RETHINKING SUPERVISION GRANULARITY: SEGMENT-LEVEL LEARNING FOR LLM-BASED THEOREM PROVING

Anonymous authors
Paper under double-blind review

ABSTRACT

LLM-based automated theorem proving in Lean 4 commonly uses step-level tactic prediction with tree search or whole-proof generation. These two paradigms represent opposite granularities for constructing supervised training data: the former provides dense local signals but may fragment coherent proof processes, while the latter preserves global structure but requires complex end-to-end generation. In this paper, we revisit supervision granularity as a training-set construction problem and propose *segment-level supervision*, which extracts locally coherent proof segments to train policy models. We further reuse the same strategy at inference time to trigger short rollouts for existing step-level models. When trained with segment-level supervision on STP, LeanWorkbook, and NuminaMath-LEAN, the resulting policy models achieve proof success rates of 64.84%, 60.90%, and 66.31% on miniF2F and 32.13%, 21.73%, and 17.20% on FATE-M, respectively, consistently outperforming both step-level and whole-proof baselines. Goal-aware rollout further improves existing step-level provers while reducing inference costs: on miniF2F, it increases the proof success rate of BFS-Prover-V2-7B from 68.77% to 70.74% and that of InternLM2.5-StepProver from 59.59% to 60.33%; on FATE-M, the corresponding success rates increase from 45.60% to 50.67% and from 33.47% to 36.13%, showing that appropriate supervision granularity better aligns model learning with proof structure and search. Code and models are available at `https://anonymous.4open.science/r/SEG-ATP`.

1 INTRODUCTION

Improving the reasoning capabilities of large language models (LLMs) has become a central direction in artificial intelligence research (Shao et al., 2024; Singh et al., 2025; Yang et al., 2025). Among various reasoning tasks, mathematical reasoning has attracted substantial attention from both academia and industry, as it requires rigor, multi-step derivation, and generalization (Dekoninck et al., 2026; Luong et al., 2025). Compared with studies that focus primarily on solving natural-language mathematical or logical problems, formal theorem proving provides a stricter and more reliable research setting (Lin et al., 2025b; Hubert et al., 2025).

Lean 4 is a widely used formal system for recent LLM-based automated theorem proving (ATP) (Moura & Ullrich, 2021). Existing LLM-based provers are typically fine-tuned on proof scripts under two paradigms: tree-search-based *step-level* proving and *whole-proof generation* (Ren et al., 2025). Step-level methods extract state–tactic pairs to predict the next tactic during search, providing dense local supervision but potentially fragmenting coherent proof progressions into atomic decisions (Xin et al., 2025a; Wu et al., 2024). Whole-proof methods train the model to generate the complete proof script from the initial proof state, preserving global structure but requiring long-horizon generation and repeated sampling attempts in inference (Xin et al., 2024b; Lin et al., 2025a). We offer a new perspective on step-level and whole-proof supervision by viewing them as two extremes of supervision granularity. As illustrated in Figure 1, our key observation is that the proofs are typically neither merely sequences of independent atomic tactics nor always suitable for one-shot whole-proof generation. Many proof processes contain locally coherent tactic segments: several consecutive tactics often jointly perform the expansion, transformation, or resolution of a subgoal. From this perspective, the intrinsic organization of a proof is better viewed as a hierarchical process driven by

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.

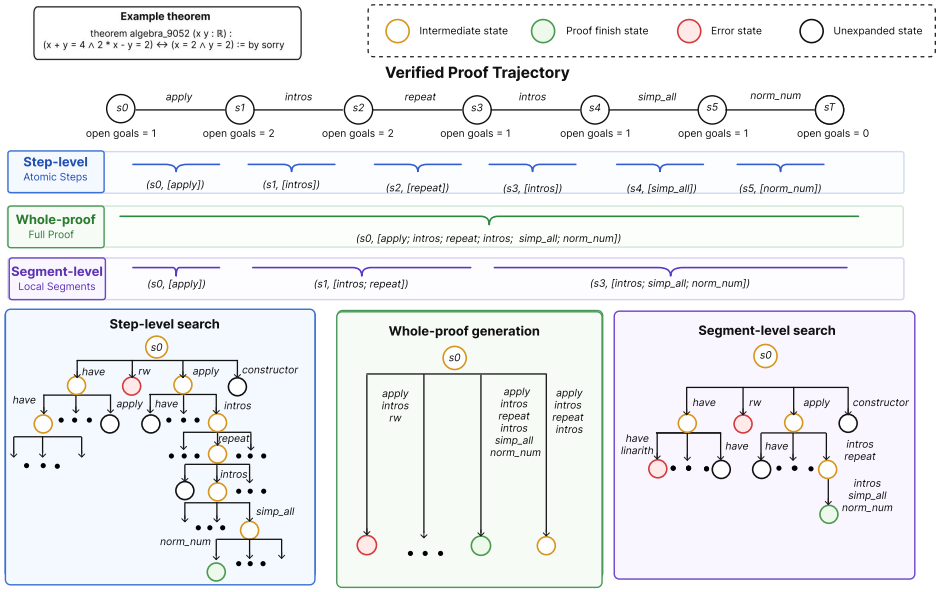


Figure 1: Illustration of supervision granularity on the problem *algebra_9052* from NuminaMath-LEAN. For simplicity, proof states show only tactic transitions and open-goal counts, omitting premises. Tactic labels are abbreviated, and `by sorry` in the theorem box is a placeholder for the omitted proof. The same verified proof trajectory is organized into step-level, whole-proof, and segment-level supervision targets, together with their corresponding inference patterns.

the repeated expansion, switching, and resolution of subgoals, while the linear proof script is only a serialized representation of this underlying structure. Prior work has already exploited subgoal structure in proof generation, hierarchical planning, and reasoning search (Cao et al., 2025; Jiang et al., 2022). This naturally raises the question of whether such structure should also affect the basic proof units used for model learning: *for ATP, what supervision granularity is more suitable for learning and searching with a policy model?*

In this paper, we formulate supervision granularity as a boundary-selection problem over proof-state trajectories. This lets us revisit supervision and search decision units in Lean 4 ATP without new model architectures or theorem–proof pairs. We instantiate this framework with *segment-level supervision*, which uses locally coherent proof segments as the basic learning units. Specifically, we use changes in the number of open goals as a lightweight structural signal to identify approximate boundaries in the proof process. During training, verified proof scripts are reorganized into local proof segments rather than single tactics or complete proofs. At inference time, the same signal can trigger short rollouts for step-level policies, concentrating search around meaningful structural transitions. Experiments on STP (Dong & Ma, 2025), LeanWorkbook (Ying et al., 2024), and NuminaMath-LEAN (Wang et al., 2025) show that segment-level policies fine-tuned from Qwen2.5-Math-7B (Yang et al., 2024) consistently outperform step-level and whole-proof baselines on miniF2F (Zheng et al., 2021) and FATE-M (Jiang et al., 2025). Applying the same open-goal signal to inference-time rollout for existing step-level provers, including BFS-Prover-V2-7B (Xin et al., 2025b) and InternLM2.5-StepProver (Wu et al., 2024), also improves proof success rates while reducing search costs. These results suggest that supervision granularity is an important design axis for better exploiting existing proof data and improving the interaction between policy learning and proof search. Our contributions are:

- We formulate supervision granularity in Lean 4 ATP as a boundary-selection problem over proof-state trajectories, showing that step-level and whole-proof supervision represent two extremes that miss intermediate local proof structures.
- We propose *segment-level supervision* based on open-goal changes, using this signal to construct locally coherent training segments and trigger short rollouts at inference time.
- Experiments on multiple Lean 4 datasets and benchmarks show that segment-level supervision improves proof success rates over step-level and whole-proof baselines, while goal-aware rollout further improves existing step-level provers and reduces search cost.

2 PRELIMINARIES

From raw proof corpora to supervision extraction. Formal theorem proving has stricter training-data requirements than natural-language mathematical reasoning. Supervised samples require a theorem statement and a proof-assistant-checkable script. Such scripts encode formal syntax, library usage, tactic patterns, and executable proof-state transitions. Because high-quality formal proofs require mathematical and proof-assistant expertise, Lean 4 corpora remain much smaller than general language or code corpora, limiting LLM performance in formal proving. Existing work addresses this data scarcity problem from two main directions: 1) expanding the corpora through autoformalization (Chen et al., 2025a; Cabral et al., 2025), proof generation (Xin et al., 2024a), or self-play (Dong & Ma, 2025), and 2) extracting additional supervision signals from existing formal objects, such as proof terms, proof states, or intermediate reasoning traces (Wu et al., 2024; Li et al., 2024).

Complementary to these directions, this paper studies how to organize existing raw proof trajectories into training data for policy models. The same proof trajectory can be decomposed into single-tactic examples, treated as a complete proof script, or reorganized into local proof segments between these two extremes. These choices determine the basic proof units learned during fine-tuning and further affect the decision and expansion behavior of search at inference time.

Proof trajectories and supervision granularities. Let the original Lean 4 proof corpus be $\mathcal{D} = \{(x^{(i)}, \pi^{(i)})\}_{i=1}^{N}$, where $x^{(i)}$ denotes the i-th formal theorem, and $\pi^{(i)}$ denotes the Lean-verifiable proof script, which can be defined as follows

$$\pi^{(i)} = (a_1^{(i)}, a_2^{(i)}, \ldots, a_{T_i}^{(i)}), \tag{1}$$

where $a_t^{(i)}$ is the t-th tactic and T_i is the proof length. For each formal theorem, we can replay the tactic sequence step by step and recover the corresponding proof-state trajectory

$$\tau = (s_0, a_1, s_1, \ldots, a_T, s_T), \tag{2}$$

where s_0 is the initial proof state of the theorem, s_T is the terminal state in which all goals have been closed, and each s_i is recovered from s_{i-1} based on the tactic a_i. Then, the proof corpus $\mathcal{D}$ could be transformed into a proof-state trajectory set $\mathcal{T} = \{\tau^{(i)} | i = 1, \ldots, N\}$. The essential difference among supervision granularities lies in *how training data is extracted from the trajectory set* $\mathcal{T}$.

Existing Lean 4 ATP methods typically correspond to two extreme supervision granularities. The first is *step-level supervision*, which takes the proof state before each tactic execution as input and uses the next tactic as the target:

$$\mathcal{D}_{\text{step}} = \bigcup_{i=1}^{N} \left\{ \left(s_{t-1}^{(i)}, a_t^{(i)} \right) \;\middle|\; t = 1, \ldots, T_i \right\}. \tag{3}$$

This approach provides dense local supervision signals. At inference time, a step-level policy is typically combined with tree search: it generates candidate tactics at each proof state, which are then executed by Lean to continue the expansion.

The second is *whole-proof supervision*, which directly takes the initial state or theorem statement as input and uses the complete tactic sequence as the target:

$$\mathcal{D}_{\text{whole}} = \left\{ \left(s_0^{(i)}, \pi^{(i)} \right) \;\middle|\; i = 1, \ldots, N \right\}. \tag{4}$$

This approach preserves the complete proof script as the training target. At inference time, the model usually generates the entire proof script in one shot.

Goal structure in proof trajectories. Step-level and whole-proof supervision represent two extremes of training-data extraction. We study the proof units between them by focusing on goal structure in proof trajectories. A Lean 4 proof state typically contains several unresolved open goals. Each open goal corresponds to a local proof obligation that remains to be proved, together with its associated context, hypotheses, and target expression. Executing a tactic transforms the current proof state into a new state and may introduce new subgoals, close existing goals, or transform the target expression and proof context while keeping the number of goals unchanged. Let the set of open goals in a proof state s be $\mathcal{G}(s)$. The number of open goals is defined as

$$g(s) := |\mathcal{G}(s)|. \tag{5}$$

When $g(s_t) > g(s_{t-1})$, the current proof obligation is typically expanded into more subgoals. When $g(s_t) < g(s_{t-1})$, some goals are closed. Otherwise, the proof may still make local progress under the same frontier size. Although this signal cannot fully characterize all semantic changes in proof states, it provides a simple, stable, and computable basis for identifying boundaries among proof stages, motivating the proposed segment-level supervision for training an LLM-based ATP model.

3 METHOD

We first formulate supervision granularity as boundary selection, then describe goal-change-based segment-level training and goal-aware rollout.

3.1 BOUNDARY VIEW OF SUPERVISION GRANULARITY

We formalize supervision granularity as boundary selection over the proof-state trajectory τ (Equation 2). Adjacent boundaries determine each training example's input state and target tactic sequence; different selections yield different supervised datasets from the same verified trajectory.

Boundary placement. Given a trajectory τ, a boundary strategy Φ selects a strictly increasing sequence of boundary positions from the atomic positions $\{0, 1, \ldots, T\}$:

$$\mathcal{B}_\Phi(\tau) = \{0 = t_0 < t_1 < \cdots < t_K = T\}. \tag{6}$$

The initial position 0 and the terminal position T are always included as boundaries. For two adjacent boundaries t_{k-1} and t_k, we define the tactic subsequence between them as

$$m_k := a_{t_{k-1}+1:t_k} = (a_{t_{k-1}+1}, \ldots, a_{t_k}), \qquad k = 1, \ldots, K. \tag{7}$$

This subsequence can be viewed as a macro action: starting from the boundary state $s_{t_{k-1}}$, it executes a sequence of tactics and reaches the next boundary state s_{t_k}. Accordingly, the boundary strategy Φ converts a single trajectory τ into a set of supervised training examples:

$$\mathcal{Z}_\Phi(\tau) = \left\{ (s_{t_{k-1}}, m_k) \;\middle|\; k = 1, \ldots, K \right\}. \tag{8}$$

For the set of verified trajectories $\mathcal{T}$ corresponding to the entire training corpus, we obtain

$$\mathcal{D}_\Phi = \bigcup_{\tau \in \mathcal{T}} \mathcal{Z}_\Phi(\tau). \tag{9}$$

Thus, the boundary strategy determines the basic learning unit: a single tactic, an entire proof, or an intermediate proof segment.

Existing regimes are special cases. The formulation of supervision granularity unifies supervision regimes. For step-level supervision, all positions are selected as boundaries:

$$\mathcal{B}_{\text{step}}(\tau) = \{0, 1, \ldots, T\}. \tag{10}$$

In this case, each macro action contains only one tactic, and the model learns to generate the atomic tactic based on the training data extracted via the boundary strategy $\mathcal{B}_{\text{step}}$. For whole-proof supervision, only the initial and terminal positions are retained in the strategy:

$$\mathcal{B}_{\text{whole}}(\tau) = \{0, T\}. \tag{11}$$

With $\mathcal{B}_{\text{whole}}$, the entire proof script is one macro action from the initial to the terminal state.

Beyond these two special cases, a boundary strategy can be instantiated using criteria such as token length, proof-state similarity, or structural changes in proof states. In this paper, we use changes in the number of open goals as a lightweight and directly computable structural signal, yielding an intermediate granularity between step-level and whole-proof supervision.

3.2 GOAL-CHANGE-BASED SEGMENT-LEVEL TRAINING

Following the open-goal count $g(s)$ defined in Equation 5, we place segment boundaries at proof states reached after changes in $g(s)$. Such changes often indicate structural transitions, such as subgoal creation or goal closure, marking the end of one local proof segment and the start of the next.

Given a trajectory τ, suppose executing tactic a_t changes the number of open goals, i.e., $g(s_t) \neq g(s_{t-1})$. This indicates that a_t changes the size of the current proof frontier. We therefore select the successor position t, corresponding to the changed proof state s_t, as a boundary. Formally, our goal-change boundary rule is defined as the ordered set

$$\begin{aligned}\mathcal{B}_{\text{seg}}(\tau) &:= (t_0, t_1, \ldots, t_K) \\ &= \text{sort}\left(\{0, T\} \cup \{t \mid 1 \leq t \leq T,\ g(s_t) \neq g(s_{t-1})\}\right), \quad 0 = t_0 < t_1 < \cdots < t_K = T.\end{aligned} \tag{12}$$

Then, the tactic subsequence in τ between two adjacent boundaries forms a segment-level action:

$$m_k = a_{t_{k-1}+1:t_k}, \qquad k = 1, \ldots, K. \tag{13}$$

Thus, the segment-level training examples induced by a trajectory τ are

$$\mathcal{Z}_{\text{seg}}(\tau) = \left\{(s_{t_{k-1}}, m_k) \;\middle|\; k = 1, \ldots, K\right\}, \tag{14}$$

and the full training set is

$$\mathcal{D}_{\text{seg}} = \bigcup_{\tau \in \mathcal{T}} \mathcal{Z}_{\text{seg}}(\tau). \tag{15}$$

Under this construction, each training example takes a boundary proof state as input and uses the subsequent local proof progression as the target. Since boundaries are placed at proof states reached after goal-changing tactics, each segment captures local proof progress until the next structural change. The next segment then starts from the changed proof state and continues under the new goal structure. Thus, the model learns goal-structured proof segments rather than isolated next tactics.

Training objective and optimization view. We train the segment-level policy with the standard autoregressive maximum-likelihood objective. Given a training example $(s, m) \in \mathcal{D}_{\text{seg}}$, where $m = (a_1, \ldots, a_L)$ is a tactic sequence, we serialize and concatenate the tactics into

$$y(m) = \text{Tok}\left(\text{ser}(a_1) \| \cdots \| \text{ser}(a_L)\right) = (y_1, \ldots, y_{R(m)}), \tag{16}$$

where $\text{ser}(\cdot)$ denotes tactic serialization, $\|$ denotes string concatenation with tactic separators, $\text{Tok}(\cdot)$ denotes tokenization, and $R(m) = |y(m)|$. The model factorizes

$$p_\theta(y(m) \mid s) = \prod_{r=1}^{R(m)} p_\theta(y_r \mid s, y_{<r}), \qquad \mathcal{L}_{\text{seg}}(\theta) = -\sum_{(s,m) \in \mathcal{D}_{\text{seg}}} \log p_\theta(y(m) \mid s). \tag{17}$$

This procedure does not change the model architecture or require additional theorem/proof generation. It only reorganizes existing verified proof-state trajectories, turning atomic tactic prediction into goal-structured segment prediction.

To interpret how boundary strategies change the learning problem, consider a general strategy Φ and its induced dataset $\mathcal{D}_\Phi$. For $(s, m) \in \mathcal{D}_\Phi$, define the token-normalized negative log-likelihood as

$$\ell_\theta(s, m) = \frac{1}{R(m)} \sum_{r=1}^{R(m)} -\log p_\theta(y_r(m) \mid s, y_{<r}(m)). \tag{18}$$

Let $P_\Phi(n)$ be the empirical target-length distribution and $\bar{\ell}_\Phi(n; \theta)$ be the average normalized loss conditioned on length n. Then

$$\begin{aligned}P_\Phi(n) &= \Pr_{(s,m) \sim \mathcal{D}_\Phi}\left[R(m) = n\right], \\ \bar{\ell}_\Phi(n; \theta) &= \mathbb{E}_{(s,m) \sim \mathcal{D}_\Phi}\left[\ell_\theta(s, m) \,|\, R(m) = n\right], \\ \bar{\mathcal{L}}_\Phi(\theta) &= \mathbb{E}_{(s,m) \sim \mathcal{D}_\Phi}\left[\ell_\theta(s, m)\right] = \sum_n P_\Phi(n)\, \bar{\ell}_\Phi(n; \theta).\end{aligned} \tag{19}$$

This decomposition suggests that supervision granularity affects learning by jointly reshaping the distribution of target lengths and the conditional difficulty of predicting targets at each length. Step-level supervision provides dense but highly local signals, where many targets capture short tactic continuations without explicitly modeling larger proof transitions. Whole-proof supervision preserves global proof structure, but imposes long-range autoregressive dependencies that are difficult to optimize from a single initial state. Segment-level supervision lies between these two extremes: it retains locally coherent multi-step proof progress while avoiding the optimization difficulty of whole-proof generation. This suggests that aligning learning units with local proof structure matters beyond target length alone. Model-performance and training-loss results (Section 4.2) support this view for both optimization and search.

3.3 GOAL-AWARE ROLLOUT FOR STEP-LEVEL INFERENCE

The same open-goal signal also guides existing step-level models at inference without fine-tuning. Given a step-level prover, it still generates one tactic at a time, but the search does not need to resume full explicit expansion after every atomic tactic. When a tactic triggers a change in the number of open goals, we let the model continue in the current local direction for a short rollout, thereby temporarily compressing several local steps into a coarser-grained search transition. In detail, let the step-level policy be $p_\theta(a \mid s)$. In standard step-level search, the model generates a candidate tactic a at the current proof state s, which is then executed by Lean:

$$\mathrm{Exec}(s, a) \in \{\bot, s'\}, \tag{20}$$

where $\bot$ denotes execution failure, and s' denotes the successor proof state after successful execution. If s' does not complete the proof, standard search immediately adds it to the search frontier as a node for subsequent explicit expansion.

We introduce *goal-aware rollout* for transitions that change the proof frontier. Given a valid transition $s \xrightarrow{a} s'$, if $g(s') \neq g(s)$ and s' does not complete the proof, then a changes the open-goal count, potentially signaling subgoal expansion, goal closure, or another structural transition. Instead of immediately resuming full search at s', we perform a linear rollout from s' for at most H steps.

Specifically, let $u_0 = s'$. At the h-th rollout step, the model generates a tactic from the current state u_{h-1}:

$$\tilde{a}_h \sim p_\theta(\cdot \mid u_{h-1}), \qquad h = 1, \ldots, H, \tag{21}$$

which is then executed by Lean to obtain the successor proof state u_h. The rollout terminates early if execution fails or the proof is completed; otherwise, it stops after H steps. Let the actually executed rollout sequence be

$$\rho = (\tilde{a}_1, \ldots, \tilde{a}_\ell), \qquad 0 \leq \ell \leq H, \tag{22}$$

and let the final state be $\bar{s} = u_\ell$. When $\ell = 0$, we have $\rho = \emptyset$ and $\bar{s} = s'$. Using $\oplus$ to denote tactic-sequence concatenation and (a) to denote the singleton sequence containing tactic a, the resulting search transition could be written as $s \xrightarrow{(a)\oplus\rho} \bar{s}$. Thus, goal-aware rollout modifies a standard single-step transition as follows:

$$\mathrm{GAR}_H(s, a) = \begin{cases} \bot, & \mathrm{Exec}(s, a) = \bot, \\ (s', (a)), & \mathrm{Exec}(s, a) = s' \text{ and } \big(g(s') = g(s) \text{ or } \mathrm{Solved}(s')\big), \\ (\bar{s}, (a) \oplus \rho), & \mathrm{Exec}(s, a) = s' \text{ and } g(s') \neq g(s) \text{ and } \neg\mathrm{Solved}(s'). \end{cases} \tag{23}$$

When a candidate tactic does not change the number of open goals, search behaves exactly as in standard step-level search. When an open-goal count changes without completing the proof, intermediate states inside the rollout are not treated as explicit branching points. If the rollout completes the proof, search returns the verified tactic sequence; otherwise, only the final valid rollout state is added to the search frontier. Algorithm 1 in Appendix A.6 gives best-first-search pseudocode with goal-change triggers, early termination, and valid-prefix retention on failure.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

We evaluate two components of our method: goal-change-based segment-level training and goal-aware rollout for step-level inference. The first set of experiments compares different supervision granularities under matched training settings and controlled evaluation protocols. The second evaluates whether the open-goal signal can improve existing step-level provers without retraining.

For segment-level training, we conduct experiments on three Lean 4 proof datasets: STP, Lean-Workbook and NuminaMath-LEAN. For each data source, we construct step-level, whole-proof, and segment-level supervision from the same verified proof trajectories, then fine-tune Qwen2.5-Math-7B policy models to compare their downstream proving performance. The rules naturally yield different example counts and target-length distributions, while the base model and optimization setup remain fixed. Concretely, all three models use Adam without weight decay, a global batch size of 2048, an initial learning rate of 2×10^{-5}, three training epochs, a warmup ratio of 0.05, and cosine decay

Table 1: Proof success rates (%, mean ± std over five independent runs) by supervision granularity and evaluation protocol. Bold marks row maxima; shading marks our method.

Training data	Eval set	Step-level	Whole-proof	Whole-proof-seg	Segment-level (ours)
STP	miniF2F	63.11 ± 1.19	61.64 ± 0.62	63.52 ± 0.29	**64.84 ± 0.53**
LeanWorkbook	miniF2F	59.02 ± 0.82	54.67 ± 0.85	54.51 ± 1.30	**60.90 ± 0.62**
NuminaMath-LEAN	miniF2F	63.11 ± 0.77	55.90 ± 1.07	52.70 ± 1.03	**66.31 ± 0.79**
STP	FATE-M	30.27 ± 1.30	31.07 ± 0.89	30.53 ± 2.42	**32.13 ± 0.56**
LeanWorkbook	FATE-M	20.80 ± 0.99	19.47 ± 0.56	16.40 ± 1.61	**21.73 ± 0.60**
NuminaMath-LEAN	FATE-M	15.73 ± 1.01	16.13 ± 0.56	15.47 ± 0.99	**17.20 ± 1.10**

Table 2: Average token cost / time cost (s) on the common-solved subset. Bold marks each row's minimum for each metric; shading marks our method.

Training data	Eval set	Step-level	Whole-proof	Whole-proof-seg	Segment-level (ours)
STP	miniF2F	809.31 / 14.54	3198.50 / 18.00	**457.82 / 6.82**	543.98 / 11.88
LeanWorkbook	miniF2F	**228.64 / 3.87**	2554.67 / 14.15	611.07 / 11.24	446.36 / 6.84
NuminaMath-LEAN	miniF2F	2141.80 / 21.21	4147.67 / 23.00	1199.51 / 15.13	**1149.80 / 10.55**
STP	FATE-M	2161.74 / 29.69	4709.70 / 37.54	1349.97 / 16.73	**1001.17 / 13.70**
LeanWorkbook	FATE-M	**366.17** / 19.94	6603.49 / 127.95	1287.80 / 36.33	608.80 / **14.11**
NuminaMath-LEAN	FATE-M	1685.47 / 22.47	11525.93 / 67.96	2031.47 / 20.92	**1369.07 / 12.05**

to 10% of the initial learning rate; complete training details are provided in Appendix A.1. Each model is evaluated on the same two standard benchmarks for formal theorem proving, miniF2F and FATE-M. We also evaluate the whole-proof policy with segment-style search (Whole-proof-seg), isolating the training objective's effect under a shared search budget. For goal-aware rollout, we apply the method to two existing step-level provers, BFS-Prover-V2-7B and InternLM2.5-StepProver without retraining. We compare standard best-first search against best-first search incorporating goal-aware rollout on miniF2F and FATE-M separately under the same search budget.

We report **proof success rate** as the accuracy metric, averaged over five inference runs using the same fixed checkpoint for each model and reported as mean ± std. To compare inference efficiency, we report *token cost*, defined as the total number of output tokens generated during inference, and *time cost*, defined as the total prover time including both model generation and Lean execution. For each evaluation setting, we use a fixed set of theorems solved by every compared method in all five runs. Token and time costs are averaged over this set and the five runs. Due to space limitations, detailed experimental settings are provided in Appendix A.1.

4.2 Main results and discussion

Segment-level training. Tables 1 and 2 report proof success rates and inference costs across supervision granularities. Table 1 shows that Segment-level has the highest proof success rate for all three training sources on both miniF2F and FATE-M. Compared with Step-level under the same tree-search budget, Segment-level proves more theorems on average in all six training-source/benchmark settings, each requiring a valid path through Lean 4's large proof space. The consistent gains across training sources and benchmarks therefore matter beyond their numerical size. Whole-proof-seg underperforms Segment-level in all settings, suggesting that segment-style search alone does not explain the gains; the training objective also needs to match the segment-style inference interface.

For efficiency, Table 2 reports token and time costs on the common-solved subset within each evaluation setting. Segment-level achieves the lowest token and time costs on miniF2F with NuminaMath-LEAN training, and on FATE-M with STP or NuminaMath-LEAN training. With LeanWorkbook training, Step-level has the lowest cost on miniF2F and the lowest token cost on FATE-M, while Segment-level has the lowest time cost. With STP training on miniF2F, Whole-proof-seg achieves the lowest inference cost, but its proof success rate remains lower than that of Segment-level.

Overall, these results suggest that proof segments constructed from open-goal changes provide an effective intermediate granularity between single-step tactic prediction and whole-proof generation. Efficiency gains are more pronounced for proofs with multi-step local progress, whereas the many short proofs in LeanWorkbook leave less room for search compression. Together with the consistent accuracy gains in Table 1, Segment-level provides a strong balance between proof success rate

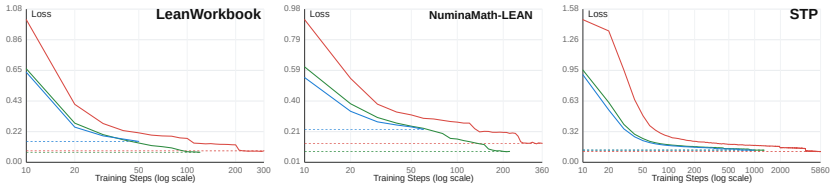


Figure 2: Training loss by supervision granularity on LeanWorkbook, NuminaMath-LEAN, and STP. Red: Step-level; green: Segment-level (ours); blue: Whole-proof. Dashed lines mark final losses.

Table 3: Results of goal-aware rollout on existing step-level provers. The rollout horizon H is fixed to 5. Accuracy is computed on the full evaluation set, while Tok. and Time are averaged on the common-solved subset. Shaded rows indicate goal-aware rollout.

Eval set	Step-level prover	Search	Acc. (%) ↑	Tok. ↓	Time (s) ↓
miniF2F	BFS-Prover-V2-7B	Best-first	68.77 ± 0.79	1664.39	35.37
miniF2F	BFS-Prover-V2-7B	+ Goal-aware rollout	**70.74 ± 1.18**	**1122.43**	**32.53**
miniF2F	InternLM2.5-StepProver	Best-first	59.59 ± 0.37	1325.49	21.60
miniF2F	InternLM2.5-StepProver	+ Goal-aware rollout	**60.33 ± 0.45**	**701.87**	**12.32**
FATE-M	BFS-Prover-V2-7B	Best-first	45.60 ± 1.30	2196.18	32.80
FATE-M	BFS-Prover-V2-7B	+ Goal-aware rollout	**50.67 ± 0.82**	**1646.97**	**32.03**
FATE-M	InternLM2.5-StepProver	Best-first	33.47 ± 2.08	1681.08	28.34
FATE-M	InternLM2.5-StepProver	+ Goal-aware rollout	**36.13 ± 0.73**	**1521.08**	**28.12**

and inference efficiency. Further analysis of these boundary cases is provided in Appendix A.4. Appendix A.3 reports additional Qwen3-14B (Yang et al., 2025) results.

Goal-aware rollout. Table 3 reports the inference-only results of applying goal-aware rollout to existing step-level provers. Without retraining, goal-aware rollout improves proof success rates for both step-level provers while reducing token and time costs on the common-solved subset. On miniF2F, goal-aware rollout raises proof success from 68.77% to 70.74% for BFS-Prover-V2-7B and from 59.59% to 60.33% for InternLM2.5-StepProver. On FATE-M, the corresponding gains are 45.60% to 50.67% and 33.47% to 36.13%. Token and time costs decrease in all four settings, as detailed in Table 3.

These results extend the use of open-goal changes beyond training: Segment-level changes the supervision target, whereas goal-aware rollout keeps the step-level policy fixed and changes only inference. The same signal can therefore improve supervision and search without new model architectures or proof data. The sensitivity analysis of the rollout horizon H is provided in Appendix A.2.2, where we compare different rollout horizons under the same inference budget.

Training loss analysis. Figure 2 empirically examines the optimization view discussed above. Whole-proof supervision generally ends with higher loss and decreases more slowly on LeanWorkbook and NuminaMath-LEAN, consistent with its long-horizon targets. Step-level supervision decreases faster in early training, reflecting its shorter and more local prediction targets.

Segment-level supervision shows a different pattern. On LeanWorkbook and NuminaMath-LEAN, despite using longer targets than step-level supervision, its loss decreases steadily and eventually reaches a comparable or lower level. On STP, the three curves are close overall, with Segment-level ending below Whole-proof and only slightly above Step-level. These observations suggest that target length alone does not determine optimization behavior: open-goal-change boundaries can produce multi-step targets that remain relatively well conditioned on the current proof state.

4.3 Ablation Study on Boundary Selection

We further study how boundary selection affects segment-level supervision by comparing our open-goal-change rule with several simple alternatives under the same training and evaluation setup. Specifically, we consider token-based segmentation, tactic-distance-based segmentation, and state-distance-based segmentation. For each alternative strategy, we evaluate three hyperparameter settings, with detailed per-setting results provided in Appendix A.2.1.

All models in this ablation are trained on NuminaMath-LEAN and evaluated on miniF2F. As shown in Table 4 in Appendix A.2.1, the open-goal-change strategy achieves the best performance, reaching 66.31%. Among the alternative strategies, tactic-distance-based segmentation achieves the strongest

results, but remains substantially below open-goal change. Token-based segmentation is relatively stable but does not yield strong proving performance, while state-distance-based segmentation is more sensitive to the threshold choice and generally performs worse.

These results indicate that boundary selection has a direct impact on the quality of segment-level supervision. Simply grouping tactics by length or textual distance does not necessarily produce useful proof segments. In contrast, open-goal changes provide a lightweight structural signal that is better aligned with useful proof boundaries in the settings studied here.

5 Related Work

Automated theorem proving. Automated theorem proving has long been dominated by symbolic methods (Pastre, 1993; Schürmann & Pfenning, 1998), while large language models enable a generation-and-verification paradigm with proof assistants. GPT-f (Polu & Sutskever, 2020) is an early example, using a language model to propose proof steps, tree search to explore proof paths, and a proof assistant to verify each step. This paradigm has led to many step-level tree-search provers. InternLM2.5-StepProver (Wu et al., 2024) uses expert iteration and critic-guided search, while BFS-Prover-V2 (Xin et al., 2025b) improves best-first search with reinforcement learning and stronger test-time search. In contrast, whole-proof generation methods directly produce complete Lean proof scripts from theorem statements or initial proof states. Systems include DeepSeek-Prover-V1.5 (Xin et al., 2024b) and Goedel-Prover-V2 (Lin et al., 2025b), which improve whole-proof proving through proof-assistant feedback, data generation, proof correction, and iterative training.

Formal proof data construction. The scarcity of high-quality formal proof data remains a central bottleneck for LLM-based automated theorem proving. Many works therefore expand the formal data available for training. DeepSeek-Prover translates natural-language problems and solutions into Lean statements, then generates and verifies proofs for large-scale training (Xin et al., 2024a). Expert iteration adds newly discovered successful proofs from a given problem set back into training (Polu et al., 2022). STP uses self-play: the model generates conjectures, and the prover attempts to prove them, producing data tailored to the current prover (Dong & Ma, 2025). Beyond data scaling, some works extract supervision from existing formal proofs. PACT builds auxiliary prediction tasks from Lean-kernel-checked proof terms, encouraging the model to learn structural information inside proof objects (Han et al., 2021). Lean-STaR inserts natural-language thoughts, retrospectively constructed from correct proofs, before tactics, allowing the model to learn both intermediate reasoning and formal actions (Lin et al., 2024). These works show that existing formal proofs contain not only final proof scripts, but also structural and intermediate supervision signals.

Goal-structured proving. Formal proofs are naturally organized around goal structure: goals are transformed, new subgoals are created, and all goals must eventually be closed (Moura & Ullrich, 2021). Many works explicitly exploit this structure to organize proof construction. DeepSeek-Prover-V2 decomposes complex problems into smaller subgoals, proves them, and combines the results into a complete proof (Ren et al., 2025). Seed-Prover generates auxiliary lemmas and proof sketches to split the main proof into relatively independent tasks (Chen et al., 2025b). Hilbert recursively decomposes a goal when direct proving fails and constructs subproofs with informal reasoning and formal verification (Varambally et al., 2025). These works show that goal structure provides an effective intermediate organization for model generation and proof search. Unlike methods that generate subgoals, lemmas, or plans, we use proof-assistant open-goal count changes as a lightweight signal to study supervision granularity in verified trajectories.

6 Conclusions

This paper studies supervision granularity for LLM-based automated theorem proving. We formulate supervision construction as boundary selection over verified proof trajectories and propose segment-level supervision based on open-goal count changes to model local proof progressions. Experiments show that segment-level supervision improves proof success across training sources, while goal-aware rollout improves existing step-level provers and reduces search cost. Training-loss analysis suggests that open-goal boundaries yield coherent multi-step supervision, preserving proof-level structure while remaining optimization-friendly.

AI USE STATEMENT

Generative AI tools were used to assist with language polishing, LaTeX formatting and consistency checks, and the retrieval and discovery of relevant literature. Any references identified with AI assistance were independently verified by the authors against the original sources. All AI-assisted manuscript changes were reviewed by the authors. The authors take responsibility for the final content of this work.

REPRODUCIBILITY STATEMENT

The experimental setup, datasets, optimization details, evaluation protocol, and primary results are described in Sections 4.1 and 4.2. Additional hyperparameters and implementation details are provided in Appendix A.1; pseudocode for goal-aware rollout is given in Appendix A.6; compute-budget comparisons are reported in Appendix A.5. Anonymous code and model checkpoints are available at `https://anonymous.4open.science/r/SEG-ATP`.

REFERENCES

Rafael Cabral, Tuan Manh Do, Xuejun Yu, Wai Ming Tai, Zijin Feng, and Xin Shen. Proofflow: A dependency graph approach to faithful proof autoformalization. *arXiv preprint arXiv:2510.15981*, 2025.

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. *arXiv preprint arXiv:2506.11487*, 2025.

Guoxin Chen, Jing Wu, Xinjie Chen, Wayne Xin Zhao, Ruihua Song, Chengxi Li, Kai Fan, Dayiheng Liu, and Minpeng Liao. Reform: Reflective autoformalization with prospective bounded sequence optimization. *arXiv preprint arXiv:2510.24592*, 2025a.

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*, 2025b.

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

Kefan Dong and Tengyu Ma. STP: Self-play LLM theorem provers with iterative conjecturing and proving. *arXiv preprint arXiv:2502.00212*, 2025.

Jesse Michael Han, Jason Rute, Yuhuai Wu, Edward W Ayers, and Stanislas Polu. Proof artifact co-training for theorem proving with language models. *arXiv preprint arXiv:2102.06203*, 2021.

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*, pp. 1–3, 2025.

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. *arXiv preprint arXiv:2210.12283*, 2022.

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

Yang Li, Dong Du, Linfeng Song, Chen Li, Weikang Wang, Tao Yang, and Haitao Mi. Hunyuanprover: A scalable data synthesis framework and guided tree search for automated theorem proving. *arXiv preprint arXiv:2412.20735*, 2024.

Haohan Lin, Zhiqing Sun, Sean Welleck, and Yiming Yang. Lean-STaR: Learning to interleave thinking and proving. *arXiv preprint arXiv:2407.10040*, 2024.

Yong Lin, Shange Tang, Bohan Lyu, Jiayun Wu, Hongzhou Lin, Kaiyu Yang, Jia Li, Mengzhou Xia, Danqi Chen, Sanjeev Arora, et al. Goedel-prover: A frontier model for open-source automated theorem proving. *arXiv preprint arXiv:2502.07640*, 2025a.

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. *arXiv preprint arXiv:2508.03613*, 2025b.

Minh-Thang Luong, Dawsen Hwang, Hoang H Nguyen, Golnaz Ghiasi, Yuri Chervonyi, Insuk Seo, Junsu Kim, Garrett Bingham, Jonathan Lee, Swaroop Mishra, et al. Towards robust mathematical reasoning. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 35406–35430, 2025.

Leonardo de Moura and Sebastian Ullrich. The lean 4 theorem prover and programming language. In *Automated Deduction–CADE 28: 28th International Conference on Automated Deduction, Virtual Event, July 12–15, 2021, Proceedings 28*, pp. 625–635. Springer, 2021.

Dominique Pastre. Automated theorem proving in mathematics. *Annals of Mathematics and Artificial Intelligence*, 8:425–447, 1993.

Stanislas Polu and Ilya Sutskever. Generative language modeling for automated theorem proving. *arXiv preprint arXiv:2009.03393*, 2020.

Stanislas Polu, Jesse Michael Han, Kunhao Zheng, Mantas Baksys, Igor Babuschkin, and Ilya Sutskever. Formal mathematics statement curriculum learning. *arXiv preprint arXiv:2202.01344*, 2022.

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. URL https://arxiv.org/abs/2504.21801.

Carsten Schürmann and Frank Pfenning. Automated theorem proving in a simple meta-logic for lf. In *International Conference on Automated Deduction*, pp. 286–300. Springer, 1998.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Y Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Aaditya Singh, Adam Fry, Adam Perelman, Adam Tart, Adi Ganesh, Ahmed El-Kishky, Aidan McLaughlin, Aiden Low, AJ Ostrow, Akhila Ananthram, et al. Openai gpt-5 system card. *arXiv preprint arXiv:2601.03267*, 2025.

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

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

Zijian Wu, Suozhi Huang, Zhejian Zhou, Huaiyuan Ying, Jiayu Wang, Dahua Lin, and Kai Chen. InternLM2.5-StepProver: Advancing automated theorem proving via expert iteration on large-scale lean problems. *arXiv preprint arXiv:2410.15700*, 2024.

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. *arXiv preprint arXiv:2405.14333*, 2024a.

Huajian Xin, ZZ Ren, Junxiao Song, Zhihong Shao, Wanjia Zhao, Haocheng Wang, Bo Liu, Liyue Zhang, Xuan Lu, Qiushi Du, et al. DeepSeek-Prover-V1.5: Harnessing proof assistant feedback for reinforcement learning and monte-carlo tree search. *arXiv preprint arXiv:2408.08152*, 2024b.

Ran Xin, Chenguang Xi, Jie Yang, Feng Chen, Hang Wu, Xia Xiao, Yifan Sun, Shen Zheng, and Kai Shen. Bfs-prover: Scalable best-first tree search for llm-based automatic theorem proving. *arXiv preprint arXiv:2502.03438*, 2025a.

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

An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao, Bowen Yu, Chengpeng Li, Dayiheng Liu, Jianhong Tu, Jingren Zhou, Junyang Lin, Keming Lu, Mingfeng Xue, Runji Lin, Tianyu Liu, Xingzhang Ren, and Zhenru Zhang. Qwen2.5-math technical report: Toward mathematical expert model via self-improvement. *arXiv preprint arXiv:2409.12122*, 2024.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang,

Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Kaiyu Yang, Aidan Swope, Alex Gu, Rahul Chalamala, Peiyang Song, Shixing Yu, Saad Godil, Ryan J Prenger, and Animashree Anandkumar. Leandojo: Theorem proving with retrieval-augmented language models. *Advances in Neural Information Processing Systems*, 36:21573–21612, 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. *arXiv preprint arXiv:2406.03847*, 2024.

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 Appendix

A.1 Additional Experimental Details

Dataset processing details. We use three Lean 4 proof datasets as training sources: STP, LeanWorkbook, and NuminaMath-LEAN, and use miniF2F and FATE-M as evaluation benchmarks. Since the raw formats of these training sources are not identical, we first normalize them into executable proof scripts. For examples that already provide a complete Lean proof script, we parse the script into a sequence of tactics. For examples that provide state–tactic pairs, we concatenate the tactics according to their original proof order to recover a proof script that can be executed step by step in Lean. We then replay each proof script in a unified Lean 4.11 environment using LeanDojo (Yang et al., 2023), and retain only the examples that can be successfully executed and eventually finish the proof. After this process, each valid example is converted into a verified proof-state trajectory, from which all supervision granularities are constructed.

Before replay, each proof script is parsed into a sequence of executable tactic blocks. Since a single Lean tactic may span multiple lines, naively splitting a proof script by line can break tactic executability. We therefore remove empty lines, strip the common indentation of the proof body, and merge multi-line fragments according to the continuation structure of Lean tactics. For example, fragments connected by indentation, `<;>`, parentheses, or braces are merged into the same tactic block. If the indentation or multi-line structure of a proof script does not match the expected format, the example is treated as a parsing failure and filtered out. For successfully parsed examples, we feed the theorem context and the extracted tactic blocks to LeanDojo, and execute the tactics sequentially from the initial tactic state of the theorem. After each tactic execution, LeanDojo returns either a new proof state or an execution failure. We record the pretty-printed proof state before each tactic together with the corresponding tactic, and use the resulting proof state as the input state for the next replay step. If any tactic fails to execute, or if the replay does not complete the proof successfully, the example is excluded from subsequent training-data construction. This ensures that all supervision examples are derived from proof processes that are actually checked by Lean.

All supervision granularities are constructed from the same set of successfully replayed verified proof-state trajectories, so that comparisons across training objectives mainly reflect differences in supervision granularity. For step-level supervision, we construct one training example from each proof state and its next tactic in the trajectory. For whole-proof supervision, we use the initial proof state as input and the complete proof script as output. For segment-level supervision, we scan the replayed trajectory in order and split it into local proof segments according to changes in the open-goal count.

Specifically, in the LeanDojo pretty-printed proof states used in our processing pipeline, different open goals are typically separated by blank lines. We therefore parse each proof state into goal blocks according to this format, and treat a block as a valid open goal if it contains exactly one target marker $\vdash$. The open-goal count is the number of such valid goal blocks. We then start from a boundary proof state and collect the following tactics into the current segment as long as the open-goal count of subsequent proof states remains the same as that of the segment-start state. When we encounter a proof state whose open-goal count changes, we close the current segment and start a new segment from that state. The final segment is closed when the trajectory successfully completes the proof. Each resulting segment takes its starting proof state as input and uses as output the consecutive tactics that, when executed in order, lead to the next boundary state or complete the proof. Compared with step-level targets, segment-level targets capture multi-step local proof progress; compared with whole-proof targets, they remain confined to a more local proof stage.

The alternative segmentation data used in the boundary-selection ablation is constructed from the same successfully replayed verified proof-state trajectories, with only the boundary selection rule changed. The detailed rules are described in Appendix A.2.1.

After constructing the supervision examples, we serialize all of them into a unified instruction-tuning format. The instruction field is

`[GOAL]\n{state}\n[PROOFSTEP]\n`,

where `{state}` denotes the input proof state of the corresponding supervision example. The `input` field is left empty, and the `output` field contains the target tactic, tactic segment, or complete proof

script depending on the supervision granularity. For segment-level data, the `output` field may contain multiple tactics joined by newline characters.

Training details. In the main supervision-granularity experiments, all policy models are fully fine-tuned from Qwen2.5-Math-7B, and all training runs are conducted on 8 H200 GPUs. For a controlled comparison, step-level, whole-proof, and segment-level models use the same training configuration: the Adam optimizer, no weight decay, a global batch size of 2048, an initial learning rate of 2×10^{-5}, 3 training epochs, a warmup ratio of 0.05, and a cosine learning-rate schedule whose minimum learning rate is set to 10% of the initial value. We conduct experiments on three Lean 4 proof datasets, STP, LeanWorkbook, and NuminaMath-LEAN. Each trained model is evaluated on miniF2F and FATE-M.

Evaluation details. We evaluate three supervision granularities and one auxiliary protocol on miniF2F and FATE-M using Lean 4.11, with all inference evaluations conducted on 4 A800 GPUs. For each model and evaluation setting, we perform five inference runs using the same fixed model checkpoint and report proof success rates as mean $\pm$ standard deviation across these runs. Step-level policies generate one atomic tactic from the current proof state and are combined with best-first tree search. Segment-level policies generate tactic sequences from boundary proof states, and each generated sequence is treated as a macro action in the same best-first search framework. Whole-proof policies generate complete proof scripts from the initial proof state and directly submit them to Lean for verification. Whole-proof-seg evaluates the whole-proof policy under the segment-style search interface to isolate the effect of the training objective under a shared search budget. We use beam search as the decoding strategy for model generation. For Step-level, Segment-level, and Whole-proof-seg, we use beam size 8, at most 600 expansions, and a per-theorem timeout of 1800 seconds. For Segment-level and Whole-proof-seg, generated tactic sequences are executed in order; if execution fails, we keep the longest successfully executed prefix that advances the proof state as the search transition. For Whole-proof evaluation, we sample at most 2048 complete proof attempts per theorem. All evaluations stop once a proof is found. We report token cost as the total number of output tokens generated during inference, including tactics, proof segments, or proof scripts, and time cost as the total prover time per theorem, including both model generation and Lean execution. For each training-data/evaluation-set pair, the common-solved subset is the fixed intersection of theorems solved by every compared method across all five runs. Each method's token and time costs are averaged over this subset and the five runs. For goal-aware rollout, we apply the method to BFS-Prover-V2-7B and InternLM2.5-StepProver without retraining, compare standard best-first search with its rollout variant under the same search budget, fix the rollout horizon to $H = 5$, and count all rollout model generations, Lean executions, and output tokens toward inference cost. For each step-level prover and evaluation set, rollout efficiency metrics use a fixed subset of theorems solved by both search methods in all five runs. Token and time costs are averaged over this subset and the five runs.

A.2 Ablation Experiment

A.2.1 Alternative Boundary Selection Strategies

To further analyze the effect of boundary selection on segment-level supervision, we compare several proof segmentation strategies under the same training and evaluation setup. This experiment is not intended to show that any particular boundary signal is globally optimal; rather, under a unified training and evaluation setup, it compares the final proving performance obtained from training data constructed with different boundary selection strategies.

We consider three alternative strategies. *Token-based* segmentation groups consecutive tactics so that the total number of tactic tokens in each segment is close to a preset threshold, with thresholds of 32, 64, and 96. *Tactic-distance-based* segmentation computes the normalized edit distance between adjacent tokenized tactics and starts a new segment when the distance exceeds a preset threshold. *State-distance-based* segmentation similarly uses normalized edit distance, but compares the current proof state with the starting proof state of the current segment. For both distance-based strategies, we test thresholds of 0.4, 0.6, and 0.8. The two distance-based strategies differ in their focus: tactic distance captures local continuity between adjacent proof actions, while state distance measures how far the current proof state has moved away from the segment's starting state. We also include the goal-change-based segment-level method used in the main text for comparison.

Table 4: Ablation study of alternative boundary selection strategies. All models are trained on NuminaMath-LEAN and evaluated on miniF2F. Results are reported as mean ± std over five independent runs unless otherwise specified.

Boundary selection	Setting	Acc. (%) ↑
Token-based	32 tokens	59.43 ± 0.87
Token-based	64 tokens	58.85 ± 1.18
Token-based	96 tokens	59.92 ± 0.53
Tactic-distance-based	threshold 0.4	61.23 ± 1.22
Tactic-distance-based	threshold 0.6	60.57 ± 0.18
Tactic-distance-based	threshold 0.8	60.08 ± 1.07
State-distance-based	threshold 0.4	59.51 ± 0.98
State-distance-based	threshold 0.6	58.11 ± 1.14
State-distance-based	threshold 0.8	52.62 ± 1.60
Open-goal change	/	**66.31 ± 0.79**

Considering the scale of different training data sources and the overall cost of a full boundary-selection ablation, we conduct the complete comparison under one representative setting: training on NuminaMath-LEAN and evaluating on miniF2F. We choose this setting because NuminaMath-LEAN is one of our main training data sources, the NuminaMath-LEAN-trained segment-level policy achieves the highest miniF2F success rate in the main results, and miniF2F is a standard benchmark for formal theorem proving. We also conduct partial tests on other training data sources and observe similar trends. Table 4 reports the mean ± std over multiple independent runs for each setting.

As shown in Table 4, boundary selection has a substantial effect on segment-level supervision when training on NuminaMath-LEAN and evaluating on miniF2F. Among the alternative segmentation strategies, token-based segmentation is relatively stable around 59%, while distance-based strategies perform better with lower thresholds: tactic-distance-based and state-distance-based segmentation reach 61.23% and 59.51% at threshold 0.4, respectively. These results show that length-based and textual-distance-based rules can produce valid segmentations in this setting. In contrast, the open-goal-change strategy used in the main experiments achieves 66.31%, outperforming all alternative boundary strategies. This suggests that the benefit of segment-level supervision comes not merely from grouping multiple tactics, but from selecting boundaries that better reflect structural changes in the proof state.

Overall, this ablation shows that boundary selection affects both the construction of segment-level training data and final proving performance. Simply increasing the supervision unit length does not necessarily produce effective segments, and edit-distance-based strategies can depend on textual representations and threshold choices. Open-goal change, while not claimed to be globally optimal, provides a simple and effective instantiation of the boundary selection framework in the settings studied here.

A.2.2 Sensitivity to the Rollout Horizon

We further analyze the effect of the rollout horizon H in goal-aware rollout. Here, H is an inference-time search hyperparameter that denotes the maximum number of linear rollout steps performed along the current branch after a candidate tactic triggers an open-goal change. The main experiments use $H = 5$ by default. In this section, we additionally compare $H = 3$ and $H = 7$ to study the sensitivity of our method to the rollout horizon.

The experimental setup follows the goal-aware rollout experiments in the main text. We do not retrain the models, and only vary the rollout horizon at inference time. We evaluate two existing step-level provers: InternLM2.5-StepProver and BFS-Prover-V2-7B. All methods use the same explicit search budget, including beam size, maximum number of expansions, and timeout. We compare standard best-first search with goal-aware rollout under $H \in \{3, 5, 7\}$. Accuracy is computed on the full miniF2F test set. Token cost and time cost follow the main evaluation protocol, and are computed on the set of problems solved by all methods under the same step-level prover.

Table 5: Sensitivity analysis of the rollout horizon H in goal-aware rollout. Accuracy is evaluated on miniF2F and reported as mean $\pm$ std over five independent runs. Token and time costs are computed on the common-solved subset. Here, the common-solved subset is computed over Best-first and all rollout horizons under the same step-level prover.

Step-level prover	Search	Acc. (%) ↑	Tok. ↓	Time (s) ↓
InternLM2.5-StepProver	Best-first	59.59 ± 0.37	966.05	17.05
InternLM2.5-StepProver	$H = 3$	60.08 ± 0.85	620.81	12.68
InternLM2.5-StepProver	$H = 5$	$\mathbf{60.33 \pm 0.45}$	**590.77**	**10.88**
InternLM2.5-StepProver	$H = 7$	60.08 ± 0.62	673.30	15.96
BFS-Prover-V2-7B	Best-first	68.77 ± 0.79	1410.36	28.34
BFS-Prover-V2-7B	$H = 3$	69.10 ± 0.62	988.57	32.27
BFS-Prover-V2-7B	$H = 5$	70.74 ± 1.18	911.12	**24.88**
BFS-Prover-V2-7B	$H = 7$	$\mathbf{70.98 \pm 1.02}$	**815.04**	25.62

Table 5 shows that goal-aware rollout with $H \in \{3, 5, 7\}$ improves proof success rates over standard best-first search on both step-level provers, while generally reducing token cost on the common-solved subset. The best horizon varies across provers. For InternLM2.5-StepProver, $H = 5$ achieves the highest accuracy, the lowest token cost, and the lowest time cost. For BFS-Prover-V2-7B, $H = 7$ achieves the highest accuracy and the lowest token cost, while $H = 5$ gives the lowest time cost, with only a small accuracy gap between the two.

These results suggest that H should be treated as an inference-time hyperparameter rather than a fixed constant with a universal optimum. Intuitively, when the policy model is weaker, a larger H may cause search to continue along incorrect branches and delay backtracking, making a shorter rollout horizon more robust. In contrast, when the policy model is stronger, or when test problems require multi-step local reasoning, a larger H may help the model continue local proof progress after an open-goal change. In practice, H can be selected from a small candidate set, such as $H \in \{3, 5, 7\}$ or a few values below 10, depending on the desired trade-off among accuracy, token cost, and time cost. In the main experiments, we use $H = 5$ as a unified default to keep the inference configuration consistent across different step-level provers.

A.3 Additional Experiments with Qwen3-14B

To examine whether the effectiveness of segment-level supervision generalizes beyond the model used in the main experiments, we further evaluate it with Qwen3-14B (Yang et al., 2025). We fine-tune Qwen3-14B on NuminaMath-LEAN with different supervision granularities and evaluate the fine-tuned provers on miniF2F and FATE-M.

As shown in Table 6, Segment-level achieves the best accuracy and inference efficiency on both benchmarks. On miniF2F, it obtains 53.36% accuracy with 458.07 tokens and 7.04 seconds, while on FATE-M it achieves 17.87% accuracy with 1574.75 tokens and 21.97 seconds. Compared with Whole-proof-seg, which uses the same segment-style search interface, Segment-level achieves substantial accuracy gains of 8.61 and 9.07 percentage points on miniF2F and FATE-M, respectively, while also reducing inference costs. These results indicate that the improvement comes from learning goal-change-based proof segments rather than merely adopting a segment-style search procedure, and further support the general applicability of segment-level supervision beyond the main experimental setting.

A.4 Additional Analysis of Accuracy and Inference Efficiency

Tables 1 and 2 show that Segment-level achieves the highest proof success rate across all training sources on both miniF2F and FATE-M, while also attaining the lowest token or time cost in most comparisons. This pattern suggests that, when proofs contain sufficient multi-step local progress, segment-level supervision can improve proof success while reducing inference cost by reducing repeated fine-grained search. A notable boundary case arises with LeanWorkbook training, where the advantage in inference cost is less consistent despite the higher proof success rate.

Table 6: Results with Qwen3-14B trained on NuminaMath-LEAN. Accuracy is reported as mean ± std, with values expressed as percentages. Token and time costs are averaged on the common-solved subset within each evaluation setting, with time measured in seconds. Bold indicates the best result for each metric within each benchmark, and shaded rows indicate our method.

Training data	Eval set	Method	Acc. (%) ↑	Tok. ↓	Time (s) ↓
NuminaMath-LEAN	miniF2F	Step-level	52.46 ± 0.65	535.89	9.00
NuminaMath-LEAN	miniF2F	Whole-proof	48.61 ± 1.11	4588.00	32.02
NuminaMath-LEAN	miniF2F	Whole-proof-seg	44.75 ± 1.17	1355.62	20.81
NuminaMath-LEAN	miniF2F	Segment-level	$\mathbf{53.36 \pm 0.34}$	**458.07**	**7.04**
NuminaMath-LEAN	FATE-M	Step-level	17.20 ± 0.56	1794.20	26.66
NuminaMath-LEAN	FATE-M	Whole-proof	16.27 ± 0.76	10412.40	55.73
NuminaMath-LEAN	FATE-M	Whole-proof-seg	8.80 ± 1.10	4335.55	57.46
NuminaMath-LEAN	FATE-M	Segment-level	$\mathbf{17.87 \pm 0.87}$	**1574.75**	**21.97**

For LeanWorkbook training, Segment-level still achieves the highest proof success rate on both miniF2F and FATE-M, but its cost advantage is less consistent. This behavior is consistent with the proof-length distribution of LeanWorkbook: about 46% of the training proofs are completed in one step and about 65% within two steps, leaving relatively little multi-step local progress for segment-level supervision to exploit. Consequently, the model has fewer opportunities to learn segments that combine several useful tactic transitions into a single action. Nevertheless, Segment-level still reduces the average number of model calls from 2.09 to 1.92 on miniF2F and from 4.51 to 3.60 on FATE-M compared with Step-level. These results suggest that segment-level supervision continues to reduce repeated model interactions, but the predominance of short proofs in LeanWorkbook limits the extent to which this reduction translates into lower token and time costs.

A.5 Computational Fairness under Actual Inference Budgets

The main experiments report final success rates under a fixed timeout. However, different proof search strategies operate at different granularities and incur different generation and execution costs. To characterize their efficiency under constrained resources, we further analyze proof solving behavior under varying inference budgets from both time and token perspectives.

A.5.1 Time-to-Solve Analysis

The main results in Section 4.2 compare final proof success rates on miniF2F and FATE-M across training sources and report average inference costs on the common-solved subset. To examine solving performance under different per-theorem time limits, we report cumulative accuracy curves on miniF2F and FATE-M.

For each theorem, solving time is the elapsed time from the start of evaluation on that theorem until a valid proof is first found. Given a time cutoff t, cumulative accuracy is the fraction of evaluation theorems proved within t seconds. This analysis uses solving-time records from existing runs rather than rerunning search at each cutoff. The x-axis is displayed as $\log(1 + t)$. Each curve reports the mean over five independent runs, and the shaded region denotes the min–max range across runs.

As shown in Figure 3, Segment-level generally achieves higher cumulative accuracy under most time cutoffs, although Step-level can perform better at early cutoffs in several settings. This is expected because single-step tactic prediction incurs lower per-step overhead, making it more efficient when the available time budget is limited or proofs are short. As the time budget increases, Segment-level gradually surpasses Step-level in most settings, indicating that its advantage does not merely come from solving additional problems near the full timeout, but from a more efficient search process. By generating local proof segments instead of individual tactics, Segment-level reduces fine-grained search overhead and better exploits multi-step proof progressions. This advantage is less pronounced when trained on LeanWorkbook, where the training data contains a large proportion of short proofs and provides fewer multi-step proof progressions for segment-level supervision to exploit.

We further apply the same analysis to goal-aware rollout on miniF2F and FATE-M. For both InternLM2.5-StepProver and BFS-Prover-V2-7B, goal-aware rollout achieves higher cumulative accuracy than standard best-first search at most time cutoffs, as shown in Figure 4. These results show that goal-aware rollout remains effective under tighter time constraints, rather than only improving final proof success with the full search budget.

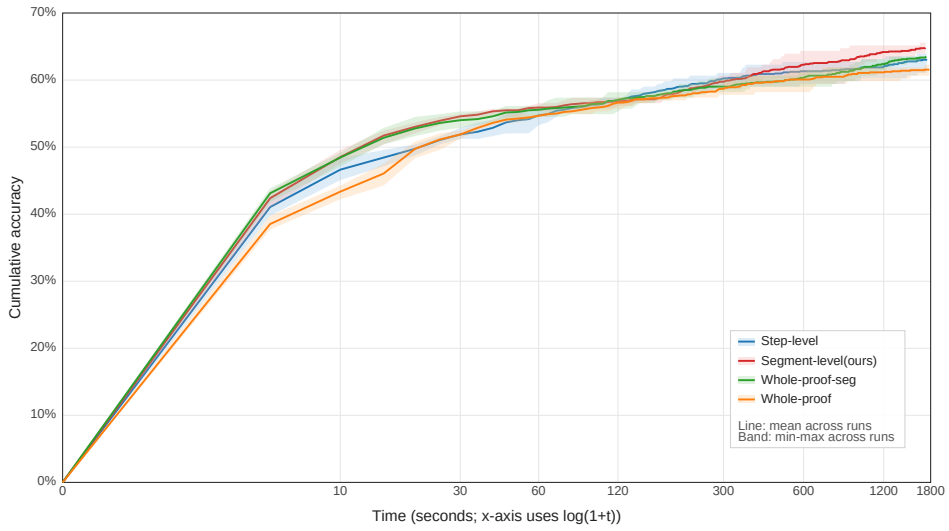
A.5.2 TOKEN-BUDGET ANALYSIS

The main results in Section 4.2 compare final proof success rates on miniF2F and FATE-M across training sources and report average inference costs on the common-solved subset. To examine solving performance under different output-token budgets, we report proof success rates under varying cumulative output-token thresholds on miniF2F and FATE-M.

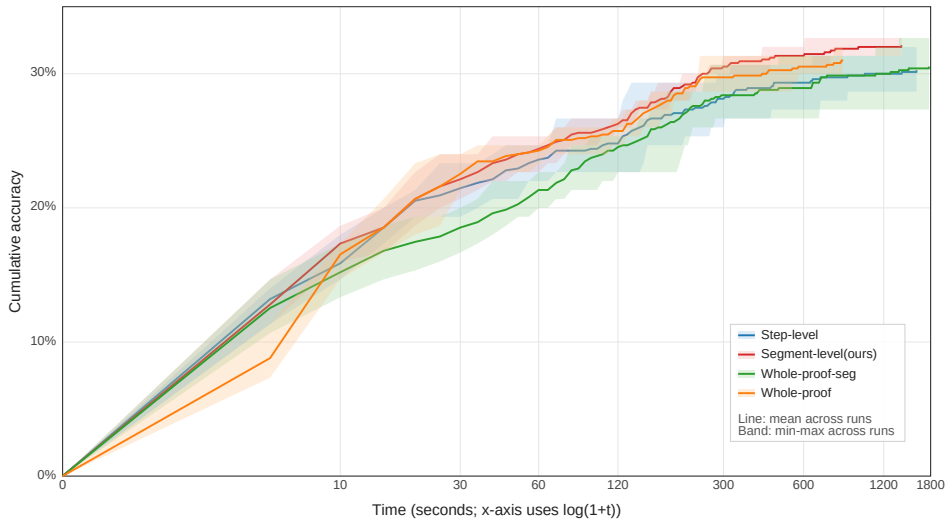
For each theorem, cumulative output-token consumption is the total number of generated tokens from the start of evaluation on that theorem until a valid proof is first found. Given a token threshold k, token-budget accuracy is the fraction of evaluation theorems proved with cumulative output-token consumption no larger than k. This analysis uses token-consumption records from existing runs rather than rerunning search at each threshold. Each reported value is averaged over five independent runs.

As shown in Table 7, Segment-level generally achieves higher proof success rates under different output-token budgets. Across the three training sources and two evaluation sets, Segment-level achieves the best performance at most token thresholds, with the advantage becoming more pronounced as the token budget increases. At very small token budgets, Step-level can occasionally perform better because single-step tactic prediction incurs lower per-step generation overhead. Under such constrained budgets, Segment-level has fewer opportunities to exploit the benefits of multi-step proof compression.

For goal-aware rollout, Table 8 shows that the improvements over standard best-first search persist as the output-token budget varies for both InternLM2.5-StepProver and BFS-Prover-V2-7B on miniF2F and FATE-M. These results demonstrate that goal-aware rollout improves search efficiency not only under the full evaluation budget, but also when the available output-token budget is constrained.

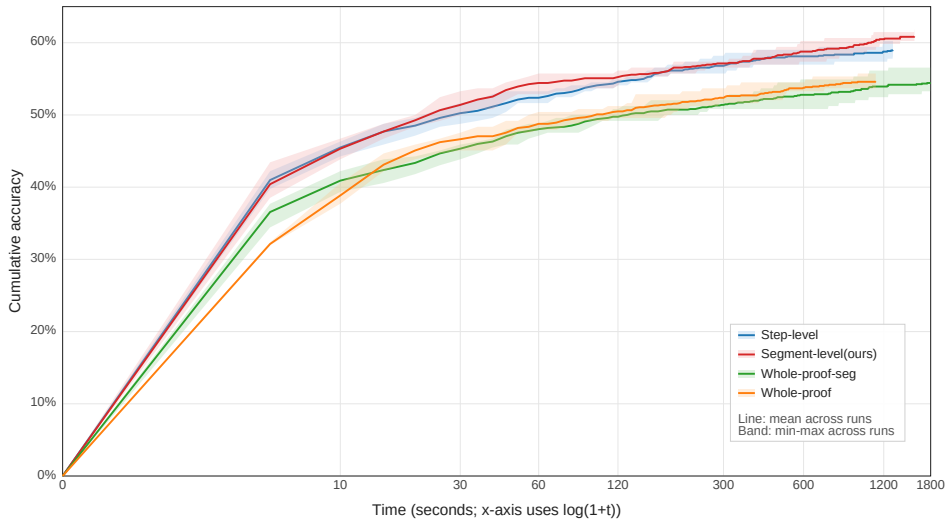


(a) miniF2F

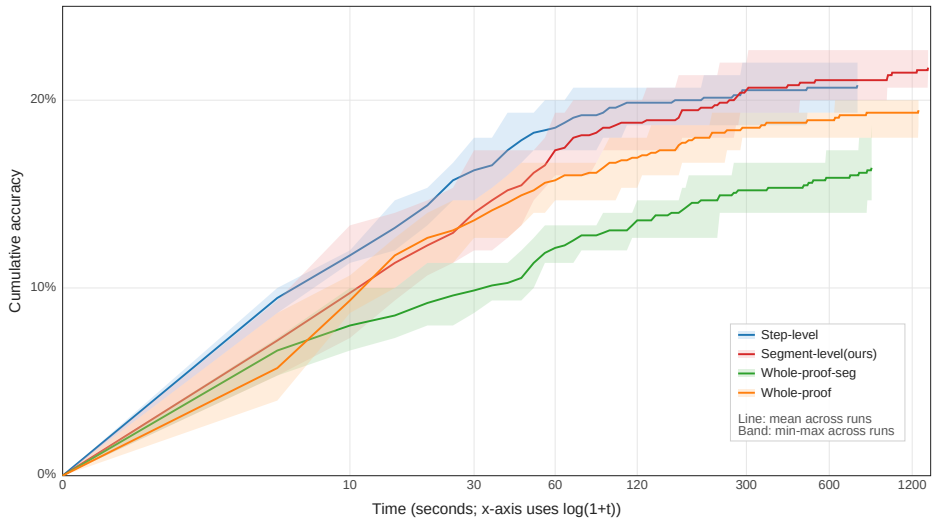


(b) FATE-M

Figure 3: Cumulative accuracy over per-theorem elapsed time for models trained on STP. The two panels report results on miniF2F and FATE-M, respectively. Cumulative accuracy is the fraction of the full evaluation set solved within t seconds, and the x-axis uses $\log(1 + t)$. Lines denote the mean over five independent runs, and shaded regions denote the min–max range across runs. Each panel retains its original time range.

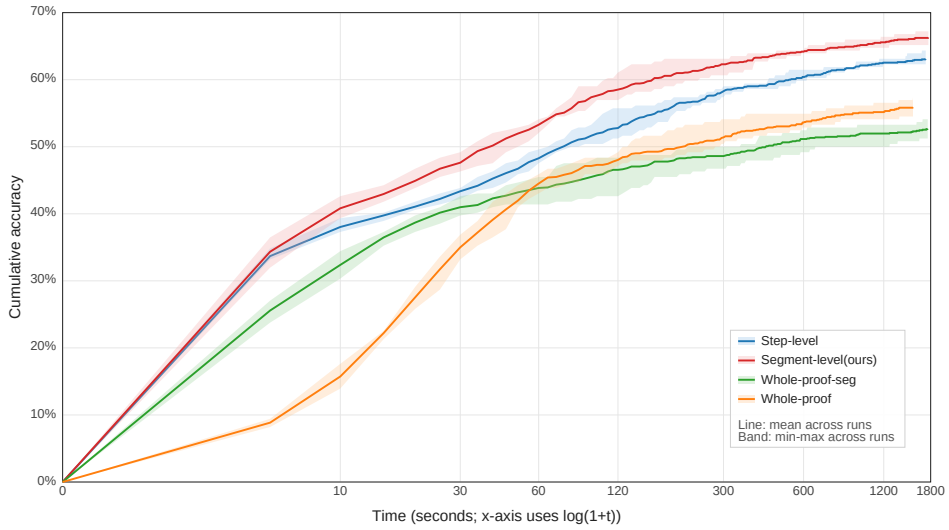


(c) miniF2F

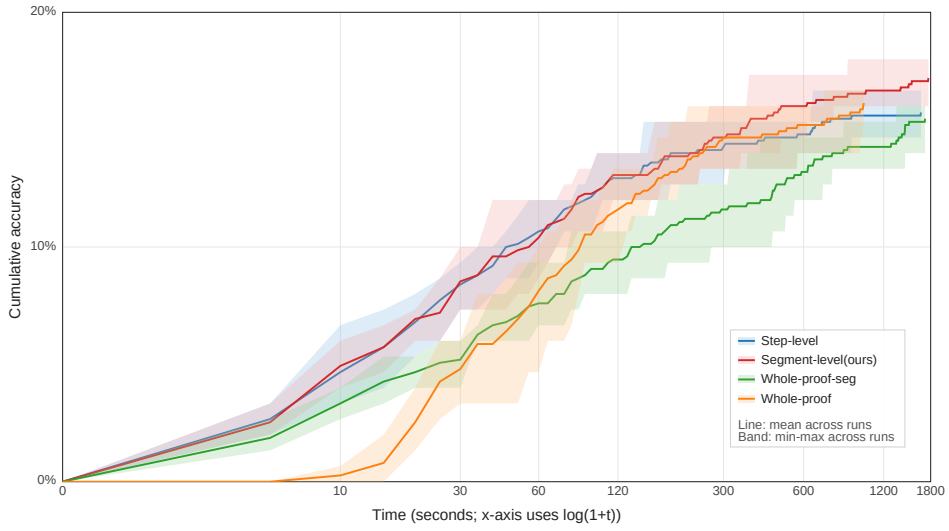


(d) FATE-M

Figure 3: Cumulative accuracy over per-theorem elapsed time for models trained on LeanWorkbook. The two panels report results on miniF2F and FATE-M, respectively, with the same plotting protocol as Figures 3(a,b). On FATE-M, Step-level leads over part of the early-to-middle time range, while Segment-level achieves higher cumulative success later.

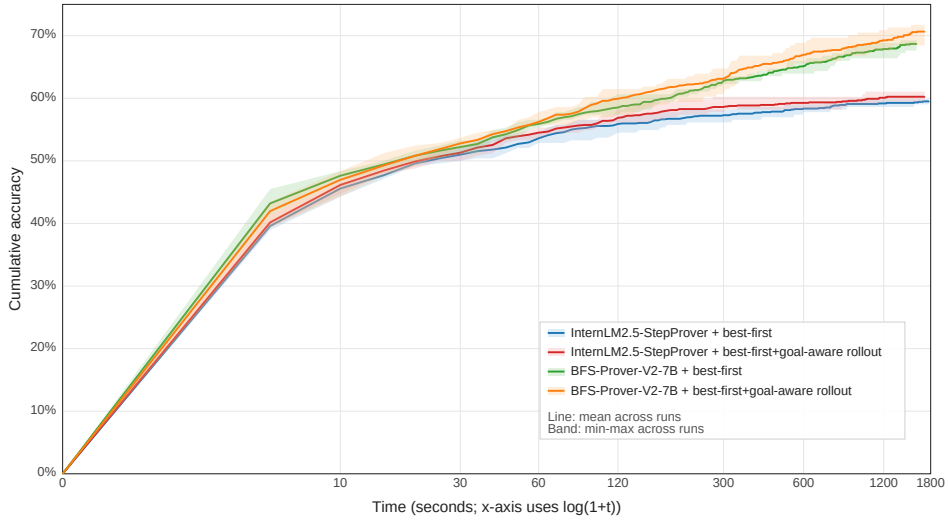


(e) miniF2F

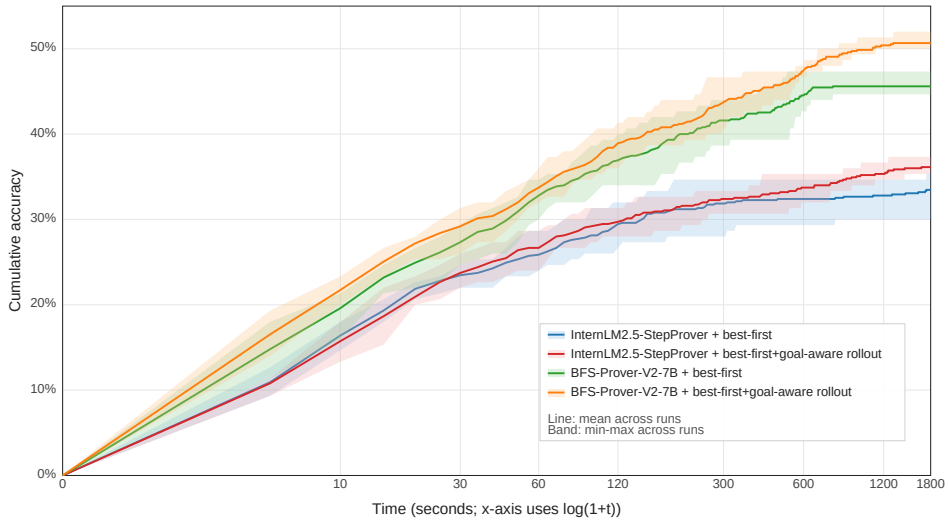


(f) FATE-M

Figure 3: Cumulative accuracy over per-theorem elapsed time for models trained on NuminaMath-LEAN. The two panels report results on miniF2F and FATE-M, respectively, with the same plotting protocol as Figures 3(a,b). On FATE-M, the Step-level and Segment-level curves are close or cross early, while Segment-level achieves higher cumulative success later.



(a) miniF2F



(b) FATE-M

Figure 4: Cumulative accuracy over per-theorem elapsed time for goal-aware rollout. The two panels report results on miniF2F and FATE-M, respectively. For each time cutoff t, cumulative accuracy is defined as the fraction of theorems solved within t seconds. All curves are computed from the same evaluation runs with a per-theorem timeout of 1800 seconds. The x-axis uses $\log(1 + t)$ for visualization. Lines denote the mean over five independent runs, and shaded regions denote the min–max range across runs.

Table 7: Proof success rates under different per-theorem cumulative output-token thresholds. Accuracy (%) is the mean over the same five independent evaluations; bold indicates the highest reported mean within each setting and threshold, with all ties at the reported precision shown in bold.

Training data	Eval set	Method	500	2,000	5,000	10,000	20,000	100,000
STP	miniF2F	Step-level	44.10	52.13	55.66	57.87	61.07	63.11
STP	miniF2F	Segment-level	44.43	**53.69**	**57.38**	**59.43**	**61.80**	**64.75**
STP	miniF2F	Whole-proof-seg	**44.84**	**53.69**	57.21	58.61	59.84	63.11
STP	miniF2F	Whole-proof	0.00	31.56	49.43	55.25	57.21	60.98
LeanWorkbook	miniF2F	Step-level	**45.25**	**51.31**	54.43	56.48	57.62	59.02
LeanWorkbook	miniF2F	Segment-level	42.54	51.23	**54.67**	**57.05**	**58.11**	**60.74**
LeanWorkbook	miniF2F	Whole-proof-seg	38.85	46.48	49.10	50.82	52.13	54.34
LeanWorkbook	miniF2F	Whole-proof	0.00	33.85	46.80	49.34	51.39	54.43
NuminaMath-LEAN	miniF2F	Step-level	**31.72**	41.48	46.56	50.98	55.82	61.97
NuminaMath-LEAN	miniF2F	Segment-level	**31.72**	**44.02**	**50.16**	**55.82**	**59.84**	**65.08**
NuminaMath-LEAN	miniF2F	Whole-proof-seg	28.44	39.75	43.44	46.56	48.69	51.72
NuminaMath-LEAN	miniF2F	Whole-proof	8.85	21.39	32.21	40.82	46.23	52.62
STP	FATE-M	Step-level	**14.00**	21.33	24.53	26.13	28.13	30.13
STP	FATE-M	Segment-level	13.47	**21.60**	**25.73**	**28.40**	**30.53**	**32.13**
STP	FATE-M	Whole-proof-seg	12.13	17.73	23.07	25.47	28.00	30.27
STP	FATE-M	Whole-proof	0.00	10.80	22.00	25.60	27.87	30.93
LeanWorkbook	FATE-M	Step-level	**10.53**	**17.33**	**20.00**	20.27	20.53	20.80
LeanWorkbook	FATE-M	Segment-level	7.73	13.73	17.20	**20.4**	**20.67**	**21.73**
LeanWorkbook	FATE-M	Whole-proof-seg	6.27	10.53	12.93	14.27	15.47	16.40
LeanWorkbook	FATE-M	Whole-proof	0.00	8.80	12.93	16.27	17.60	19.47
NuminaMath-LEAN	FATE-M	Step-level	**3.47**	**8.40**	11.20	13.07	13.87	15.60
NuminaMath-LEAN	FATE-M	Segment-level	**3.47**	**8.40**	**11.47**	**13.20**	**14.00**	**16.40**
NuminaMath-LEAN	FATE-M	Whole-proof-seg	2.00	4.67	7.07	9.07	11.20	14.27
NuminaMath-LEAN	FATE-M	Whole-proof	0.00	0.53	3.07	6.40	10.93	15.07

Table 8: Proof success rates under different per-theorem cumulative output-token thresholds for goal-aware rollout. Accuracy (%) is the mean over the same five independent evaluations; bold indicates the higher reported mean within each setting and threshold.

Step-level prover	Eval set	Search	500	2,000	5,000	10,000	20,000	100,000
InternLM2.5-StepProver	miniF2F	Best-first	43.77	52.21	55.82	56.89	57.70	59.43
InternLM2.5-StepProver	miniF2F	+ Goal-aware rollout	**44.75**	**53.36**	**56.80**	**58.28**	**59.02**	**60.16**
BFS-Prover-V2-7B	miniF2F	Best-first	47.79	54.75	58.93	61.56	64.75	68.52
BFS-Prover-V2-7B	miniF2F	+ Goal-aware rollout	**48.77**	**56.07**	**60.66**	**64.26**	**67.46**	**70.74**
InternLM2.5-StepProver	FATE-M	Best-first	13.20	24.13	28.00	30.93	32.53	33.47
InternLM2.5-StepProver	FATE-M	+ Goal-aware rollout	**14.00**	**25.07**	**29.47**	**31.73**	**33.20**	**36.13**
BFS-Prover-V2-7B	FATE-M	Best-first	16.40	27.07	34.27	37.73	41.60	45.60
BFS-Prover-V2-7B	FATE-M	+ Goal-aware rollout	**21.07**	**30.40**	**37.73**	**40.80**	**45.33**	**50.67**

A.6 Pseudocode for Goal-Aware Rollout

Algorithm 1 shows how the goal-aware rollout described in Section 3.3 is integrated with best-first search. The inputs are the initial proof state s_0, step-level policy p_θ, open-goal counting function g, rollout horizon H, and search budget B. Queue entries store a proof state together with the valid tactic path from the initial state to that state. $\mathrm{Solved}(s)$ indicates that Lean has verified proof completion. $\mathrm{ExpandCandidates}$ uses the step-level policy to generate candidate tactics, $\mathrm{PopBest}$ follows the original best-first search priority rule, and $\mathrm{InsertFrontier}$ retains its original scoring, deduplication, and pruning rules. The algorithm introduces no new node-scoring function.

Rollout is triggered only when a candidate tactic changes the number of open goals without completing the proof. The horizon H specifies the maximum number of additional rollout steps after the triggering

Algorithm 1 Goal-aware rollout integrated with best-first search

Require: Initial proof state s_0, step-level policy p_θ, open-goal counting function g, rollout horizon H, search budget B
Ensure: A Lean-verified proof path, or no proof found within the budget

$Q \leftarrow \{(s_0, \emptyset)\}$
while $Q \neq \emptyset$ and budget B is not exhausted **do**
  $(s, \pi) \leftarrow \mathrm{PopBest}(Q)$
  if $\mathrm{Solved}(s)$ **then**
    return π
  end if
  for each $a \in \mathrm{ExpandCandidates}(p_\theta, s)$ **do**
    if budget B is exhausted **then**
      return no proof found within the budget
    end if
    $u \leftarrow \mathrm{Exec}(s, a)$
    if $u = \bot$ **then**
      continue ▷ Candidate tactic fails; skip this candidate
    end if
    $m \leftarrow (a)$
    if $g(u) \neq g(s)$ and $\neg\mathrm{Solved}(u)$ **then**
      for $h = 1, \ldots, H$ **do**
        if budget B is exhausted **then**
          break
        end if
        $\tilde{a} \sim p_\theta(\cdot \mid u)$ ▷ Generate one continuation tactic
        $v \leftarrow \mathrm{Exec}(u, \tilde{a})$
        if $v = \bot$ **then**
          break ▷ Retain the last valid state and prefix
        end if
        $u \leftarrow v, m \leftarrow m \oplus (\tilde{a})$
        if $\mathrm{Solved}(u)$ **then**
          break
        end if
      end for
    end if
    if $\mathrm{Solved}(u)$ **then**
      return $\pi \oplus m$
    end if
    $\mathrm{InsertFrontier}(Q, u, \pi \oplus m)$ ▷ Only the final valid state enters the search frontier
  end for
end while
return no proof found within the budget

tactic; subsequent goal-count changes during rollout neither reset H nor trigger another rollout. The rollout terminates early if the proof is completed or a rollout tactic fails. In the latter case, the failed tactic is discarded while all preceding successful tactics are retained. Intermediate rollout states are not inserted into the search frontier separately; only the final valid proof state is retained.

Model generation and Lean execution remain subject to the original search budget and timeout constraints. The budget checks in the pseudocode represent these constraints and do not allow any generation or execution to exceed the original budget. The generation and execution costs of all rollout attempts, including unsuccessful ones, are included in the corresponding inference costs.