The consistent advantage of BudgetPV in replay confirms that the savings observed in live execution come from better allocation of proof attempts rather than from random variation in proof generation.

Replay also allows us to compare BudgetPV with the Statistics Oracle, which uses the measured success rates of the solvers in the complete proof-attempt pools. The Statistics Oracle provides a reference for how the allocation could improve if the success estimates were more accurate. Its advantage is most visible at the higher solving-rate targets. With three solvers at the 90% target, the Statistics Oracle requires 2,878 GPU-seconds on Lean Workbook, compared with 13,361 for BudgetPV. On ProofNet, the corresponding values are 385 and 2,268 GPU-seconds. PutnamBench shows a similar difference, with 669 GPU-seconds for the Statistics Oracle and 11,002 for BudgetPV. The oracle comparison shows that more accurate success estimation could further reduce computation within the same solver setting. BudgetPV already recovers part of the saving by using its learned success estimates, but considerably more computation can be avoided when the success information is more accurate. This suggests that improving cross-solver success estimation is a direct way to further reduce computation without changing the basic predict-and-verify allocation framework.

Additional analysis and ablations. Appendix A provides additional experimental details, including the generation settings of evaluated solvers and the experiments used to select the three solvers, as well as the detailed training configuration of the cross-solver predictors. Appendix B further evaluates the predictors on the Lean Workbook evaluation set and during live proving.

5 Conclusion

We presented BudgetPV, an agentic predict-and-verify framework for managing computation across unresolved proof subgoals and heterogeneous LLM solvers. BudgetPV estimates the probability that each available solver can close each subgoal and combines this estimate with the measured generation

Table 2: Controlled replay generation GPU-seconds required to reach the corresponding solving-rate target. BudgetPV results are mean ± standard deviation over five runs. Uniform and Statistics Oracle are deterministic for a fixed proof-attempt pool. Lower is better.

Dataset	Solvers	Method	50%	60%	70%	80%	90%
miniF2F	1	Uniform	1,243	1,385	1,558	1,724	2,756
	1	BudgetPV	**724±17**	**891±27**	**1,097±3**	**1,605±2**	**2,205±7**
	1	Statistics Oracle	427	812	954	1,118	1,257
	2	Uniform	682	743	978	1,301	2,180
	2	BudgetPV	**334±1**	**412±2**	**686±98**	**929±3**	**1,958±189**
	2	Statistics Oracle	200	482	626	792	962
	3	Uniform	341	488	700	961	1,870
	3	BudgetPV	**234±1**	**335±38**	**442±7**	**741±83**	**1,610±25**
	3	Statistics Oracle	159	159	450	632	801
ProofNet	1	Uniform	1,158	1,725	2,168	3,095	4,886
	1	BudgetPV	**732±9**	**1,150±2**	**1,508±14**	**2,399±9**	**3,327±7**
	1	Statistics Oracle	165	274	274	402	448
	2	Uniform	1,228	2,904	2,904	4,250	5,957
	2	BudgetPV	**929±9**	**1,347±6**	**1,693±308**	**2,490±231**	**3,031±247**
	2	Statistics Oracle	165	273	273	404	453
	3	Uniform	590	1,374	3,322	3,322	5,044
	3	BudgetPV	**517±34**	**878±28**	**1,218±283**	**1,691±268**	**2,268±38**
	3	Statistics Oracle	159	159	264	264	385
PutnamBench	1	Uniform	3,163	3,787	4,922	10,471	13,057
	1	BudgetPV	**845±0**	**1,406±0**	**4,474±0**	**8,009±619**	**12,682±16**
	1	Statistics Oracle	469	469	469	651	722
	2	Uniform	2,320	3,253	5,376	8,418	17,062
	2	BudgetPV	**706±45**	**1,498±427**	**4,628±1,674**	**7,361±413**	**10,771±640**
	2	Statistics Oracle	529	529	529	529	712
	3	Uniform	1,188	1,617	3,310	6,568	14,439
	3	BudgetPV	**439±81**	**697±197**	**1,831±142**	**6,330±211**	**11,002±128**
	3	Statistics Oracle	481	481	481	481	669
Lean Workbook	1	Uniform	5,487	6,397	7,236	8,256	14,895
	1	BudgetPV	**2,445±207**	**3,209±36**	**4,576±19**	**6,468±4**	**13,120±7**
	1	Statistics Oracle	1,915	2,940	3,477	3,857	3,996
	2	Uniform	2,743	3,073	4,204	8,335	14,757
	2	BudgetPV	**1,078±4**	**1,654±33**	**3,349±150**	**6,491±165**	**11,796±141**
	2	Statistics Oracle	1,374	2,227	2,582	2,755	3,069
	3	Uniform	2,275	3,584	4,418	6,805	16,046
	3	BudgetPV	**997±18**	**1,717±98**	**2,943±138**	**5,875±362**	**13,361±384**
	3	Statistics Oracle	904	2,004	2,399	2,576	2,878

cost of the solver to select subsequent proof attempts. After each attempt is checked by the proof assistant, the resulting evidence is used to update the success estimates and subsequent allocation.

Across four datasets and three progressive solver settings, BudgetPV requires less LLM generation computation than uniform allocation at every reported solving-rate target in both live proving and controlled replay. BudgetPV benefits both from better allocation across unresolved subgoals and from access to lower-cost solvers. With the expensive solver alone, BudgetPV reduces computation by deciding which subgoals should receive additional proof attempts. As medium- and lower-cost solvers are added, it obtains further savings by assigning suitable subgoals to less expensive solvers. The Statistics Oracle further indicates that more accurate success estimation could reduce computation.

Limitations. BudgetPV depends on the accuracy of its success estimates, and the gap to the Statistics Oracle shows that prediction remains an important source of further improvement. The current predictor is trained on Lean Workbook and evaluated on four theorem-proving datasets, so broader training data and additional solver families may improve its generalization. BudgetPV also operates on a given proof sketch and does not revise the decomposition when repeated failures indicate that a subgoal may be difficult because of the chosen sketch. Future work can jointly consider proof-sketch revision and computation allocation, improve success estimation from accumulated verification feedback, and evaluate larger solver sets and broader formal reasoning tasks.

REFERENCES

Zhangir Azerbayev, Bartosz Piotrowski, Hailey Schoelkopf, Edward W. Ayers, Dragomir Radev, and Jeremy Avigad. Proofnet: Autoformalizing and formally proving undergraduate-level mathematics, 2023.

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V. Le, Christopher Ré, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling, 2024.

Mehul Damani, Idan Shenfeld, Andi Peng, Andreea Bobu, and Jacob Andreas. Learning how hard to think: Input-adaptive allocation of lm computation. In *The Thirteenth International Conference on Learning Representations*, 2025.

Leonardo de Moura and Sebastian Ullrich. The lean 4 theorem prover and programming language. In *Automated Deduction – CADE 28*, volume 12699 of *Lecture Notes in Computer Science*, pages 625–635. Springer, 2021. doi: 10.1007/978-3-030-79876-5_37.

Carlo Di Cicco. Code correctness signals in llm hidden states: Pre-generation probing and repair geometry. *arXiv preprint arXiv:2606.14530*, 2026.

Kefan Dong, Arvind Mahankali, and Tengyu Ma. Formal theorem proving by rewarding llms to decompose proofs hierarchically, 2024.

Tom Fawcett. An introduction to roc analysis. *Pattern Recognition Letters*, 27(8):861–874, 2006. doi: 10.1016/j.patrec.2005.10.010.

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. On calibration of modern neural networks. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1321–1330. PMLR, 2017.

Thomas Hubert, Rishi Mehta, Laurent Sartran, Miklós Z. Horváth, Goran Žužić, Eric Wieser, Aja Huang, Julian Schrittwieser, Yannick Schroecker, Hussain Masoom, et al. Olympiad-level formal mathematical reasoning with reinforcement learning. *Nature*, 651:607–613, 2026.

Albert Q. Jiang, Wenda Li, Szymon Tworkowski, Konrad Czechowski, Tomasz Odrzygóźdź, Piotr Miłoś, Yuhuai Wu, and Mateja Jamnik. Thor: Wielding hammers to integrate language models and automated theorem provers. In *Advances in Neural Information Processing Systems*, 2022.

Albert Q. Jiang, Sean Welleck, Jin Peng Zhou, Wenda Li, Jiacheng Liu, Mateja Jamnik, Timothée Lacroix, Yuhuai Wu, and Guillaume Lample. Draft, sketch, and prove: Guiding formal theorem provers with informal proofs. In *International Conference on Learning Representations*, 2023.

Saurav Kadavath, Tom Conerly, Amanda Askell, Tom Henighan, Dawn Drain, Ethan Perez, Nicholas Schiefer, Zac Hatfield-Dodds, Nova DasSarma, Eli Tran-Johnson, Scott Johnston, Sheer El-Showk, Andy Jones, Nelson Elhage, Tristan Hume, Anna Chen, Yuntao Bai, Sam Bowman, Stanislav Fort, Deep Ganguli, Danny Hernandez, Josh Jacobson, Jackson Kernion, Shauna Kravec, Liane Lovitt, Kamal Ndousse, Catherine Olsson, Sam Ringer, Dario Amodei, Tom Brown, Jack Clark, Nicholas Joseph, Ben Mann, Sam McCandlish, Chris Olah, and Jared Kaplan. Language models (mostly) know what they know, 2022.

Guillaume Lample, Marie-Anne Lachaux, Thibaut Lavril, Xavier Martinet, Amaury Hayat, Gabriel Ebner, Aurélien Rodriguez, and Timothée Lacroix. Hypertree proof search for neural theorem proving. In *Advances in Neural Information Processing Systems*, 2022.

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

Yong Lin, Shange Tang, Bohan Lyu, Jiayun Wu, Hongzhou Lin, Kaiyu Yang, Jia Li, Mengzhou Xia, Danqi Chen, Sanjeev Arora, and Chi Jin. Goedel-prover: A frontier model for open-source automated theorem proving, 2025.

Yong Lin, Shange Tang, Bohan Lyu, Ziran Yang, Jui-Hui Chung, Haoyu Zhao, Lai Jiang, Yihan Geng, Jiawei Ge, Jingruo Sun, Jiayun Wu, Jiri Gesi, Ximing Lu, David Acuna, Kaiyu Yang, Hongzhou Lin, Yejin Choi, Danqi Chen, Sanjeev Arora, and Chi Jin. Goedel-prover-v2: Scaling formal theorem proving with scaffolded data synthesis and self-correction. In *International Conference on Learning Representations*, 2026.

Numina and Kimi Team. Kimina-prover: Applying test-time rl search on large formal reasoning models. Hugging Face Blog, 2025.

Lawrence C. Paulson and Jasmin C. Blanchette. Three years of experience with sledgehammer, a practical link between automatic and interactive theorem provers. In *Proceedings of the 8th International Workshop on the Implementation of Logics*, 2010.

Stanislas Polu and Ilya Sutskever. Generative language modeling for automated theorem proving, 2020.

Z. Z. Ren, Zhihong Shao, Junxiao Song, Huajian Xin, Haocheng Wang, Wanjia Zhao, Liyue Zhang, Zhe Fu, Qihao Zhu, Dejian Yang, Z. F. Wu, Zhibin Gou, Shirong Ma, Hongxuan Tang, Yuxuan Liu, Wenjun Gao, Daya Guo, and Chong Ruan. Deepseek-prover-v2: Advancing formal mathematical reasoning via reinforcement learning for subgoal decomposition, 2025.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters, 2024.

George Tsoukalas, Jasper Lee, John Jennings, Jimmy Xin, Michelle Ding, Michael Jennings, Amitayush Thakur, and Swarat Chaudhuri. Putnambench: Evaluating neural theorem-provers on the putnam mathematical competition. In *Advances in Neural Information Processing Systems, Datasets and Benchmarks Track*, volume 37, 2024. doi: 10.52202/079017-0368.

Tanay Varshney, Annie Surla, Michelle Xu, Gomathy Venkata Krishnan, Maximilian Jeblick, David Austin, Neal Vaidya, and Davide Onofrio. Llm router: Rethinking routing with prefill activations. *arXiv preprint arXiv:2603.20895*, 2026.

Haiming Wang, Huajian Xin, Zhengying Liu, Wenda Li, Yinya Huang, Jianqiao Lu, Zhicheng Yang, Jing Tang, Jian Yin, Zhenguo Li, and Xiaodan Liang. Proving theorems recursively. In *Advances in Neural Information Processing Systems*, 2024.

Hanqi Xiao, Vaidehi Patil, Hyunji Lee, Elias Stengel-Eskin, and Mohit Bansal. Generalized correctness models: Learning calibrated and model-agnostic correctness predictors from historical patterns. *arXiv preprint arXiv:2509.24988*, 2025.

Huajian Xin, Daya Guo, Zhihong Shao, Zhizhou Ren, Qihao Zhu, Bo Liu, Chong Ruan, Wenda Li, and Xiaodan Liang. Deepseek-prover: Advancing theorem proving in llms through large-scale synthetic data, 2024a.

Huajian Xin, Z. Z. Ren, Junxiao Song, Zhihong Shao, Wanjia Zhao, Haocheng Wang, Bo Liu, Liyue Zhang, Xuan Lu, Qiushi Du, Wenjun Gao, Qihao Zhu, Dejian Yang, Zhibin Gou, Z. F. Wu, Fuli Luo, and Chong Ruan. Deepseek-prover-v1.5: Harnessing proof assistant feedback for reinforcement learning and monte-carlo tree search, 2024b.

Ran Xin, Zeyu Zheng, Yanchen Nie, Kun Yuan, and Xia Xiao. BFS-Prover-V2: Scaling up multi-turn off-policy rl and multi-agent tree search for llm step-provers. *arXiv preprint arXiv:2509.06493*, 2025.

Kaiyu Yang, Aidan M. Swope, Alex Gu, Rahul Chalamala, Peiyang Song, Shixing Yu, Saad Godil, Ryan Prenger, and Anima Anandkumar. Leandojo: Theorem proving with retrieval-augmented language models. In *Advances in Neural Information Processing Systems*, 2023.

Huaiyuan Ying, Zijian Wu, Yihan Geng, Jiayu Wang, Dahua Lin, and Kai Chen. Lean workbook: A large-scale lean problem set formalized from natural language math problems. In *Advances in Neural Information Processing Systems*, volume 37, 2024. doi: 10.52202/079017-3357.

Kunhao Zheng, Jesse Michael Han, and Stanislas Polu. Minif2f: A cross-system benchmark for formal olympiad-level mathematics. In *International Conference on Learning Representations*, 2022.

Table 3: Generation settings used for the evaluated solvers.

Solver	Temperature / top-p	Max tokens	Context
Goedel-Prover-V2-8B	1.0 / 0.95	40,960	40,960
Goedel-Prover-V2-32B	1.0 / 0.95	40,960	40,960
Kimina-Prover-Distill-8B	0.6 / 0.95	8,096	16,384
Pythagoras-Prover-4B	1.0 / 0.95	30,720	32,768
Pythagoras-Prover-32B	1.0 / 0.95	30,720	32,768
DeepSeek-Prover-V2-7B	1.0 / 0.95	8,192	16,384

Table 4: Solver selection. Close@8 is the percentage of unresolved subgoals closed within eight proof attempts. Generation cost is the mean GPU-seconds per attempt measured on Lean Workbook.

	Close@8 (%)				
Solver	Lean Workbook	miniF2F	ProofNet	PutnamBench	GPU-s/attempt
Pythagoras-Prover-4B	31.6	67.4	18.2	11.1	3.74
Pythagoras-Prover-32B	37.0	70.3	24.6	14.1	8.19
Goedel-Prover-V2-8B	40.8	71.6	23.4	14.3	5.00
Goedel-Prover-V2-32B	43.2	75.8	33.8	17.2	10.44
DeepSeek-Prover-V2-7B	37.2	69.0	26.0	13.3	4.21
Kimina-Prover-Distill-8B	30.2	66.5	20.8	12.2	2.07

A Additional Experimental Details and Ablations

A.1 Solver Generation Settings

Table 3 gives the generation settings used for the solvers in our experiments. We follow the recommended generation settings of each model. Goedel-Prover-V2-32B is served with tensor parallelism of two H100 GPUs, while the 8B models use a single H100 GPU for generation.

Table 5: Computation saving from adding a lower-cost candidate to Goedel-Prover-V2-8B. Values report the reduction in GPU-seconds required to reach the full coverage of Goedel-Prover-V2-8B. Positive values indicate a saving.

Candidate	LWB	miniF2F	ProofNet	PutnamBench
Kimina-Prover-Distill-8B	+10%	+30%	+28%	+13%
Pythagoras-Prover-4B	-1%	-3%	-12%	+1%
DeepSeek-Prover-V2-7B	-24%	-20%	+35%	–

A.2 Solver Selection

To construct the progressive solver sets, we test six candidate solvers with different model sizes and generation costs. For each solver and subgoal, eight proof attempts are generated and checked by the proof assistant. We define Close@8 as the fraction of unresolved subgoals for which at least one of the eight attempts is accepted. Generation cost is measured in GPU-seconds per attempt.

Table 4 shows the test results. On every dataset where it was evaluated, Goedel-Prover-V2-32B achieves the highest Close@8 among the screened solvers. Goedel-Prover-V2-8B retains relatively high solving capability at roughly half of its generation cost on Lean Workbook. They are selected as the expensive and medium-cost solvers in the progressive settings.

The lowest-cost solver is selected based on computation saving rather than Close@8 alone. For each candidate, we combine it with Goedel-Prover-V2-8B and use the measured solver success rates to allocate attempts. We then measure the generation computation required to reach the coverage obtained by Goedel-Prover-V2-8B alone. Table 5 reports the relative saving. Kimina-Prover-Distill-8B reduces computation on all four datasets at the full Goedel-Prover-V2-8B coverage, whereas the

Table 6: Success prediction on Lean Workbook evaluation set in the three-solver setting. Each entry reports AUROC / Brier score / ECE. Higher AUROC and lower Brier score and ECE are better.

Objective	Any solver	Kimina-8B	Goedel-8B	Goedel-32B
Class-weighted CE (Θ_A)	0.782 / 0.215 / 0.155	0.813 / 0.161 / 0.042	0.798 / 0.188 / 0.074	0.778 / 0.221 / 0.169
Plain CE (Θ_B)	0.789 / 0.189 / 0.052	0.822 / 0.157 / 0.040	0.808 / 0.177 / 0.051	0.784 / 0.191 / 0.059

other candidates do not give consistent reductions. Therefore, Kimina-Prover-Distill-8B is used as the third low-cost solver.

A.3 Predictor Training Details

The cross-solver predictors Θ_A and Θ_B use Goedel-Prover-V2-8B as the backbone. We fine-tune the model with LoRA using rank 32, scaling factor 64, and dropout 0.05. LoRA is applied to the q, k, v, o, gate, up, and down projections. Training uses bfloat16, a learning rate of 10^{-5}, batch size 16 with micro-batch size 4, three epochs, and a maximum sequence length of 1,536. Each model is trained on one H100 GPU.

The training set contains 3,488 Lean Workbook subgoals, with 870 additional subgoals reserved for evaluation using a problem-level split. The predictors are trained from the formal subgoal only, without previous proof attempts or proof-assistant feedback. For every training subgoal, the target is the least expensive available solver that closes the subgoal in the recorded eight-attempt proof pools. If no available solver closes it, the target is `none`. In the three-solver setting, the resulting training-label distribution is 30% Kimina-Prover-Distill-8B, 12% Goedel-Prover-V2-8B, 6% Goedel-Prover-V2-32B, and 52% `none`. Θ_A uses class-weighted cross-entropy training loss and Θ_B uses plain cross-entropy. All other model and optimization settings are identical. We train five random seeds for each objective and solver setting.

B Predictor Analysis

B.1 Success Prediction on the Evaluation Set

We first examine whether the predicted success probabilities can distinguish subgoals that a solver can close from those it cannot, and whether the predicted probabilities are well calibrated.

The predictor outputs a distribution over the least expensive solver that can close the subgoal and the `none` class. We convert this distribution into a success probability for each solver using Eq. 1. For example, the success probability of Goedel-Prover-V2-8B includes the probability assigned to both Kimina-Prover-Distill-8B and Goedel-Prover-V2-8B, since either class indicates that Goedel-Prover-V2-8B is sufficient. The probability that at least one solver can close the subgoal is $1 - q_{\mathrm{none}}$.

Table 6 reports AUROC, Brier score, and expected calibration error (ECE) results on the 870 Lean Workbook evaluation subgoals. AUROC measures how well the predictor ranks solvable subgoals above unsolvable ones, with larger values indicating better discrimination. Brier score measures the error of the predicted probabilities, and ECE measures their calibration. For both Brier score and ECE, smaller values are better.

The two objectives have similar AUROC across all four predictions, indicating similar ability to distinguish solvable from unsolvable subgoals. Their main difference is calibration. Plain cross-entropy gives lower Brier score and ECE in every column. For Goedel-Prover-V2-32B, for example, ECE decreases from 0.169 with class-weighted cross-entropy to 0.059 with plain cross-entropy. This better calibration is useful in the later stage of proving, where BudgetPV compares predicted success probabilities across many previously attempted subgoals. The complementary benefit of class weighting for initial solver assignment is examined in the next ablation.

B.2 Effect of Class Weighting on Solver Assignment

The class-weighted and plain cross-entropy predictors make different initial solver assignments. We examine the 394 Lean Workbook evaluation subgoals that can be closed by at least one of the three

Table 7: Initial solver-assignment behavior on 394 subgoals that are closable by at least one solver.

Objective	First choice closes	Over-assigned	Under-assigned
Class-weighted CE (Θ_A)	337 / 394	102	52
Plain CE (Θ_B)	319 / 394	48	72

Table 8: Success prediction during live proving, grouped by the number of previous attempts on the same subgoal. Higher AUROC is better.

Predictor	Previous attempts	Attempts	AUROC
Θ_A	0	12,830	0.820
Θ_A	1	7,886	0.817
Θ_A	2–3	2,561	0.817
Θ_B	0	3,504	0.827
Θ_B	1	7,857	0.782
Θ_B	2–3	35,079	0.707

solvers. An assignment is under-assigned when the selected solver cannot close the subgoal but a more expensive solver can, and over-assigned when a more expensive solver is selected although a less expensive solver can close it.

Class weighting reduces under-assignment from 72 to 52 and increases the number of subgoals closed by the first solver choice from 319 to 337. This is useful in the initial stage, where assigning a solver that is too weak causes an additional failed attempt before a stronger solver is used. Plain cross-entropy makes fewer over-assignments and provides better calibrated probabilities, which is more useful later when BudgetPV compares previously attempted subgoals using their absolute success estimates.

B.3 Success Prediction During Live Proving

In live proving, BudgetPV continues to estimate success after a subgoal has received failed proof attempts. We evaluate whether the predictor can still distinguish successful from unsuccessful attempts as the number of previous attempts increases. Table 8 reports AUROC on decisions collected during the live experiments, grouped by the number of previous attempts on the same subgoal.

For Θ_A, AUROC remains approximately 0.82 from the first attempt through two to three previous attempts. Θ_B also retains useful discrimination after previous failures, with AUROC of 0.782 after one previous attempt and 0.707 after two to three attempts. These results show that the success estimates remain informative as BudgetPV revisits previously attempted subgoals.