MONOTONIC REFERENCE-FREE REFINEMENT FOR AUTOFORMALIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

While statement autoformalization has advanced rapidly, full-theorem autoformalization remains largely unexplored. Existing iterative refinement methods for statement autoformalization typically improve isolated aspects of formalization quality, such as syntactic correctness, but struggle to jointly optimize multiple dimensions of quality, a key requirement for full-theorem autoformalization. We introduce a reference-free iterative monotonic process for full-theorem autoformalization at inference time that leverages complementary feedback from theorem provers and LLM-based judges, without access to existing formalizations or human intervention. Our approach optimizes a masked composite objective over a hard dimension, formal validity as determined by a theorem prover, and a set of soft dimensions. We introduce an acceptance policy over candidate formalizations that provides a certified notion of monotonic progress, ensuring at each iteration that either the true objective does not decrease or its lower confidence bound increases, while establishing conditions for convergence and termination. We also propose a responsiveness map that characterizes how different LLMs, acting in different roles, preferentially improve these dimensions, enabling principled generator selection. Experiments show that our method simultaneously improves multiple dimensions of formalization quality, achieving overall scores of 90.27% on miniF2F and 52.45% on ProofNet.

1 INTRODUCTION

Recent advances in Large Language Models (LLMs) have substantially improved their ability to perform mathematical reasoning (Shao et al., 2024; 2025; Petrov et al., 2025; Zhang et al., 2025e). However, reasoning trajectories expressed in natural-language mathematics are difficult to verify automatically, posing a major challenge for reliably evaluating instances generated through mathematical discovery (Romera-Paredes et al., 2024; Naskrecki & Ono, 2025). Autoformalization (Wu et al., 2022; Yang et al., 2025) provides a crucial bridge between informal and formal mathematics by converting informal reasoning into transparent and mechanically verifiable formalizations. Existing autoformalization research has primarily focused on statement formalization (Zhang et al., 2024; Lu et al., 2024; Li et al., 2024; Gao et al., 2025), often omitting full formal proofs by relying on "sorry" placeholders. However, proof-level autoformalization is equally essential for faithfully verifying the correctness of natural-language mathematical reasoning and remains an underexplored yet critical direction.

Given a natural-language theorem and its accompanying proof, *full-theorem autoformalization* aims to produce a complete formalization consisting of both a formal theorem statement and a machine-verified proof. This setting differs fundamentally from a pipeline that first autoformalizes the statement and then applies automated theorem proving (ATP). Such a pipeline treats the generated formal statement as fixed before proof generation begins. Consequently, if the generated statement is not syntactically well-formed in the target formal system, the ATP stage may not even be applicable. Even if the statement is formally well-formed, semantic errors introduced during statement formalization cannot be corrected or reconsidered in light of the proof. In contrast, full-theorem autoformalization jointly constructs the formal statement and proof from the natural-language theorem and its accompanying proof, allowing the two components to inform and constrain one another.

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.

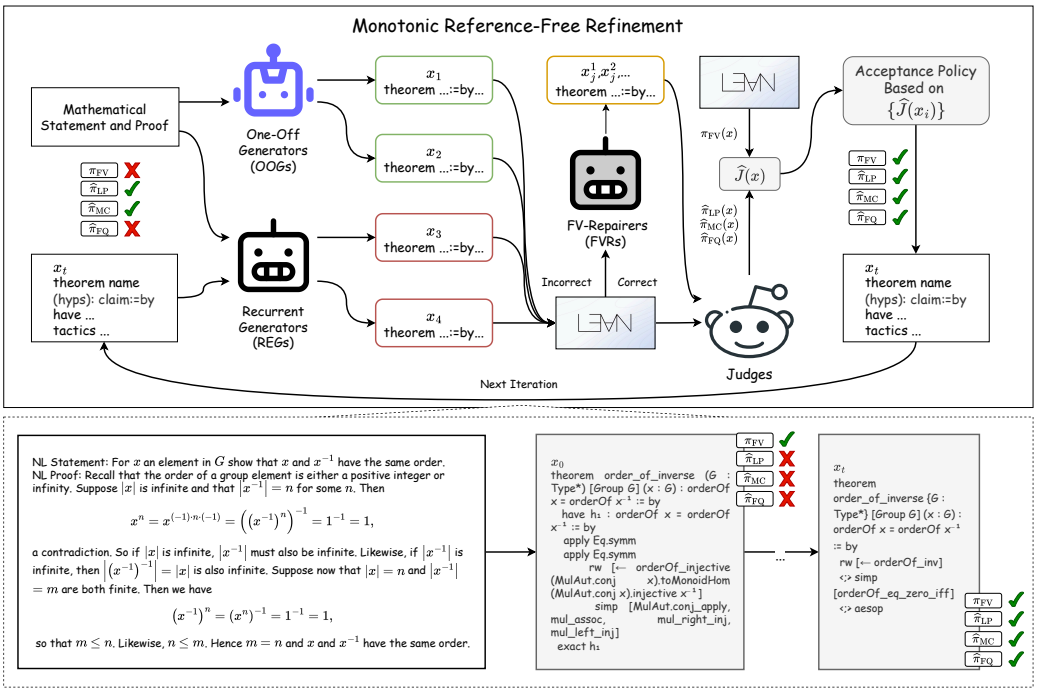


Figure 1: Schematic illustration of the monotonic process with an example. In this process, One-Off Generators (OOGs) produce formalizations from scratch, Recurrent Generators (REGs) refine the current best formalization using feedback from LLM judges, and FV-Repairers (FVRs) correct formally invalid candidates generated by OOGs and REGs. The acceptance policy retains only the formalization estimated to have the highest quality among all candidates and the previous best.

The process of full-theorem autoformalization is inherently a multi-dimensional optimization problem. *Formal Validity* (FV), which requires the generated formal code to be well-formed and the proof to be mechanically verified by the theorem prover, is a hard requirement for any successful formalization. However, machine-verified validity alone does not guarantee that the resulting formalization faithfully captures the intended theorem. For example, OpenAI (2026) reported several Lean 4-verified theorems and proofs, but Nielsen (2026) later identified semantic misalignment errors in some of these formalizations, calling the reliability of the corresponding results into question. Moreover, these formalizations can span thousands or even tens of thousands of lines and contain substantial redundancy, making human inspection and error localization difficult. These observations suggest that full-theorem autoformalization systems should also consider softer dimensions (Zhang et al., 2025c), such as *Logical Preservation* (LP) and *Mathematical Consistency* (MC), which assess semantic alignment with the original informal theorem and proof, as well as *Formal Quality* (FQ), which captures properties such as clarity, structure, concision, and reusability of the resulting formalization.

Given the multi-dimensional nature of full-theorem autoformalization, relying on a single LLM at test time, whether through vanilla prompting (Wu et al., 2022) or refinement-based methods, has limited capacity for simultaneously improving multiple dimensions (Zhang et al., 2024). Moreover, different LLMs used for refinement often exhibit distinct biases toward particular dimensions of formalization quality. This observation motivates the following research question: *How can we leverage the complementary strengths of different LLMs to improve full-theorem autoformalization across multiple objectives at test time?*

To address this research question, we propose a unified objective that accounts for both hard and soft dimensions, and design a *reference-free*, *iterative*, *monotonic* optimization process at test time (Figure 1) that provides a theoretically certified notion of progress under judge uncertainty. Our main contributions are summarized as follows:

1. To the best of our knowledge, we are the first to formally define *full-theorem autoformalization* and to study it using LLM-based methods. We propose a *masked composite objective* for this task that jointly considers multiple quality dimensions, including a hard dimension, Formal Validity (FV), and several soft dimensions.

2. We decompose the roles of LLMs in full-theorem autoformalization into three categories: One-Off Generators (OOG), FV-Repairers (FVR), and Recurrent Generators (REG). The FVRs and REGs can leverage feedback to refine formalizations along different quality dimensions. We further introduce a *responsiveness map* that models the expected, dimension-specific improvements induced by each generator, thereby enabling sample-efficient generator selection.

3. We propose a monotonic optimization process at test time that employs an acceptance policy aggregating signals from both theorem prover and LLM-based judges, without access to ground-truth or existing formalizations and without human involvement in selection, ensuring certified progress across iterations, such that either the true objective does not decrease or its *lower confidence bound* (LCB) increases.

4. We empirically demonstrate that specialized mathematical LLMs and general-purpose LLMs exhibit complementary strengths across different generator roles, and that our iterative framework effectively ensembles these strengths to achieve monotonic improvement. For full-theorem autoformalization on the miniF2F and ProofNet benchmarks, our final monotonic process achieves 100.00% and 77.96% formal validity, as verified by the Lean4 theorem prover, and overall scores of 90.27% and 52.45%, based on combined judgments from the theorem prover and GPT-4.1-mini judges, respectively.

2 Monotonic Test-Time Optimization for Autoformalization

2.1 Problem Statement

The task of *full-theorem autoformalization* can be formally defined as follows. Given a mathematical statement s_{nl} expressed in natural language and its corresponding proof p_{nl}, the goal is to automatically construct a formalization $x = (s, p)$, where s is a formal theorem statement and p is a formal proof of s. The formal statement s should be semantically aligned with s_{nl}, while the proof p should preserve the essential reasoning of p_{nl} without necessarily following exactly the same sequence of reasoning steps. The quality of a formalization $x = (s, p)$ can be characterized along multiple dimensions:

Hard Dimension – Formal Validity (FV). A full-theorem formalization should be *formally valid* such that the formal statement s must be syntactically well-formed in the target formal system, and the formal proof p must be accepted by the corresponding theorem prover as a valid proof of s. Formal validity can therefore be rigorously and automatically verified by the theorem prover, and is also commonly measured by the pass rate.

Soft Dimensions $\mathcal{D}_{\mathrm{soft}}$. Soft dimensions capture properties of a formalization that cannot be determined solely by verification from the theorem prover. Examples include: **Logical Preservation (LP):** the formal theorem s should preserve the *logical content and inferential structure* of s_{nl}; **Mathematical Consistency (MC):** the formal theorem s should faithfully represent the *mathematical objects, relations, and operations* expressed in s_{nl};and **Formal Quality (FQ):** the full-theorem formalization x should exhibit high intrinsic quality in terms of structural clarity and conciseness, avoiding unnecessary statements and redundant reasoning steps.

Dimension Scores. Let $\mathcal{X}$ denote the space of all possible candidate formalizations. We score each candidate $x \in \mathcal{X}$ using the vector

$$\pi(x) = \Big(\pi_{\mathrm{FV}}(x), \big(\pi_i(x)\big)_{i \in \mathcal{D}_{\mathrm{soft}}}\Big), \tag{1}$$

where $\pi_{\mathrm{FV}}(x) \in \{0, 1\}$ indicates whether x passes verification by the theorem prover, and $\pi_i(x) \in [0, 1]$ measures the quality of x along soft dimension i. The goal of full-theorem autoformalization is to identify an ideal formalization $x^\star \in \mathcal{X}$ that achieves the optimal score across all dimensions, i.e., $\pi(x^\star) = \mathbf{1}$.

Masked Composite Objective. For the optimization process, we define an objective function:

$$J_{\mathrm{OA}}(x) = \pi_{\mathrm{FV}}(x) \sum_{i \in \mathcal{D}_{\mathrm{soft}}} w_i \pi_i(x), \tag{2}$$

where $w_i > 0$ and $\sum_i w_i = 1$. The multiplicative factor $\pi_{\mathrm{FV}}(x)$ is a *mask* since it enforces FV as a hard constraint while retaining a *single scalar potential* for the soft dimensions. This objective function precludes selecting invalid-but-high soft scores and matches downstream utility where FV is binary and primary. $x^\star \in \mathcal{X}$ is an ideal formalization if and only if $J_{\mathrm{OA}}(x^\star) = 1$.

2.2 Approximating Soft Dimensions with LLM Judges

The automatic and rigorous evaluation of soft dimensions such as LP, MC, FQ is normally intractable. Previous work (Zhang et al., 2025c) indicates the ability of using LLM judges for the evaluation of certain aspects of formalizations, however, LLM judges can be ambiguous and should be accepted under uncertainty. We make the following assumptions for LLM judges:

Assumption 2.1. We assume that for each soft dimension $i \in \mathcal{D}_{\mathrm{soft}}$, the fused estimator

$$\widehat{\pi}_i(x) = \sum_{k=1}^{K_i} w_{i,k} \widehat{\pi}_i^{(k)}(x), \tag{3}$$

constructed from signals $\widehat{\pi}_i^{(k)}(x) \in \{0, 1\},\ k = 1, \ldots, K_i$ produced by a set of K_i LLM judges, where $w_{i,k} > 0$ and $\sum_k w_{i,k} = 1$, satisfy the following:

(i) **Asymptotic stability**: After asymptotic calibration of the weights and LLM-based judgments, for example by using a sufficiently large number of samples from each LLM judge to obtain consistent evaluations, each $\widehat{\pi}_i(x)$ converges almost surely to a deterministic value for every x.

(ii) **Measurable uncertainty**: There exists a *one-sided lower confidence bound* for the true $\pi_i(x)$ at uncertainty level $\delta_i \in (0, 1)$, given by:

$$\Pr\left(\widehat{\pi}_i(x) - m_i(\delta_i) \le \pi_i(x)\right) \ge 1 - \delta_i, \tag{4}$$

where the margin term $m_i(\delta_i) > 0$ absorbs calibration error, inter-judge correlation, and distributional drift.

Assumption 2.1 formalizes the reliability of LLM judges. Specifically, (i) asserts that their judgments are stable across comparisons, and (ii) limits the likelihood that low-quality formalizations are mistakenly evaluated favorably along soft dimensions, with such errors occurring with probability at most δ_i. We provide a naive uncertainty estimation for LLM judges using Eq. 4 in Appendix G.3.

Theorem 2.2 (Lower Confidence Bound of J_{OA})**.** *Assuming that the evaluations of $i \in \mathcal{D}_{\mathrm{soft}}$ are independent, there exists a lower confidence bound for the masked composite objective $J_{\mathrm{OA}}(x)$ given by:*

$$\mathrm{LCB}_\delta(x) = \pi_{\mathrm{FV}}(x) \sum_{i \in \mathcal{D}_{\mathrm{soft}}} w_i \big(\widehat{\pi}_i(x) - m_i(\delta_i)\big) \tag{5}$$

with uncertainty level $\delta = 1 - \prod_{i \in \mathcal{D}_{\mathrm{soft}}} (1 - \delta_i)$.

Proof. See Appendix A. □

Theorem 2.2 establishes a lower confidence bound for the masked composite objective, enabling its use in an LCB-based optimization framework.

2.3 The Monotonic Optimization Process

We propose a *monotonic* process $\{x_t\}$ that gradually generates candidate formalizations and selects the one most likely to achieve the highest quality.

2.3.1 CANDIDATE GENERATION

Let $\mathcal{C}_t \subseteq \mathcal{X}$ denote the set of candidate formalizations obtained at step t. Maintaining diversity within $\mathcal{C}_t$ is crucial for covering the space of possible formalizations as comprehensively as possible. We define $\mathcal{G} = \{G_1, \ldots, G_M\}$ as the set of generators to sample candidate formalizations from $\mathcal{X}$. We classify generators into three categories:

One-Off Generator (OOG). The one-off generator G assigns a fixed distribution over samples conditioned on $(s_{\mathrm{nl}}, p_{\mathrm{nl}})$, generating outputs according to $x \sim p_G(x \mid s_{\mathrm{nl}}, p_{\mathrm{nl}})$.

FV-Repairer (FVR). Given that the objective places the highest priority on formal validity, the FV-Repairer G is designed to correct instances in which $\pi_{\mathrm{FV}}(x) = 0$, such that $\Pr\Big(\pi_{\mathrm{FV}}\big(G(x)\big) = 1\Big) > 0$.

Recurrent Generator (REG). The recurrent generator G assigns a fixed distribution over samples conditioned on $(s_{\mathrm{nl}}, p_{\mathrm{nl}})$ and a previous candidate formalization x_{t-1}, generating outputs according to $x_t \sim p_G(x \mid s_{\mathrm{nl}}, p_{\mathrm{nl}}, x_{t-1})$.

In practice, the one-off generator can rely on few-shot prompting (Wu et al., 2022) or retrieval-augmented generation (Zhang et al., 2025a; Liu et al., 2025a). The FV-repairer may be implemented using either LLM-based approaches (Zhang et al., 2024) or rule-based systems (Zhang et al., 2025a). The recurrent generator can be instantiated via refinement-based methods (Zhang et al., 2025b). When using prompt-based refinement methods either in FVR settings with feedback from theorem prover, or REG settings with feedback from LLM judges, we propose a method to model each generator's dimension-wise response to the feedback.

Definition 2.3 (Responsiveness Map)**.** For generator G_m which leverages LLM with feedback FB, we define the *local responsiveness* for dimension $i \in \{\mathrm{FV}\} \cup \mathcal{D}_{\mathrm{soft}}$ at state x as

$$\rho_i(G_m \mid x) = \mathbb{E}_{x' \sim \mathrm{LLM}(x;\mathrm{FB}(x))}\big[\pi_i(x') - \pi_i(x)\big]. \tag{6}$$

The responsiveness map captures the expected local gain for each dimension. For the FV dimension, this quantity is equivalent to the probability that the generator fixes formal validity when it is configured as the FV-Repairer. When G_m proposes $B_m^{(t)}$ candidates $x_t^{(m,b)}$ at each time step t, the estimation of responsiveness of generator G_m at step t for $i \in \mathcal{D}_{\mathrm{soft}}$ could be achieved by aggregated differences: $\widehat{\rho}_i(G_m \mid x_t) = \frac{1}{B_m^{(t)}} \sum_{b=1}^{B_m^{(t)}} \big(\widehat{\pi}_i(x_t^{(m,b)}) - \widehat{\pi}_i(x_t)\big)$ with some variance introduced by LLM judges, and for FV, $\widehat{\rho}_{\mathrm{FV}}(G_m \mid x_t) = \frac{1}{B_m^{(t)}} \sum_{b=1}^{B_m^{(t)}} \big(\pi_{\mathrm{FV}}(x_t^{(m,b)}) - \pi_{\mathrm{FV}}(x_t)\big)$. For sample efficiency, the proposal budgets should be set proportionally to generator G_m with better responsiveness maps.

An example of sampling candidate formalizations from different generators is provided in Algorithm 1 in Appendix D. We also provide a concrete process of how to determine the generator roles for a set of candidate LLMs in Appendix G.1.

2.3.2 THE ACCEPTANCE POLICY

We first define a plug-in estimator which indicates the overall performance as:

$$\widehat{J}(x) = \max\big\{\pi_{\mathrm{FV}}(x), \epsilon\big\} \sum_{i \in \mathcal{D}_{\mathrm{soft}}} w_i \widehat{\pi}_i(x), \tag{7}$$

where $\epsilon > 0$ is a sufficiently small number such that $\epsilon < \min(\{m_i(\delta_i)\} \cup \{w_{i,k}\})$. Assumption 2.1 provides a theoretical guarantee that $\widehat{J}(x)$ is asymptotically stable and can therefore be treated as a deterministic value and used for reliable comparison. We propose the following acceptance policy:

Conservative Acceptance Policy. Assume $\widehat{J}(x_0) < 1$. We define the best candidate as:

$$x_{t+1}^{\star} = \arg\max_{x \in \mathcal{C}_t} \widehat{J}(x). \tag{8}$$

The acceptance rule at step $t+1$ is designed as:

$$x_{t+1} = \begin{cases} x^\star_{t+1}, & \text{if} \quad \widehat{J}(x^\star_{t+1}) > \widehat{J}(x_t) \\ x_t, & \text{else} \end{cases}. \tag{9}$$

The process applying this acceptance rule is illustrated in Algorithm 2 in Appendix D. Such a process $\{x_t\}$ is certified to exhibit monotonic progress by the following theorem:

Theorem 2.4. *For all t, we have $J_{\mathrm{OA}}(x_{t+1}) \geq J_{\mathrm{OA}}(x_t)$ or $\mathrm{LCB}_\delta(x_{t+1}) > \mathrm{LCB}_\delta(x_t)$.*

Proof. See Appendix B. □

Theorem 2.4 indicates that, at each step of the process, either the true objective does not decrease or its lower confidence bound increases. Hence, the process guarantees non-negative progress under this criterion, thereby promoting improvement in autoformalization.

Theorem 2.5. *For the monotonic process $\{x_t\}$ defined above, we have:*

(i) (Convergence) $\exists t_0, \forall t > t_0, \mathrm{LCB}_\delta(x_{t+1}) \geq \mathrm{LCB}_\delta(x_t)$ and $\{\mathrm{LCB}_\delta(x_t)\}$ converges;

(ii) (Process Terminability) If there exists $x \in \bigcup_t \mathcal{C}_t$ such that $\widehat{J}(x) = 1$, then there exists an optimal candidate $x^* \in \bigcup_t \mathcal{C}_t$ such that $\exists t_0, \forall t > t_0, x_t = x^*$ and $\Pr\left(1 - \sum_{i \in \mathcal{D}_{\mathrm{soft}}} w_i m_i(\delta_i) \leq J_{\mathrm{OA}}(x^*)\right) \geq 1 - \delta$.

Proof. See Appendix C. □

Theorem 2.5 implies that after a sufficiently large number of steps, the change in the lower confidence bound becomes incremental, and the bound tends to stabilize. Moreover, if the plug-in estimator prioritizes a candidate (e.g., an ideal formalization), the process selects a strong candidate whose true objective value exceeds a certain threshold with probability at least $1 - \delta$.

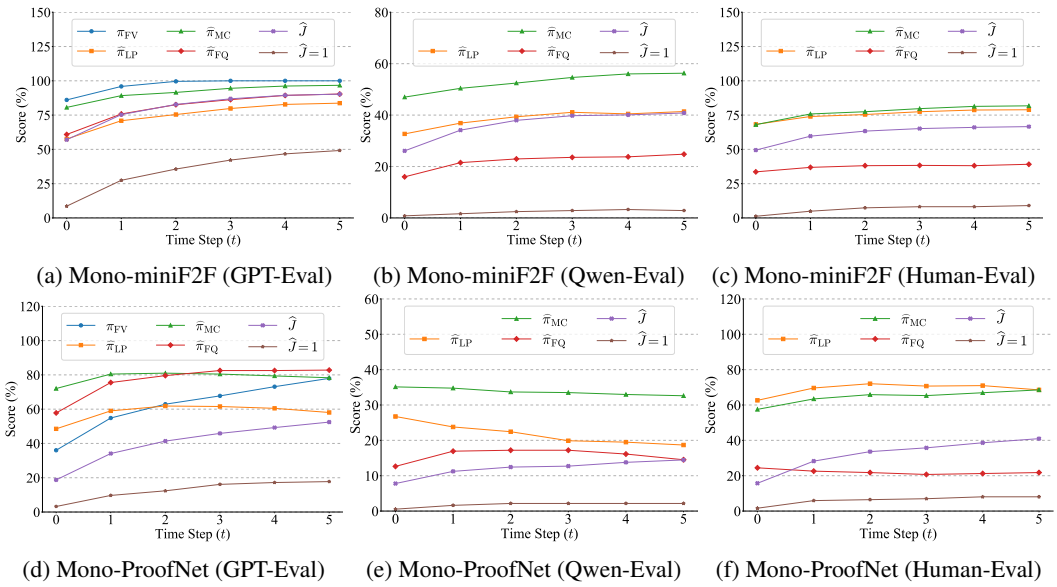
3 The Monotonic Process

3.1 Setup

Lean4 (de Moura et al., 2015) (version v4.19.0-rc3 with REPL) is chosen as the formal language for all experiments. We evaluate our approach on miniF2F-Test (Zheng et al., 2022; Jiang et al., 2023) and ProofNet-Test (Azerbayev et al., 2023), as both benchmarks provide natural-language proofs. DeepSeek-Prover-V2-7B (Ren et al., 2025), Goedel-Prover-V2-8B (Lin et al., 2025), Seed-Coder-8B (Seed et al., 2025), Qwen2.5-7B (Qwen-Team, 2025), and GPT series (OpenAI, 2024; 2025) are selected as candidate LLMs for our experiments. The description and rationale behind the selection of benchmarks and models are provided in Appendix E.

We select Logical Preservation (LP), Mathematical Consistency (MC), and Formal Quality (FQ) as the target soft dimensions for full-theorem autoformalization in our experiments. For the fused estimators in Eq. 3, we adopt the OAP setting from (Zhang et al., 2025c), as it naturally fits the formulation in Eq. 3 while providing better efficiency. For simplicity, we assign equal weights, setting $w_i = \frac{1}{3}$ for each soft dimension, since determining the relative importance of different soft dimensions is a separate problem beyond the scope of this paper. We set $\epsilon = 0.001$ in Eq. 7.

On the evaluation front, we use the theorem prover-derived score π_{FV} to assess formal validity. For the soft dimensions, we report $\widehat{\pi}_{\mathrm{LP}}$, $\widehat{\pi}_{\mathrm{MC}}$, and $\widehat{\pi}_{\mathrm{FQ}}$, as assessed by for-acceptance judges (GPT-4.1-mini), external LLM judges (Qwen2.5-Coder-7B), and human judges. This evaluation setting allows us to examine whether using the same LLM judges both during the formalization process and for final evaluation introduces evaluator bias or inflates the measured performance. We also report their aggregate score, $\widehat{J}$, as an overall evaluation metric. To provide a stricter measure of performance, we additionally report the proportion of perfect formalizations, defined as those satisfying $\widehat{J} = 1$, i.e., receiving a score of 1 on every dimension.



(a) Mono-miniF2F (GPT-Eval) (b) Mono-miniF2F (Qwen-Eval) (c) Mono-miniF2F (Human-Eval)

(d) Mono-ProofNet (GPT-Eval) (e) Mono-ProofNet (Qwen-Eval) (f) Mono-ProofNet (Human-Eval)

Figure 2: Performance of the monotonic process using an ensemble of LLMs acting different generator roles. The evaluators of soft-dimensions includes: (**GPT-Eval**): GPT-4.1-mini judges; (**Qwen-Eval**): Qwen2.5-Coder-7B judges; (**Human-Eval**): one human expert.

We employ Algorithm 1 in Appendix D to construct candidate formalizations at each step of the monotonic optimization process. DeepSeek-Prover-7B and Goedel-Prover-8B are selected as the One-Off Generators and FV Repairers. For Recurrent Generators, we consider refinement using feedback on individual soft dimensions from LLM judges (REG-LP, REG-MC, and REG-FQ). Specifically, we use Seed-Coder-8B with REG-LP, Qwen2.5-7B with REG-LP, and DeepSeek-Prover-7B with REG-FQ in the monotonic process. Further discussion of this setup is provided in Appendix G.1.

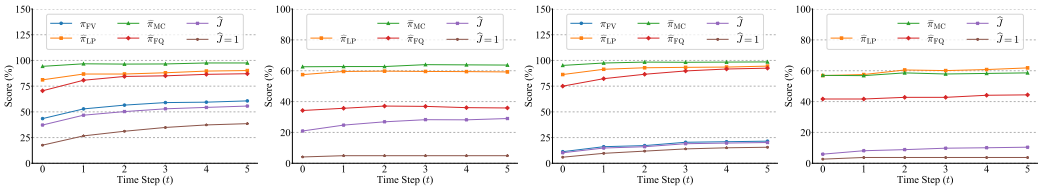
3.2 Main Results

We run the monotonic optimization process for six iterations on both miniF2F and ProofNet. Performance at each iteration is illustrated in Figure 2 with detailed numbers provided in Table 4 in Appendix K.

The monotonic process shows non-decreasing progress and converges within a few steps. In Figure 2, across both datasets, the overall evaluation scores consistently improve as the process proceeds. On miniF2F, the gains become relatively incremental after the fourth time step, indicating that the process tends to converge. At the final time step, the monotonic process achieves formal validity of 100.00%, an overall score of 90.27%, and 49.18% perfect formalizations on miniF2F (Figure 2a). On the much harder benchmark ProofNet, the monotonic process leads to 77.96% formal validity, 52.45% overall score, and 17.74% perfect formalizations (Figure 2d).

Different evaluation dimensions exhibit distinct trends throughout the monotonic process. On ProofNet, while formal validity improves across all steps, some soft-dimensions tend to stabilize or fluctuate after the first three iterations on both datasets. However, this discrepancy still leads to a gradual increase in the overall objective, as validated by human evaluation, suggesting that the process produces a set of auto-formalizations with progressively higher quality. Such improvements indicate the potential of this approach to support mathematical discovery and to generate high-quality formalization data for further training of existing models.

Monotonic improvement is not solely an artifact of using the same LLM judges for both acceptance and evaluation. The monotonic process continues to exhibit a near non-decreasing trend and convergence when evaluated with LLM judges from a different model family (Figure 2b, 2e). While bias may arise from LLM-based evaluation, this observation suggests that the improvements are not purely driven by the preferences of a particular judge, but instead reflect a more consis-



(a) miniF2F (GPT-Eval) (b) miniF2F (Qwen-Eval) (c) ProofNet (GPT-Eval) (d) ProofNet (Qwen-Eval)

Figure 3: Performance of the monotonic process with only GPT-5.4 as OOG and FVR.

tent improvement behavior of the proposed monotonic process. Human evaluation provides further evidence supporting the monotonic improvement of the proposed process (Figure 2c, 2f).

3.3 Ablation Studies

We first use a single strong LLM (GPT-5.4) as both the OOG and the FVR in the monotonic process. The results are reported in Figure 3. **The gains from the monotonic process do not inherently depend on an ensemble of multiple LLMs, however, an ensemble of smaller LLMs can outperform a single larger model.** As shown in Figures 3a and 3c, applying the monotonic process with GPT-5.4 yields consistent improvements across all evaluated dimensions. These improvements are further corroborated by external judges (Figures 3b and 3d). Collectively, these results show that the proposed monotonic process can induce consistent iterative improvements even when instantiated with a single LLM, supporting the generality of the underlying optimization framework. However, both formal validity and overall performance are notably lower than when multiple LLMs are assigned to distinct roles. This result further supports our design choice of role specialization within the monotonic process.

We further compare against iterative self-refinement (ISR) using three LLMs independently (DeepSeek-Prover-7B, Goedel-Prover-8B, and GPT-5.4), where each model receives error feedback from the theorem prover but no acceptance policy is applied. The results are reported in Table 6 in Appendix K. **Monotonic improvement does not emerge simply from repeatedly refining a formalization with theorem-prover feedback.** Under ISR with DeepSeek-Prover-7B and Goedel-Prover-8B, the scores fluctuate across iterations, with no clear trend of consistent improvement. Although ISR with GPT-5.4 exhibits some degree of monotonicity in formal validity and overall performance, the improvement is substantially weaker than that achieved by the monotonic process with GPT-5.4.

3.4 Comparison with Other Methods

We present a comparative evaluation of different full-theorem generation methods, using a policy that selects, for each sample, the formalization with the highest $\widehat{J}$. We consider two generation baselines with DeepSeek-Prover-7B, Goedel-Prover-8B, and GPT-5.4: iterative self-refinement (ISR) and sampling k independent outputs from a single model (analogous to Pass@k). The results are reported in Table 1.

Best-of-k is more effective for models with less capability, whereas Best-of-ISR is more effective for those with large capability. For smaller or weaker base models such as DeepSeek-Prover-7B and Goedel-Prover-8B, Best-of-k significantly outperforms Best-of-ISR in terms of the final score. In contrast, for stronger models such as GPT-5.4, which have shown explicit capability for refinement, Best-of-ISR performs better. This suggests that sampling diversity combined with post-hoc selection is more effective than ISR when the underlying model has limited reasoning capability, whereas ISR benefits more capable models.

The monotonic process demonstrates superiority for full-theorem autoformalization. Even in the single-model configuration, our proposed monotonic process yields robust and consistent improvements across multiple metrics compared to both Best-of-ISR and Best-of-k. Most importantly, the monotonic process with an ensemble of smaller LLMs achieves the strongest or comparable performance across nearly all metrics. It attains the highest π_{FV} and substantially improves $\widehat{J}$, out-

Table 1: Performance of different methods. Best-of-ISR selects the best formalization based on $\widehat{J}$ across six iterations for each sample. Best-of-k ($k = 8$) selects the best formalization based on $\widehat{J}$ from k outputs of a single model. All numbers are reported as percentages. **Bold** and Underline mark the highest and second-highest scores.

	miniF2F-Test						ProofNet-Test					
t	π_{FV}	$\widehat{\pi}_{\mathrm{LP}}$	$\widehat{\pi}_{\mathrm{MC}}$	$\widehat{\pi}_{\mathrm{FQ}}$	$\widehat{J}$	$\widehat{J}=1$	π_{FV}	$\widehat{\pi}_{\mathrm{LP}}$	$\widehat{\pi}_{\mathrm{MC}}$	$\widehat{\pi}_{\mathrm{FQ}}$	$\widehat{J}$	$\widehat{J}=1$
DeepSeek-Prover-V2-7B												
Best-of-ISR	60.66	58.40	86.07	63.73	44.74	11.89	18.28	50.94	78.85	55.11	12.24	3.76
Best-of-k	88.11	72.34	90.85	69.88	<u>68.78</u>	21.72	43.01	57.26	83.69	66.94	25.86	6.99
Goedel-Prover-V2-8B												
Best-of-ISR	82.38	51.33	80.19	68.44	57.49	6.15	36.56	37.23	63.98	64.25	19.94	3.23
Best-of-k	<u>90.57</u>	62.50	86.34	68.44	66.16	9.43	<u>54.84</u>	47.04	71.15	61.02	<u>27.27</u>	3.76
GPT-5.4												
Best-of-ISR	56.97	87.60	97.27	85.45	51.88	32.79	20.43	<u>95.43</u>	**98.92**	<u>91.94</u>	19.54	14.52
Best-of-k	49.59	**90.37**	<u>97.40</u>	<u>88.93</u>	45.24	32.79	15.59	**96.91**	98.57	91.40	14.46	10.22
Monotonic Process (Ours)												
Single (GPT-5.4)	60.66	<u>90.16</u>	**97.54**	87.09	55.64	<u>38.52</u>	21.51	94.49	<u>98.57</u>	**92.47**	20.19	<u>15.59</u>
Ensemble	**100.00**	83.71	96.72	**90.37**	**90.27**	**49.18**	**77.96**	58.06	78.32	82.80	**52.45**	**17.74**

performing both Best-of-k and Best-of-ISR across all settings by a large margin. It also achieves the best $\widehat{J} = 1$ scores, indicating improved rates of perfect autoformalization.

Computational cost vs performance gain. We report the computational costs of different settings, measured by the number of LLM calls, in Table 7 (Appendix K). In the single-model setting, although the monotonic process requires more generator calls, it uses fewer judge calls and achieves higher performance, especially in producing high-quality formalizations (as indicated by $\widehat{J} = 1$). The monotonic process with an ensemble of LLMs incurs substantially higher computational cost, however, it relies on smaller LLMs, and the cost decreases progressively over iterations. Given its superior performance, we consider this a reasonable trade-off between computational cost and performance.

4 Related Work

Recent advances in LLMs have renewed interest in autoformalization, which bridges informal mathematical language and formal proof systems (Wu et al., 2022; Yang et al., 2025; Mensfelt et al., 2025; Zhang et al., 2025a; Wu et al., 2025; Xuejun et al., 2025). Natural language proofs have further enabled automated theorem-proving pipelines (Jiang et al., 2023; Tarrach et al., 2024; Liu et al., 2025b), but integrating such reasoning directly into autoformalization remains challenging. Retrieval-augmented methods improve correctness by leveraging existing formal libraries in Isabelle and Lean (Zhang et al., 2024; Yang et al., 2023; Liu et al., 2025a; Wang et al., 2025; Zhang et al., 2025d), but are inherently reference-based. In contrast, we propose a reference-free framework. Prior process- and agent-based approaches (Lu et al., 2024; Zhang et al., 2025b) and self-consistency methods (Li et al., 2024) rely on heuristic designs, whereas our method is grounded in a theoretically justified monotonic optimization process. An extended related work is provided in Appendix F.

5 Conclusion

We present an iterative monotonic framework for full-theorem autoformalization that combines a masked multi-dimensional objective, role-specialized LLMs guided by a responsiveness map, and a principled acceptance policy. The framework provides a certified notion of monotonic progress, ensuring at each iteration that either the true objective does not decrease or its lower confidence bound increases, under the stated assumptions on judge uncertainty. Experiments on miniF2F and ProofNet demonstrate that our approach effectively integrates the complementary strengths of general-purpose and specialized LLMs, achieving high formal validity while improving overall formalization quality. This work offers a practical and theoretically grounded step toward more reliable autoformalization, with future directions including the design of more effective test-time LLM allocation policies.

AI USE STATEMENT

In this work, we used generative AI tools for [formulate mathematical claims, assist with translation, provide feedback on research experiments]. We have not used generative AI tools for [help develop theoretical models or conceptual frameworks, provide critical ingredients for proving mathematical claims, assist in the writing of proofs, propose or refine hypotheses, design research methodology or experiments, implement methods, support qualitative and thematic data analysis, interpret results], and [generate synthetic data sets, clean and reformat dataset] are not applicable to this work. Additionally, we used generative AI tools for [sourcing/searching for information, edit a research paper to improve readability, propose a title or keywords for a research paper]. We have reviewed all AI-assisted work. All text generated by LLMs was inspected to ensure that it conveyed the intended meaning. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We provide detailed information to support the reproducibility of our theoretical and empirical results. The assumptions underlying our theoretical analysis are stated in Section 2.2, and complete proofs of the main theoretical results are provided in Appendices A, B, C. The algorithms used for candidate construction and the monotonic optimization process are specified in Appendix D. Our experimental setup, including the Lean4 version, evaluation benchmarks, candidate LLMs, soft-dimension definitions, and evaluation protocol, is described in Section 3.1, with additional rationale for benchmark and model selection provided in Appendix E. The generator-role assignments and decoding settings are further detailed in Appendix G, in particular, open-source models use greedy decoding to improve reproducibility. The prompts and atomic properties used for LLM-based evaluation are provided in Appendix H. Finally, detailed experimental results and computational costs, including the number of generator and judge calls, are reported in Appendix K.

REFERENCES

Zhangir Azerbayev, Bartosz Piotrowski, Hailey Schoelkopf, Edward W. Ayers, Dragomir Radev, and Jeremy Avigad. Proofnet: Autoformalizing and formally proving undergraduate-level mathematics, 2023. URL https://arxiv.org/abs/2302.12433.

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

Leonardo de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn, and Jakob von Raumer. The lean theorem prover (system description). In Amy P. Felty and Aart Middeldorp (eds.), *Automated Deduction - CADE-25*, pp. 378–388, Cham, 2015. Springer International Publishing.

Guoxiong Gao, Yutong Wang, Jiedong Jiang, Qi Gao, Zihan Qin, Tianyi Xu, and Bin Dong. Herald: A natural language annotated lean 4 dataset. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=Se6MgCtRhz.

Albert Qiaochu Jiang, Sean Welleck, Jin Peng Zhou, Timothee Lacroix, Jiacheng Liu, Wenda Li, Mateja Jamnik, Guillaume Lample, and Yuhuai Wu. Draft, sketch, and prove: Guiding formal theorem provers with informal proofs. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=SMa9EAovKMC.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks. In

H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 9459–9474. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/6b493230205f780e1bc26945df7481e5-Paper.pdf.

Zenan Li, Yifan Wu, Zhaoyu Li, Xinming Wei, Xian Zhang, Fan Yang, and Xiaoxing Ma. Autoformalize mathematical statements by symbolic equivalence and semantic consistency. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=8ihVBYpMV4.

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, 2025. URL https://arxiv.org/abs/2508.03613.

Qi Liu, Xinhao Zheng, Xudong Lu, Qinxiang Cao, and Junchi Yan. Rethinking and improving autoformalization: towards a faithful metric and a dependency retrieval-based approach. In *The Thirteenth International Conference on Learning Representations*, 2025a. URL https://openreview.net/forum?id=hUb2At2DsQ.

Qi Liu, Xinhao Zheng, Renqiu Xia, Qinxiang Cao, and Junchi Yan. Bootstrapping hierarchical autoregressive formal reasoner with chain-of-proxy-autoformalization. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025b. URL https://openreview.net/forum?id=2Xn8h68mP3.

Jianqiao Lu, Yingjia Wan, Zhengying Liu, Yinya Huang, Jing Xiong, Chengwu Liu, Jianhao Shen, Hui Jin, Jipeng Zhang, Haiming Wang, Zhicheng Yang, Jing Tang, and Zhijiang Guo. Process-driven autoformalization in lean 4, 2024. URL https://arxiv.org/abs/2406.01940.

Shuai Lu, Nan Duan, Hojae Han, Daya Guo, Seung-won Hwang, and Alexey Svyatkovskiy. ReACC: A retrieval-augmented code completion framework. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (eds.), *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 6227–6240, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.431. URL https://aclanthology.org/2022.acl-long.431.

Agnieszka Mensfelt, David Tena Cucala, Santiago Franco, Angeliki Koutsoukou-Argyraki, Vince Trencsenyi, and Kostas Stathis. Towards a common framework for autoformalization, 2025. URL https://arxiv.org/abs/2509.09810.

Bartosz Naskrecki and Ken Ono. Mathematical discovery in the age of artificial intelligence. *Nat. Phys.*, 21(10):1504–1506, October 2025. doi: 10.1038/s41567-025-03042-0. URL https://www.nature.com/articles/s41567-025-03042-0.

Jenny Lorraine Nielsen. Conne's rigidity theorem: Disproof of the open ai and anthropic counterexamples (with proposed proof of the conjecture). PhilArchive, 2026. URL https://philarchive.org/rec/NIEWTC.

OpenAI. Gpt-4 technical report, 2024. URL https://arxiv.org/abs/2303.08774.

OpenAI. Openai gpt-5 system card, 2025. URL https://arxiv.org/abs/2601.03267.

OpenAI. Ten advances in mathematics and theoretical computer science, August 2026. URL https://cdn.openai.com/pdf/ten-proofs-oai.pdf.

Lawrence C. Paulson. Isabelle: The next 700 theorem provers, 2000.

Ivo Petrov, Jasper Dekoninck, Lyuben Baltadzhiev, Maria Drencheva, Kristian Minchev, Mislav Balunović, Nikola Jovanović, and Martin Vechev. Proof or bluff? evaluating llms on 2025 usa math olympiad, 2025. URL https://arxiv.org/abs/2503.21934.

Qwen-Team. Qwen2.5 technical report, 2025. URL https://arxiv.org/abs/2412.15115.

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.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M Pawan Kumar, Emilien Dupont, Francisco J R Ruiz, Jordan S Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, January 2024. doi: 10.1038/s41586-023-06924-6. URL https://www.nature.com/articles/s41586-023-06924-6.

ByteDance Seed, Yuyu Zhang, Jing Su, Yifan Sun, Chenguang Xi, Xia Xiao, Shen Zheng, Anxiang Zhang, Kaibo Liu, Daoguang Zan, Tao Sun, Jinhua Zhu, Shulin Xin, Dong Huang, Yetao Bai, Lixin Dong, Chao Li, Jianchong Chen, Hanzhi Zhou, Yifan Huang, Guanghan Ning, Xierui Song, Jiaze Chen, Siyao Liu, Kai Shen, Liang Xiang, and Yonghui Wu. Seed-coder: Let the code model curate data for itself, 2025. URL https://arxiv.org/abs/2506.03524.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024. URL https://arxiv.org/abs/2402.03300.

Zhihong Shao, Yuxiang Luo, Chengda Lu, Z. Z. Ren, Jiewen Hu, Tian Ye, Zhibin Gou, Shirong Ma, and Xiaokang Zhang. Deepseekmath-v2: Towards self-verifiable mathematical reasoning, 2025. URL https://arxiv.org/abs/2511.22570.

Guillem Tarrach, Albert Q. Jiang, Daniel Raggi, Wenda Li, and Mateja Jamnik. More details, please: Improving autoformalization with more detailed proofs. In *AI for Math Workshop @ ICML 2024*, 2024. URL https://openreview.net/forum?id=AkJvzpYMvK.

Shaoqi Wang, Lu Yu, Siwei Lou, Feng Yan, Chunjie Yang, Qing Cui, and Jun Zhou. Improving autoformalization using direct dependency retrieval, 2025. URL https://arxiv.org/abs/2511.11990.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=1PL1NIMMrw.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed H. Chi, Quoc V Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=_VjQlMeSB_J.

Yuhuai Wu, Albert Qiaochu Jiang, Wenda Li, Markus Norman Rabe, Charles E Staats, Mateja Jamnik, and Christian Szegedy. Autoformalization with large language models. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=IUikebJ1Bf0.

Yutong Wu, Di Huang, Ruosi Wan, Yue Peng, Shijie Shang, Chenrui Cao, Lei Qi, Rui Zhang, Zidong Du, Jie Yan, and Xing Hu. Stepfun-formalizer: Unlocking the autoformalization potential of llms through knowledge-reasoning fusion, 2025. URL https://arxiv.org/abs/2508.04440.

Yu Xuejun, Jianyuan Zhong, Zijin Feng, Pengyi Zhai, Roozbeh Yousefzadeh, Wei Chong Ng, Haoxiong Liu, Ziyi Shou, Jing Xiong, Yudong Zhou, Claudia Beth Ong, Austen Jeremy Sugiarto, Yaoxi Zhang, Wai Ming Tai, Huan Cao, Dongcai Lu, Jiacheng Sun, Qiang Xu, Shen Xin, and Zhenguo Li. Mathesis: Towards formal theorem proving from natural languages, 2025. URL https://arxiv.org/abs/2506.07047.

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. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 21573–21612. Curran Associates, Inc., 2023. URL https://openreview.net/forum?id=g7OX2sOJtn.

Kaiyu Yang, Gabriel Poesia, Jingxuan He, Wenda Li, Kristin E. Lauter, Swarat Chaudhuri, and Dawn Song. Position: Formal mathematical reasoning—a new frontier in AI. In *Forty-second International Conference on Machine Learning Position Paper Track*, 2025. URL https://openreview.net/forum?id=HuvAM5x2xG.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik R Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=5Xc1ecxO1h.

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 *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024. URL https://openreview.net/forum?id=Vcw3vzjHDb.

Zhouliang Yu, Ruotian Peng, Keyi Ding, Yizhe Li, Zhongyuan Peng, Minghao Liu, Yifan Zhang, Zheng Yuan, Huajian Xin, Wenhao Huang, Yandong Wen, Ge Zhang, and Weiyang Liu. Formalmath: Benchmarking formal mathematical reasoning of large language models, 2025. URL https://arxiv.org/abs/2505.02735.

Kun Zhang, Xiexiong Lin, Yuanzhuo Wang, Xin Zhang, Fei Sun, Cen Jianhe, Hexiang Tan, Xuhui Jiang, and Huawei Shen. ReFSQL: A retrieval-augmentation framework for text-to-SQL generation. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 664–673, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-emnlp.48. URL https://aclanthology.org/2023.findings-emnlp.48.

Lan Zhang, Xin Quan, and Andre Freitas. Consistent autoformalization for constructing mathematical libraries. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 4020–4033, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.233. URL https://aclanthology.org/2024.emnlp-main.233/.

Lan Zhang, Marco Valentino, and Andre Freitas. Autoformalization in the wild: Assessing LLMs on real-world mathematical definitions. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 1720–1738, Suzhou, China, November 2025a. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.90. URL https://aclanthology.org/2025.emnlp-main.90/.

Lan Zhang, Marco Valentino, and Andre Freitas. MASA: LLM-driven multi-agent systems for autoformalization. In Ivan Habernal, Peter Schulam, and Jörg Tiedemann (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 615–624, Suzhou, China, November 2025b. Association for Computational Linguistics. ISBN 979-8-89176-334-0. doi: 10.18653/v1/2025.emnlp-demos.44. URL https://aclanthology.org/2025.emnlp-demos.44/.

Lan Zhang, Marco Valentino, and Andre Freitas. Beyond gold standards: Epistemic ensemble of llm judges for formal mathematical reasoning, 2025c. URL https://arxiv.org/abs/2506.10903.

Meiru Zhang, Philipp Borchert, Milan Gritta, and Gerasimos Lampouras. Drift: Decompose, retrieve, illustrate, then formalize theorems, 2025d. URL https://arxiv.org/abs/2510.10815.

Ziyin Zhang, Jiahao Xu, Zhiwei He, Tian Liang, Qiuzhi Liu, Yansi Li, Linfeng Song, Zhenwen Liang, Zhuosheng Zhang, Rui Wang, Zhaopeng Tu, Haitao Mi, and Dong Yu. Deeptheorem: Advancing llm reasoning for theorem proving through natural language and reinforcement learning, 2025e. URL https://arxiv.org/abs/2505.23754.

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. URL https://openreview.net/forum?id=9ZPegFuFTFv.

A Proof of Theorem 2.2

Proof. Recall:

$$J_{\mathrm{OA}}(x) = \pi_{\mathrm{FV}}(x) \sum_{i \in \mathcal{D}_{\mathrm{soft}}} w_i \pi_i(x)$$
$$\mathrm{LCB}_\delta(x) = \pi_{\mathrm{FV}}(x) \sum_{i \in \mathcal{D}_{\mathrm{soft}}} w_i \big(\widehat{\pi}_i(x) - m_i(\delta_i)\big)$$
$$\delta = 1 - \prod_{i \in \mathcal{D}_{\mathrm{soft}}} (1 - \delta_i)$$

Let $d = |\mathcal{D}_{\mathrm{soft}}|$. We have:

$$\begin{aligned}
&\Pr\Big(\mathrm{LCB}_\delta(x) \le J_{\mathrm{OA}}(x)\Big) \\
&\ge \Pr\Big(\sum_{i \in \mathcal{D}_{\mathrm{soft}}} w_i\big(\widehat{\pi}_i(x) - m_i(\delta_i)\big) \le \sum_{i \in \mathcal{D}_{\mathrm{soft}}} w_i \pi_i(x)\Big) \\
&\ge \Pr\Big(w_1\big(\widehat{\pi}_1(x) - m_1(\delta_1)\big) \le w_1\pi_1(x), \ldots, w_d\big(\widehat{\pi}_d(x) - m_d(\delta_d)\big) \le w_d \pi_d(x)\Big) \\
&= \Pr\Big(\widehat{\pi}_1(x) \le \pi_1(x) + m_1(\delta_1), \ldots, \widehat{\pi}_d(x) \le \pi_d(x) + m_d(\delta_d)\Big).
\end{aligned} \tag{10}$$

Using the assumption of independence and Eq. 4, we have:

$$\begin{aligned}
&\Pr\Big(\widehat{\pi}_1(x) \le \pi_1(x) + m_1(\delta_1), \ldots, \widehat{\pi}_d(x) \le \pi_d(x) + m_d(\delta_d)\Big) \\
&= \prod_{i \in \mathcal{D}_{\mathrm{soft}}} \Pr\Big(\widehat{\pi}_i(x) \le \pi_i(x) + m_i(\delta_i)\Big) = \prod_{i \in \mathcal{D}_{\mathrm{soft}}} \Pr\Big(\widehat{\pi}_i(x) - m_i(\delta_i) \le \pi_i(x)\Big) \\
&\ge \prod_{i \in \mathcal{D}_{\mathrm{soft}}} (1 - \delta_i) = 1 - \delta.
\end{aligned} \tag{11}$$

Combining Eq. 10 and 11, we have: $\Pr\Big(\mathrm{LCB}_\delta(x) \le J_{\mathrm{OA}}(x)\Big) \ge 1 - \delta$, meaning $\mathrm{LCB}_\delta(x)$ is a lower confidence bound for $J_{\mathrm{OA}}(x)$ with confidence level $1 - \delta$. □

B Proof of Theorem 2.4

Proof. Recall:

$$\widehat{J}(x) = \max\big\{\pi_{\mathrm{FV}}(x), \epsilon\big\} \sum_{i \in \mathcal{D}_{\mathrm{soft}}} w_i \widehat{\pi}_i(x)$$
$$x^\star_{t+1} = \arg\max_{x \in \mathcal{C}_t} \widehat{J}(x)$$
$$x_{t+1} = \begin{cases} x^\star_{t+1}, & \text{if} \quad \widehat{J}(x^\star_{t+1}) > \widehat{J}(x_t) \\ x_t, & \text{else} \end{cases}$$

When $x_{t+1} = x_t$, we have $J_{\mathrm{OA}}(x_{t+1}) = J_{\mathrm{OA}}(x_t) \ge J_{\mathrm{OA}}(x_t)$. When $x_{t+1} = x^\star_{t+1}$, we use proof by cases:

a. If $\pi_{\mathrm{FV}}(x_t) = 0$, we have $J_{\mathrm{OA}}(x_{t+1}) \geq 0 = J_{\mathrm{OA}}(x_t)$.

b. If $\pi_{\mathrm{FV}}(x_t) = 1, \pi_{\mathrm{FV}}(x^\star_{t+1}) = 0$, we have

$$\mathrm{LCB}_\delta(x_t) = \widehat{J}(x_t) - \sum_{i \in \mathcal{D}_{\mathrm{soft}}} w_i m_i(\delta_i) < \widehat{J}(x^\star_{t+1}) - \sum_{i \in \mathcal{D}_{\mathrm{soft}}} w_i m_i(\delta_i) \leq \epsilon - \sum_{i \in \mathcal{D}_{\mathrm{soft}}} w_i m_i(\delta_i). \quad (12)$$

Since ϵ is a sufficiently small number such that $\epsilon < \min\{m_i(\delta_i)\}$, it follows that

$$\epsilon - \sum_{i \in \mathcal{D}_{\mathrm{soft}}} w_i m_i(\delta_i) = \sum_{i \in \mathcal{D}_{\mathrm{soft}}} w_i \big(\epsilon - m_i(\delta_i)\big) < 0 = \mathrm{LCB}_\delta(x_{t+1}). \quad (13)$$

Hence, we have $\mathrm{LCB}_\delta(x_{t+1}) > \mathrm{LCB}_\delta(x_t)$.

c. If $\pi_{\mathrm{FV}}(x_t) = 1, \pi_{\mathrm{FV}}(x^\star_{t+1}) = 1$, we have $\mathrm{LCB}_\delta(x_{t+1}) - \mathrm{LCB}_\delta(x_t) = \widehat{J}(x^\star_{t+1}) - \widehat{J}(x_t) > 0$. □

C Proof of Theorem 2.5

Proof. (i) We first prove that $\exists t_0, \forall t > t_0, \mathrm{LCB}_\delta(x_{t+1}) \geq \mathrm{LCB}_\delta(x_t)$ by cases:

a. If $\exists t_0$ such that $\forall t > t_0, \pi_{\mathrm{FV}}(x_{t+1}) = \pi_{\mathrm{FV}}(x_t) = 0$, thereby $\forall t > t_0, \mathrm{LCB}_\delta(x_{t+1}) = \mathrm{LCB}_\delta(x_t) = 0$.

b. If $\forall t, \pi_{\mathrm{FV}}(x_{t+1}) = \pi_{\mathrm{FV}}(x_t) = 1$, then either $x_{t+1} = x_t$, in which case $\mathrm{LCB}_\delta(x_{t+1}) = \mathrm{LCB}_\delta(x_t)$, or $x_{t+1} = x^\star_{t+1}$, in which case $\mathrm{LCB}_\delta(x_{t+1}) - \mathrm{LCB}_\delta(x_t) = \widehat{J}(x^\star_{t+1}) - \widehat{J}(x_t) > 0$. In both cases, $\mathrm{LCB}_\delta(x_{t+1}) \geq \mathrm{LCB}_\delta(x_t)$, therefore, $\forall t, \mathrm{LCB}_\delta(x_{t+1}) \geq \mathrm{LCB}_\delta(x_t)$.

c. If $\exists t_1, \pi_{\mathrm{FV}}(x_{t_1}) = 1, \pi_{\mathrm{FV}}(x_{t_1+1}) = 0$ and $\exists t_2 > t_1, \pi_{\mathrm{FV}}(x_{t_2}) = 0, \pi_{\mathrm{FV}}(x_{t_2+1}) = 1$, we have $\widehat{J}(x_{t_2+1}) > \widehat{J}(x_{t_2}) \geq 0$, thereby $\widehat{J}(x_{t_2+1}) \geq \min\{w_{i,k}\} > \epsilon$. For every $t > t_2$, we next prove that if $\pi_{\mathrm{FV}}(x_t) = 1$ and $\widehat{J}(x_t) > \epsilon$, then $\pi_{\mathrm{FV}}(x_{t+1}) = 1$ and $\widehat{J}(x_{t+1}) > \epsilon$. Assume that $\pi_{\mathrm{FV}}(x_t) = 1$ and $\widehat{J}(x_t) > \epsilon$. Then either $x_{t+1} = x_t$, in which case $\widehat{J}(x_{t+1}) = \widehat{J}(x_t) > \epsilon$, or $x_{t+1} = x^\star_{t+1}$, in which case $\widehat{J}(x_{t+1}) = \widehat{J}(x^\star_{t+1}) > \widehat{J}(x_t) > \epsilon$. Moreover, $\pi_{\mathrm{FV}}(x_{t+1}) = 0$ is impossible, otherwise, we would have $\widehat{J}(x_{t+1}) \leq \epsilon$ which contradicts $\widehat{J}(x_{t+1}) > \epsilon$. Applying mathematical induction, we have $\forall t > t_2, \pi_{\mathrm{FV}}(x_t) = 1$. Using the same strategy as in case b, we have $\forall t > t_2, \mathrm{LCB}_\delta(x_{t+1}) \geq \mathrm{LCB}_\delta(x_t)$.

We have proved that $\{\mathrm{LCB}_\delta(x_t)\}_{t > t_0}$ is a non-decreasing sequence. Since $\{\mathrm{LCB}_\delta(x_t)\}$ are bounded by 1, the monotone convergence theorem implies that $\{\mathrm{LCB}_\delta(x_t)\}$ converges.

(ii) Since there exists $x \in \bigcup_t \mathcal{C}_t$ such that $\widehat{J}(x) = 1$, we have a non-empty set $\{t \mid \exists x \in \mathcal{C}_t, \widehat{J}(x) = 1\}$. Let $t_0 = \min\{t \mid \exists x \in \mathcal{C}_t, \widehat{J}(x) = 1\}$ and $x^* = x^\star_{t_0+1} = \arg\max_{x \in \mathcal{C}_{t_0}} \widehat{J}(x)$. We have $\widehat{J}(x_{t_0}) < 1 = \widehat{J}(x^\star_{t_0+1})$, thereby $x_{t_0+1} = x^\star_{t_0+1} = x^*$ and $\widehat{J}(x_{t_0+1}) = 1$. We next prove that if $\widehat{J}(x_t) = 1$, then $x_{t+1} = x_t$ and $\widehat{J}(x_{t+1}) = 1$. Assuming $\widehat{J}(x_t) = 1$, we have $\max_{x \in \mathcal{C}_t} \widehat{J}(x) \leq 1 = \widehat{J}(x_t)$. Hence, $\widehat{J}(x^\star_{t+1}) \leq 1 = \widehat{J}(x_t)$. Applying acceptance rule Eq. 9, we have $x_{t+1} = x_t$ and thereby $\widehat{J}(x_{t+1}) = \widehat{J}(x_t) = 1$. Applying mathematical induction, we have $\forall t > t_0, x_t = x^*$. Since $\widehat{J}(x^*) = 1$, we have $\pi_{\mathrm{FV}}(x^*) = 1$. Applying Eq. 5 and 7, we have $\mathrm{LCB}_\delta(x^*) = 1 - \sum_{i \in \mathcal{D}_{\mathrm{soft}}} w_i m_i(\delta_i)$. In other words, $\Pr\left(1 - \sum_{i \in \mathcal{D}_{\mathrm{soft}}} w_i m_i(\delta_i) \leq J_{\mathrm{OA}}(x^*)\right) \geq 1 - \delta$. □

D Algorithms

We provide the algorithm we used in practice to construct candidate formalizations at each time step in Algorithm 1, and the algorithm of the monotonic process in Algorithm 2.

E Rationale for Empirical Setups

We describe the existing formal benchmarks and the reasons for selecting miniF2F and ProofNet as follows:

Algorithm 1 Candidates Construction at Step t

Inputs: Natural language statement and proof $(s_{\mathrm{nl}}, p_{\mathrm{nl}})$, Previous formalization x_t, Set of One-Off Generators $\mathcal{G}_i$, Set of FV-Repairers $\mathcal{G}_j$, Set of Recurrent Generators $\mathcal{G}_k$, Theorem Prover $\mathcal{TP}$.
$\mathcal{C}_t \leftarrow \emptyset$
for $G \in \mathcal{G}_i$ **do**
  Sample one $x \sim p_G(x \mid s_{\mathrm{nl}}, p_{\mathrm{nl}})$
  if $\pi_{\mathrm{FV}}(x) == 0$ from $\mathcal{TP}$ **then**
    for $G' \in \mathcal{G}_j$ **do**
      Sample one $x' \sim p_{G'}(x' \mid s_{\mathrm{nl}}, p_{\mathrm{nl}}, x)$
      $\mathcal{C}_t \leftarrow \mathcal{C}_t \cup \{x'\}$
    end for
  else
    $\mathcal{C}_t \leftarrow \mathcal{C}_t \cup \{x\}$
  end if
end for
if $t > 0$ **then**
  for $G \in \mathcal{G}_k$ **do**
    Sample one $x \sim p_G(x \mid s_{\mathrm{nl}}, p_{\mathrm{nl}}, x_t)$
    if $\pi_{\mathrm{FV}}(x) == 0$ from $\mathcal{TP}$ **then**
      for $G' \in \mathcal{G}_j$ **do**
        Sample one $x' \sim p_{G'}(x' \mid s_{\mathrm{nl}}, p_{\mathrm{nl}}, x)$
        $\mathcal{C}_t \leftarrow \mathcal{C}_t \cup \{x'\}$
      end for
    else
      $\mathcal{C}_t \leftarrow \mathcal{C}_t \cup \{x\}$
    end if
  end for
end if

Algorithm 2 The Monotonic Process

Inputs: Natural language statements and proofs $\mathcal{D} = \{(s_{\mathrm{nl}}, p_{\mathrm{nl}})\}$, Soft dimensions $\mathcal{D}_{\mathrm{soft}}$, Sets of LLM Judges $\{\mathcal{J}_i\}_{i \in \mathcal{D}_{\mathrm{soft}}}$, Theorem Prover $\mathcal{TP}$, Maximum number of iterations T.
for $(s_{\mathrm{nl}}, p_{\mathrm{nl}}) \in \mathcal{D}$ **do**
  $x_0 \leftarrow$ "", $J_0 \leftarrow -1$
  for $t = 0, \ldots, T-1$ **do**
    Construct $\mathcal{C}_t$ (e.g., Algorithm 1)
    $J \leftarrow J_t$
    for $x \in \mathcal{C}_t$ **do**
      Obtain $\pi_{\mathrm{FV}}(x)$ from theorem prover $\mathcal{TP}$
      if $J \leq \epsilon$ or $\pi_{\mathrm{FV}}(x) == 1$ **then**
        for $i \in \mathcal{D}_{\mathrm{soft}}$ **do**
          Obtain estimation $\widehat{\pi}_i(x)$ from $\mathcal{J}_i$ using Eq. 3
        end for
        Calculate plug-in estimator $\widehat{J}(x)$ using Eq. 7
        if $\widehat{J}(x) \geq J$ **then**
          $x^\star_{t+1} \leftarrow x$, $J \leftarrow \widehat{J}(x)$
        end if
      end if
    end for
    if $J > J_t$ **then**
      $x_{t+1} \leftarrow x^\star_{t+1}$, $J_{t+1} \leftarrow J$
    else
      $x_{t+1} \leftarrow x_t$, $J_{t+1} \leftarrow J_t$
    end if
    if $J_{t+1} == 1$ **then**
      break
    end if
  end for
end for

- miniF2F (Zheng et al., 2022) is a dataset of formal Olympiad-level mathematics problems with both Isabelle (Paulson, 2000) and Lean formalizations. The original dataset lacks natural language proofs, but an improved version (Jiang et al., 2023) provides them. It contains 244 validation instances and 244 test instances. Due to the availability of natural language proofs and its relatively small size, miniF2F is considered as a representative benchmark for validating the methods in this work.
- ProofNet (Azerbayev et al., 2023) is a benchmark for autoformalization and formal proving of undergraduate-level mathematics, consisting of 185 validation samples and 186 test samples. Compared to miniF2F, it has higher complexity. Its combination of natural language proofs and manageable scale makes it suitable for evaluating our monotonic process.
- Lean Workbook (Ying et al., 2024) contains 57,231 formalized math problems (5k with formal solutions) generated via a pipeline of synthetic data generation and filtering. However, it lacks natural language proofs, making it unsuitable for the full-theorem autoformalization task in this work.
- FormalMATH (Yu et al., 2025) is a large-scale Lean4 benchmark with 5,560 formally verified problems ranging from high-school Olympiad challenges to undergraduate-level theorems. While it includes natural language proofs, its large scale makes it less practical as a representative benchmark for testing our methods in this paper.
- Herald (Gao et al., 2025) contains 579,883 samples generated from a structural-information-aware pipeline, of which 44,553 include natural language proofs. Similar to FormalMATH, although it provides natural language proofs, its large scale limits its suitability as a representative benchmark for this work.

We elicit the LLMs used in this work and the reasons for their selection as follows:

- DeepSeek-Prover-V2 (Ren et al., 2025) is an open-source LLM designed for formal theorem proving in Lean4, available in 7B and 671B sizes. This model family has demonstrated state-of-the-art performance in generating formal proofs (Chen et al., 2025). Since this work focuses on full-theorem autoformalization, which requires complete formalization including proofs, its superiority in generating Lean4 proofs makes it a suitable candidate. The 7B version is selected here due to computational resource constraints.
- Goedel-Prover-V2 (Lin et al., 2025) is another series of open-source LLMs for automated theorem proving in Lean4, available in 8B and 32B sizes. Unlike DeepSeek-Prover-V2, this model can iteratively revise its proofs by leveraging feedback from the Lean theorem prover, which is particularly useful in the FV-Repair setting of this work. We select the 8B version due to computational resource constraints.
- Qwen2.5 (Qwen-Team, 2025) is a comprehensive series of open-source models designed to meet diverse needs. Qwen2.5-7B-Instruct is a representative small-scale general-purpose model within our computational budget and is therefore included in this study.
- Seed-Coder (Seed et al., 2025) is a series of open-source LLMs specialized in code generation. While not fully general-purpose, formal languages can be considered a type of code, and thus this model serves as a state-of-the-art coding model suitable for our purposes. The 8B version is chosen due to computational resource limitations.
- GPT series (OpenAI, 2024; 2025) are the latest and largest state-of-the-art general-purpose closed-source LLM at the time of this work. Full versions of GPT-4.1 and GPT-5.4 are used for generating formalizations, while the smaller GPT-4.1-mini balances performance and cost, making it suitable as LLM judges in our experiments.

F Extended Related Work

Autoformalization Autoformalization refers to the automatic translation of natural-language mathematics into formal mathematics (Wu et al., 2022; Yang et al., 2025; Mensfelt et al., 2025). Most research in this area has focused on statement formalization, including theorem autoformalization (Zhang et al., 2024; Lu et al., 2024; Li et al., 2024; Gao et al., 2025) and definition autoformalization (Zhang et al., 2025a). On the proof autoformalization side, some works develop pipelines

for generating formal proofs (Ying et al., 2024; Gao et al., 2025), however, the quality of the generated proofs is often unclear, and such outputs are mainly used for fine-tuning. Other approaches leverage natural-language proof reasoning steps within automated theorem-proving pipelines to improve formal reasoning (Jiang et al., 2023; Tarrach et al., 2024; Liu et al., 2025b). While integrating natural-language reasoning can increase proof success rates, it also presents an opportunity for full-theorem autoformalization, directly translating theorems and proofs from natural language into fully formalized statements. This paper focuses specifically on this scope.

Retrieval-Augmented Autoformalization. Retrieval-Augmented Generation (RAG) (Lewis et al., 2020) has shown improvements in code generation (Lu et al., 2022; Zhang et al., 2023). In formal mathematics, Yang et al. (2023) trained a retrieval-augmented language model for premise selection and theorem proving in Lean. In autoformalization, Zhang et al. (2024) use a BM25 retriever to find similar examples from existing libraries in Isabelle and inject them into prompts. Similarly, Liu et al. (2025a); Wang et al. (2025) retrieve potentially dependent objects from formal libraries to augment statement autoformalization in Lean, and Zhang et al. (2025d) decomposes informal statements into smaller components and retrieves relevant premises from Mathlib. All of these methods rely on content from formal mathematical libraries as references. In contrast, our proposed monotonic process operates in a reference-free manner, requiring no retrieval from existing formal libraries to guide generation.

Test-Time Optimization. Chain-of-Thoughts (Wei et al., 2022) highlights the importance of studying optimization at test time for LLMs. Building on this idea, token-level methods such as Tree-of-Thoughts (Yao et al., 2023) focus on optimizing intermediate tokens to produce higher-quality completions, whereas response-level methods such as Self-Consistency (Wang et al., 2023) aggregate multiple LLM outputs to improve overall performance. In the domain of autoformalization, Li et al. (2024) propose a scoring framework that selects the best candidate from multiple autoformalizations using complementary self-consistency measures. Multi-agent frameworks (Zhang et al., 2025b) similarly organize outputs from multiple LLMs to collectively optimize results. Our framework extends these ideas by employing both theorem provers and LLM-based judges to iteratively select high-quality autoformalization candidates, resulting in a monotonic improvement process at test time.

G Complementary Results

All smaller LLMs used in this work are run on two 24GB NVIDIA Quadro RTX 6000 GPUs, while GPT models are accessed via their API.

G.1 Determination of Generator Roles

For the following experiments, we use greedy decoding for open-source models to improve reproducibility. We sample one instance per generator to reduce computational cost, while empirical evidence suggests that the results remain indicative.

G.1.1 One-Off Generators and FV-Repairers

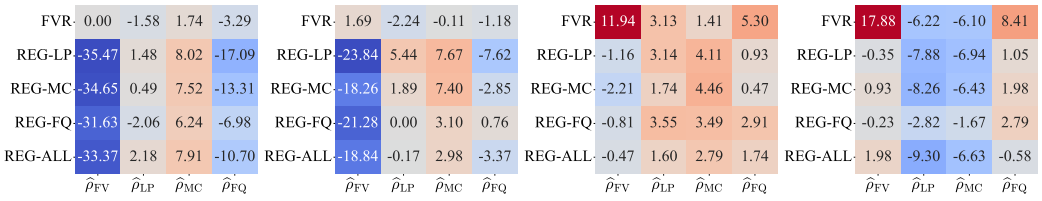
We assess LLMs' capabilities as (i) One-Off Generators under zero-shot autoformalization, and (ii) FV-Repairers under refinement guided by rigorous error feedback from theorem provers. We report scores in Table 2.

Models fine-tuned on the specialized domain perform best in the One-Off Generator setting. Both DeepSeek-Prover-7B and Goedel-Prover-8B achieve higher FV scores than GPT-4.1 under this setting. In particular, Goedel-Prover-8B produces syntactically correct formal theorems and proofs for 58.61% of miniF2F examples and 17.74% of ProofNet examples. Under zero-shot autoformalization, specialized LLMs generate formalizations with higher FV and overall scores, making them well suited to the One-Off Generator setting. However, formalizations produced by these specialized LLMs still receive relatively lower scores on soft semantic dimensions such as LP and FQ, indicating persistent challenges in alignment faithfulness.

Table 2: Performance of LLMs as One-Off Generators (**OOG**) and FV-Repairers (**FVR**) for full theorem autoformalization on miniF2F and ProofNet benchmark using Lean4. FVRs are applied to formalizations generated by OOGs (i.e., zero-shot autoformalization). LLMs include: **Qwen** (Qwen2.5-7B-Instruct), **Seed** (Seed-Coder-8B-Instruct), **DSP** (DeepSeek-Prover-V2-7B), **GDP** (Goedel-Prover-V2-8B), and **GPT** (GPT-4.1).

Type	LLM	π_{FV}	$\widehat{\pi}_{\text{LP}}$	$\widehat{\pi}_{\text{MC}}$	$\widehat{\pi}_{\text{FQ}}$	$\widehat{J}$
miniF2F-Test						
OOG	Qwen2.5-7B-Instruct	0.00	28.89	73.50	15.78	0.04
FVR	Qwen2.5-7B-Instruct	0.00	−0.61	+1.23	+0.41	0.00
FVR	Seed-Coder-8B-Instruct	+0.82	+0.92	−1.78	+2.05	+0.57
FVR	DeepSeek-Prover-V2-7B	+22.95	+18.24	+4.64	+23.36	+15.07
FVR	Goedel-Prover-V2-8B	+30.33	+12.60	−6.01	+28.89	+18.71
FVR	GPT	+12.30	+59.84	+23.91	+53.89	+11.31
OOG	Seed-Coder-8B-Instruct	7.38	28.38	63.39	21.11	5.35
FVR	Seed-Coder-8B-Instruct	+1.64	−1.43	+3.01	+2.05	+1.21
FVR	DeepSeek-Prover-V2-7B	+21.31	+13.73	+10.52	+16.39	+14.45
FVR	Goedel-Prover-V2-8B	+20.49	+8.61	+3.28	+18.65	+11.97
FVR	GPT	+13.11	+54.00	+32.79	+46.93	+11.54
OOG	DeepSeek-Prover-V2-7B	50.41	55.02	80.87	51.02	35.15
FVR	Qwen2.5-7B-Instruct	0.00	−1.74	+1.37	−5.33	0.00
FVR	Seed-Coder-8B-Instruct	+1.64	−4.92	−1.50	−6.97	+1.16
FVR	DeepSeek-Prover-V2-7B	+1.64	+0.92	+0.96	−2.05	+0.90
FVR	Goedel-Prover-V2-8B	+6.56	−3.38	−2.05	+1.02	+4.02
FVR	GPT	+3.69	+18.24	+9.97	+11.48	+3.38
OOG	Goedel-Prover-V2-8B	58.61	47.34	73.63	49.80	35.61
FVR	DeepSeek-Prover-V2-7B	+1.64	0.00	−0.68	−1.02	+1.15
FVR	Goedel-Prover-V2-8B	+7.79	−1.23	−4.51	+1.43	+5.50
FVR	GPT	+2.05	+16.91	+9.15	+10.25	+1.92
OOG	GPT	9.02	90.16	98.09	58.81	8.33
FVR	DeepSeek-Prover-V2-7B	+20.08	−7.99	−3.96	−3.69	+16.36
FVR	Goedel-Prover-V2-8B	+27.87	−20.39	−9.15	−2.05	+21.65
FVR	GPT	+6.56	+2.46	+1.23	+9.22	+6.08
ProofNet-Test						
OOG	Seed-Coder-8B-Instruct	0.00	13.71	51.25	6.18	0.02
OOG	DeepSeek-Prover-V2-7B	11.83	41.80	67.38	35.75	7.14
FVR	DeepSeek-Prover-V2-7B	+0.54	+1.21	+2.15	+3.49	+0.35
FVR	Goedel-Prover-V2-8B	+4.30	−2.96	−0.54	+3.76	+2.88
FVR	GPT	+1.08	+46.77	+27.42	+36.56	+1.11
OOG	Goedel-Prover-V2-8B	17.74	31.59	63.44	29.84	6.89
FVR	DeepSeek-Prover-V2-7B	0.00	+0.13	−1.43	+0.81	0.00
FVR	Goedel-Prover-V2-8B	+2.69	−7.53	−5.38	+4.30	+1.61
FVR	GPT	+1.08	+48.52	+25.63	+38.44	+1.11
OOG	GPT	1.08	91.67	99.46	66.67	1.12
FVR	DeepSeek-Prover-V2-7B	+1.61	−10.22	−5.02	−7.26	+1.51
FVR	Goedel-Prover-V2-8B	+6.99	−31.85	−16.31	−6.99	+4.76
FVR	GPT	0.00	+2.96	−0.18	+8.33	0.00

Specialized LLMs and general-purpose larger LLMs can both serve as effective FV-Repairers for high-validity formalizations. On miniF2F, when acting as FV-Repairers, the general-purpose smaller LLMs Qwen2.5-7B and Seed-Coder-8B show no and limited improvement in FV, respectively. Across both miniF2F and ProofNet, GPT-4.1 achieves larger gains in FV than DeepSeek-Prover-7B when refining formalizations that already have relatively high initial validity, although



(a) Qwen2.5-7B (b) Seed-Coder-8B (c) DeepSeek-Prover-7B (d) Goedel-Prover-8B

Figure 4: Average of local responsiveness (%)for LLMs as different generators.

its improvements still fall short of Goedel-Prover-8B's performance. Goedel-Prover-8B delivers the highest overall gains in FV, but these improvements come at the cost of lower scores on soft-semantic dimensions such as LP and MC. This trade-off indicates that FV alone is insufficient for evaluating overall performance. Taken together, these results highlight the strengths of specialized LLMs and GPT-4.1 as FVRs. Under the FVR setting with Goedel-Prover-8B, we obtain 66.39% syntactically correct formal theorems with proofs on miniF2F and 20.43% on ProofNet.

Specialized LLMs acting as FV-Repairers on low-validity formalizations do not necessarily outperform their own zero-shot autoformalization results. On miniF2F, when DeepSeek-Prover-7B and Goedel-Prover-8B refine zero-shot outputs generated by Qwen2.5-7B, Seed-Coder-8B, or GPT-4.1, the percentage of recovered syntactically correct full theorems is substantially lower than their respective zero-shot performances. GPT-4.1 does achieve more correct full theorems when refining the zero-shot outputs of Qwen2.5-7B and Seed-Coder-8B than in its own zero-shot setting, but the magnitude of its gains remains smaller than those achieved by the specialized LLMs.

G.1.2 Responsiveness Maps of Generators

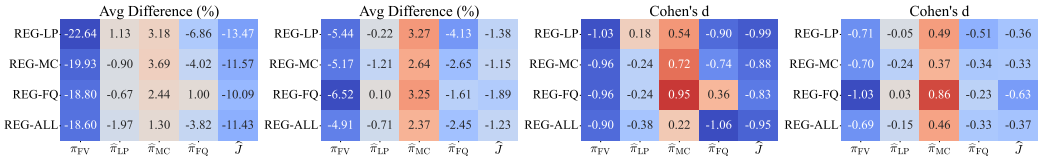
We evaluate the ability of LLMs to act as Recurrent Generators under iterative refinement with feedback from LLM judges. In addition to REG-LP, REG-MC, and REG-FQ, we also consider REG-ALL, which uses feedback aggregated across all evaluation dimensions. We consider Qwen2.5-7B, Seed-Coder-8B, DeepSeek-Prover-7B, and Goedel-Prover-8B as generators, but exclude GPT-4.1 to avoid potential bias from GPT-4.1-mini judgments. Refinement is applied to formalizations with high formal validity produced by the specialized LLMs DeepSeek-Prover-7B and Goedel-Prover-8B. For all generators that use LLM-based feedback (FVR and REG), we compute the responsiveness map for each individual instance and report the mean across all samples for the four LLMs in Figure 4.

Responsiveness maps can illustratively prioritize settings that exhibit the greatest expected gain across the soft dimensions. Among all REG settings, Seed-Coder-8B with REG-LP achieves the highest average expected gain with respect to LP (5.44%), Qwen2.5-7B with REG-MC attains the highest gain for MC (8.02%), and DeepSeek-Prover-7B with REG-FQ yields the largest gain for FQ (2.91%).

Specialized LLMs are more responsive to formal validity, whereas general-purpose LLMs tend to focus on soft semantic properties. Across all settings, Qwen2.5-7B exhibits no positive local responsiveness (Figure 4a). Seed-Coder-8B likewise shows negative responsiveness toward FV in REG settings and only limited responsiveness when configured as FVR (Figure 4b). In contrast, both DeepSeek-Prover-7B and Goedel-Prover-8B demonstrate substantial responsiveness ($>$10%) when configured as FVR (Figures 4c, 4d). These results indicate that specialized LLMs are more responsive to FV. Specialized models also exhibit stronger responsiveness in formal quality compared to general-purpose LLMs. However, for dimensions less directly tied to formal structure (LP and MC), specialized LLMs show limited refinement capability, particularly Goedel-Prover-8B. This contrast suggests that general-purpose and specialized LLMs emphasize different aspects of refinement, reflecting a trade-off between formal and informal dimensions.

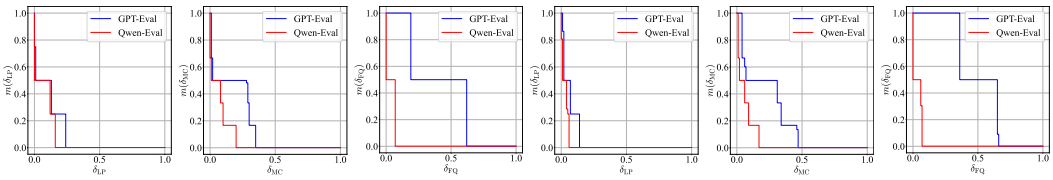
G.2 Dataset-Level Performance of Recurrent Generators

We report the average improvement and Cohen's d on the miniF2F and ProofNet datasets in Figure 5.



(a) miniF2F (b) ProofNet (c) miniF2F (d) ProofNet

Figure 5: Average and Cohen's d of improvements using different feedback on each dataset.



(a) miniF2F LP (b) miniF2F MC (c) miniF2F FQ (d) Proofnet LP (e) ProofNet MC (f) ProofNet FQ

Figure 6: Uncertainty estimation of LLM judges.

Providing feedback on a single dimension yields targeted improvements on that dimension. In Figure 5a, on miniF2F, REG-LP, REG-MC, REG-FQ consistently lead to average improvements in $\widehat{\pi}_{\text{LP}}$, $\widehat{\pi}_{\text{MC}}$, $\widehat{\pi}_{\text{FQ}}$, respectively. In Figure 5b and 5d, on ProofNet, REG-MC likewise exhibits a positive effect on MC judgments. Taken together, these results suggest that providing LLMs with feedback focused on a specific dimension is an effective strategy for improving performance along that dimension.

However, direct feedback on a given dimension is not necessarily the most effective way to improve that dimension. Across both datasets, all REG settings lead to improvements in MC. Notably, the method achieving the largest Cohen's d for $\widehat{\pi}_{\text{MC}}$ is REG-FQ rather than REG-MC (Figure 5c, 5d). This result suggests that refining formalizations using feedback on formal quality can have a statistically larger impact on mathematical consistency than providing MC-specific feedback alone.

Refining one dimension does not guarantee the preservation of other dimensions. All REG settings lead to decreases in formal validity, even when error details are provided alongside LLM judgments. Moreover, in some cases, non-targeted dimensions are negatively affected by giving feedback focused on a specific target dimension. This pattern highlights the difficulty of jointly optimizing multiple dimensions of formalizations and further underscores the necessity of our proposed iterative monotonic process for more reliable autoformalization.

G.3 Uncertainty Estimation

Using the results of the human evaluation in the monotonic process experiments as the ground-truth evaluation $\pi_i(x)$ in Eq. 4, we compute the curves of δ_i and $m(\delta_i)$, which correspond to the maximum admissible margin $m_i(\delta_i)$ for the LLM judges used in our experiments (GPT-4.1-mini and Qwen2.5-Coder-7B). Figure 6 illustrates the behavior of LLM judges under uncertainty.

There is a trade-off between uncertainty level and the effort required to calibrate LLM judges. Across all settings, we observe that the required margin $m(\delta_i)$ increases as the uncertainty level δ_i decreases. This suggests that achieving lower uncertainty requires stronger calibration of LLM judgments. Conversely, using less-calibrated LLM judges leads to higher uncertainty levels.

LLM judges exhibit different levels of uncertainty across different aspects. On both datasets, both LLM judges are more confident in logical preservation (LP) than in mathematical consistency (MC), and least confident in formal quality (FQ). This behavior indicates that MC and FQ are harder to align with human evaluations for LLM judges.

While Qwen2.5-Coder-7B requires less calibration at the same uncertainty level, it is a more conservative judge than GPT-4.1-mini for autoformalization evaluation and is less informative

for selection. The curves corresponding to GPT-4.1-mini consistently lie above those of Qwen2.5-Coder-7B, indicating that larger margins are needed to guarantee the same confidence level. This suggests that GPT-4.1-mini exhibits higher variability and requires stronger correction to obtain reliable positive judgments. However, Qwen2.5-Coder-7B is a conservative evaluator that tends to assign lower scores. Although it has lower uncertainty in the lower-bound sense, it also does not provide sufficient discrimination for selecting better formalizations.

H Details of Prompts

H.1 Prompts for Generators

One-Off Generator

System Prompt:
You are an expert in formal language Lean4.
You will be given a mathematical statement and its proof written in natural language and LaTeX symbols.
Your task is to provide the formal code of the given natural language mathematical statement and its proof in Lean4 with the following instructions:
1. You should give the formal code directly without any additional comments or explanations.
2. In case that you need to import any necessary preambles, you should not import any fake (non-exist) preambles.
3. You should wrap the formal code in a way illustrated as the following:
%%%%%%%%%%
Your Formal Code
%%%%%%%%%%
Strictly follow the instructions that have been claimed.

User Prompt:
Natural language statement: {Natural Language Statement}
Natural language proof: {Natural Language Proof}
Give me the Lean4 formal code of them:

FV-Repairer

System Prompt:
You are an expert in formal language Lean4.
You will be given a mathematical statement and its proof written in natural language and LaTeX symbols.
You will also be given a formal code which attempted to describe the given mathematical statement and its proof in Lean4.
Your task is to refine the given formal code to make it correct while maintaining the alignment with the given natural language mathematical statement and proof.
Here are some instructions for your task:
1. You should give the formal code directly without any additional comments or explanations.
2. In case that you need to import any necessary preambles, you should not import any fake (non-exist) preambles.
3. You should wrap the formal code in a way illustrated as the following:
%%%%%%%%%%
Your Formal Code
%%%%%%%%%%
Strictly follow the instructions that have been claimed.

User Prompt:
Natural language statement: {Natural Language Statement}

Natural language proof: {Natural Language Proof}
There are some Lean4 formal codes describing the given mathematical statement and its proof: {Formal Code}
You should refine the formal code for your task to make it correct.
Here are some feedbacks about the formal code which can be used to help your task: {According to the theorem prover, the error details of the provided formal code are:
Error Details
}

Example of Error Details:
Error on line 5, start column 0, end column 1: unexpected token '#'; expected command
Error on line 138, start column 0, end column None: unexpected end of input; expected '{' or indented tactic sequence

Recurrent Generator

System Prompt:
You are an expert in formal language Lean4.
You will be given a mathematical statement and its proof written in natural language and LaTeX symbols.
You will also be given a formal code which attempted to describe the given mathematical statement and its proof in Lean4.
Your task is to refine the given formal code to make it correct while maintaining the alignment with the given natural language mathematical statement and proof.
Here are some instructions for your task:
1. You should give the formal code directly without any additional comments or explanations.
2. In case that you need to import any necessary preambles, you should not import any fake (non-exist) preambles.
3. You should wrap the formal code in a way illustrated as the following:
%%%%%%%%%%
Your Formal Code
%%%%%%%%%%
Strictly follow the instructions that have been claimed.

User Prompt:
Natural language statement: {Natural Language Statement}
Natural language proof: {Natural Language Proof}
There are some Lean4 formal codes describing the given mathematical statement and its proof: {Formal Code}
You should refine the formal code for your task to make it correct.
Here are some feedbacks about the formal code which can be used to help your task: {Feedback from LLM Judges}

Example of Feedback from LLM Judges:
Aspect: Is the formalized code expressed in a minimal, non-redundant form, avoiding unnecessary repetition or complexity?
Evaluation: Explanation: The formalized code is not expressed in a minimal, non-redundant form. It contains multiple intermediate steps (h4, h5, h6, h7, h8) that are stated but left as "sorry," indicating incomplete proofs or placeholders. While these steps reflect the logical progression of the proof, they are somewhat verbose and could be streamlined by directly manipulating the logarithmic equalities or by combining steps to avoid repetition. For example, the relationships between the logarithms of x, y, and z could be derived more compactly without separately stating each intermediate equality. Additionally, the final step h9 is also a "sorry," so the proof is incomplete and does not demonstrate minimality or efficiency in its

current form. Overall, the code structure suggests redundancy and unnecessary complexity rather than a minimal, concise formalization.
Judgement: False

Aspect: Is the formalized code *internally coherent and contains no contradictions* under the logical rules of the relevant formal system?
Evaluation: Explanation: The formalized code is internally coherent and contains no contradictions under the logical rules of Lean4 and the real number theory it uses. The theorem statement is well-formed, assuming the standard definitions of logarithms (Real.logb) and the domain conditions ($x, y, z > 1, w > 0$). The proof structure is consistent: it introduces variables and hypotheses, then states intermediate lemmas (h4 to h9) that relate the logarithms of w and the bases x, y, z. Although the proof steps are marked as "sorry" (placeholders for omitted proofs), this does not introduce contradictions; it simply means the proof is incomplete but not inconsistent. The chain of equalities and substitutions aligns with the natural language proof logic, and the final conclusion matches the problem statement. Therefore, the code is internally coherent and free of contradictions.
Judgement: True

H.2 Prompts for LLM Judges

We use the Operable Atomic Properties (OAPs) in (Zhang et al., 2025c) to estimate scores for soft-dimensions. OAPs within one soft-dimension are considered equally (i.e., for $i \in \{\mathrm{LP}, \mathrm{MC}, \mathrm{FQ}\}$, $w_{i,k}$ is the same for every k in Eq. 3). The descriptions of OAPs used in the prompts are listed as follows:

For $\widehat{\pi}_{\mathrm{LP}}$ 1. *Pre-arg Structure*: 'Does the formalized code reflect the *inherent predicate-argument structure* of the natural language statement?'; 2. *Quantification*: 'Does the formalized code accurately formalize all *quantifiers*, such as universal and existential, present in the natural language statement?'; 3. *Formula*: 'Are all *mathematical formulas and expressions* in the natural language statement, such as equations and inequalities, correctly and completely represented in the formalized code?'; 4. *Relation*: 'Are the *logical and mathematical relationships between propositions* in the natural language statement preserved in the formalized code?'.

For $\widehat{\pi}_{\mathrm{MC}}$ 1. *Concept*: 'Are all *mathematical concepts* mentioned in the natural language statement, such as integers, fractions, real numbers, complex numbers, derivatives, integrals, vectors, matrices, probabilities, expectations, and variances, are correctly formalized in the formalized code?'; 2. *Constant*: 'Are all *mathematical constants* mentioned in the natural language statement, such as 1, $\frac{2}{3}$, π, e, are properly included in the formalized code?'; 3. *Operator*: 'Are all *mathematical operators* used in the natural language statement, such as addition, subtraction, multiplication, division, summation, exponentiation, and product, are correctly represented in the formalized code?'.

For $\widehat{\pi}_{\mathrm{FQ}}$ 1. *Conciseness*: 'Is the formalized code expressed in a minimal, non-redundant form, avoiding unnecessary repetition or complexity?'; 2. *Logical Consistency*: 'Is the formalized code *internally coherent and contains no contradictions* under the logical rules of the relevant formal system?'.

LLM Judges

System Prompt:
You are an expert in formal language Lean4.
You will be given a mathematical statement and its proof written in natural language and LaTeX symbols.
You will also be given a formal code which attempted to describe the given mathematical statement and its proof in Lean4.
Your task is to evaluate a specific aspect of the formal code.
The description of the aspect is: {OAP Description}

Table 3: Annotation criteria

Logical Preservation (LP) From your point of view, does the formal code capture the logical structure and content of the natural language statement? **1**: Fully captures the logical structure and content. **0.5**: Partially captures the logical structure and content. **0**: Does not capture the logical structure or content.
Mathematical Consistency (MC) From your point of view, does the formal code accurately represent mathematical objects and operations present in the natural language statement? **1**: Accurately represents all mathematical objects and operations. **0.5**: Partially represents the involved mathematical objects and operations. **0**: Does not represent the involved mathematical objects or operations.
Formal Quality (FQ) From your point of view, does the formalized code alone demonstrate high quality in terms of structural clarity, readability, and usefulness? **1**: Demonstrates high structural clarity, readability, and usefulness. **0.5**: Demonstrates moderate structural clarity, readability, or usefulness. **0**: Does not demonstrate adequate structural clarity, readability, or usefulness.

Your need to give two things about your evaluation:
1. the judgement of whether the formalization meets this aspect. This should be a binary value in "True" or "False".
2. the detailed explanation of your judgement.
You should wrap your final results in a way illustrated as the following:
%%%%%%%%%%
Explanation: Your Detailed Explanation
Judgement: Your Binary Judgement
%%%%%%%%%%
Strictly follow the instructions that have been claimed.

User Prompt:
Natural language statement: {Natural Language Statement}
Natural language proof: {Natural Language Proof}
There are some Lean4 formal codes describing the given mathematical statement and its proof: {Formal Code}

I Human Evaluation Criteria

We provide the detailed human evaluation criteria of formalizations in Table 3. Logical Preservation (LP), Mathematical Consistency (MC), and Formal Quality (FQ) are rated on a 3-point scale. Since mathematics is a rigorous discipline and formal languages are designed to reflect this rigor, any human with sufficient knowledge of mathematics and formal language should be able to apply these criteria consistently and reach similar judgments.

J Limitations

One-sided uncertainty modeling of LLM judges. The uncertainty modeling of LLM judges in this work focuses on a one-sided lower confidence bound (LCB), rather than full two-sided uncertainty intervals. This choice is motivated by the design of our monotonic optimization framework, where decision-making is based on conservative acceptance under lower-bound estimates. As a result, our notion of reliability is also defined in terms of lower-bound guarantees. A more comprehensive

treatment of bidirectional uncertainty estimation is an interesting direction but is beyond the scope of this study.

Potential bias from using LLM judges in both optimization and evaluation. LLM-based judges are used both in the test-time optimization loop and in reporting soft-dimension evaluation metrics, which may introduce systematic bias due to shared model preferences. To mitigate this concern, we decouple formal correctness from LLM judgments by using the Lean4 theorem prover as an external verifier for formal validity. In addition, human and cross-model evaluation are conducted to validate the observed trends of the monotonic process. In our framework, LLM judges are explicitly modeled as noisy and imperfect estimators rather than ground-truth oracles. The monotonic acceptance rule based on lower confidence bounds is designed to reduce sensitivity to judge noise and promote conservative updates. While these design choices reduce potential bias, we acknowledge that some residual coupling between optimization and evaluation may remain, particularly in soft metrics.

Dependence on single-annotator human evaluation for uncertainty estimation. The empirical estimation of uncertainty parameters in the LCB formulation relies on human evaluation conducted by a single annotator. While this provides a consistent labeling signal, it may not fully capture inter-annotator variability, and thus may introduce noise in the absolute calibration of uncertainty estimates. Nevertheless, the evaluation is primarily used to characterize relative trends rather than to establish absolute ground-truth values. Future work could explore multi-annotator protocols or probabilistic annotation models to refine uncertainty estimation. Importantly, the proposed monotonic process itself does not depend on perfect calibration of these parameters, and its qualitative behavior remains consistent under reasonable perturbations of the estimates.

Computational cost of ensemble-based monotonic optimization. The proposed monotonic process involves multiple LLM generations and evaluations, and thus incurs higher computational cost compared to single-pass inference methods. This overhead is a direct consequence of its ensemble-style decision mechanism and iterative refinement procedure. However, this cost is offset by improved performance and stability across tasks. In addition, the framework naturally allows performance to scale with increased computation budget (e.g., more iterations or stronger judges), suggesting a form of test-time scaling behavior. Exploring more efficient variants of the monotonic process is a promising direction for future work.

K Additional Results, Tables, and Figures

The explicit results of the monotonic process, the monotonic process with GPT-5.4 only, and the iterative self-refinement process are provided in Table 4, 5, and 6, respectively. The costs of different settings, in terms of the number of LLM calls, are reported in Table 7. We provide a running example of full-theorem autoformalization with our monotonic process for "Dummit-Foote_exercise_1_1_20" from ProoNet in Figure 7. The explicit results of Recurrent Generators (REGs) are reported in Table 8.

Table 4: Performance of the monotonic process. All numbers are reported in percentage.

	miniF2F-Test						ProofNet-Test					
t	π_{FV}	$\widehat{\pi}_{\mathrm{LP}}$	$\widehat{\pi}_{\mathrm{MC}}$	$\widehat{\pi}_{\mathrm{FQ}}$	$\widehat{J}$	$\widehat{J}=1$	π_{FV}	$\widehat{\pi}_{\mathrm{LP}}$	$\widehat{\pi}_{\mathrm{MC}}$	$\widehat{\pi}_{\mathrm{FQ}}$	$\widehat{J}$	$\widehat{J}=1$
GPT-4.1-mini Judges												
0	86.07	57.58	80.60	60.86	57.03	8.61	36.02	48.52	72.04	57.80	18.75	3.23
1	95.90	70.90	89.21	75.82	75.33	27.46	54.84	59.01	80.47	75.54	34.12	9.68
2	99.59	75.41	91.53	82.58	82.76	35.66	62.90	61.83	81.00	79.57	41.37	12.37
3	100.00	79.82	94.54	86.27	86.87	42.21	67.74	61.56	80.47	82.53	45.86	16.13
4	100.00	82.79	96.17	89.34	89.44	46.72	73.12	60.48	79.39	82.53	49.26	17.20
5	100.00	83.71	96.72	90.37	90.27	49.18	77.96	58.06	78.32	82.80	52.45	17.74
Qwen2.5-Coder-7B Judges												
0	-	32.68	46.99	15.98	26.10	0.82	-	26.75	35.13	12.63	7.78	0.54
1	-	36.89	50.41	21.52	34.14	1.64	-	23.79	34.77	16.94	11.23	1.61
2	-	39.34	52.46	22.95	37.96	2.46	-	22.45	33.69	17.20	12.44	2.15
3	-	41.09	54.64	23.57	39.77	2.87	-	19.89	33.51	17.20	12.70	2.15
4	-	40.47	56.01	23.77	40.08	3.28	-	19.49	32.97	16.13	13.78	2.15
5	-	41.39	56.28	24.80	40.82	2.87	-	18.68	32.62	14.52	14.46	2.15
Human Evaluation												
0	-	68.24	68.03	33.61	49.46	1.23	-	62.63	57.53	24.46	15.71	1.61
1	-	73.98	75.82	36.89	59.63	4.92	-	69.62	63.44	22.58	28.25	5.91
2	-	75.41	77.46	38.11	63.32	7.38	-	72.04	65.86	21.78	33.62	6.45
3	-	77.46	79.71	38.32	65.16	8.20	-	70.70	65.32	20.70	35.77	6.99
4	-	78.69	81.35	38.11	66.05	8.20	-	70.97	66.94	21.24	38.63	8.06
5	-	78.89	81.76	39.14	66.60	9.02	-	68.55	68.55	21.77	40.96	8.06

Table 5: Performance of the monotonic process with a single LLM (GPT-5.4).

	miniF2F-Test						ProofNet-Test					
t	π_{FV}	$\widehat{\pi}_{\mathrm{LP}}$	$\widehat{\pi}_{\mathrm{MC}}$	$\widehat{\pi}_{\mathrm{FQ}}$	$\widehat{J}$	$\widehat{J}=1$	π_{FV}	$\widehat{\pi}_{\mathrm{LP}}$	$\widehat{\pi}_{\mathrm{MC}}$	$\widehat{\pi}_{\mathrm{FQ}}$	$\widehat{J}$	$\widehat{J}=1$
GPT-4.1-mini Judges												
0	43.44	81.15	94.26	70.49	37.20	17.62	11.29	86.29	95.16	75.00	10.23	5.91
1	52.87	86.78	96.72	80.74	46.76	26.64	16.13	91.40	97.49	82.26	14.83	9.68
2	56.56	86.78	96.45	84.43	50.27	31.15	17.20	92.88	98.39	86.56	16.10	11.83
3	59.02	88.01	96.58	85.04	52.96	34.84	20.43	93.28	98.21	89.78	19.03	13.98
4	59.43	89.65	97.54	86.48	54.34	37.30	20.97	93.55	98.39	91.67	19.65	15.05
5	60.66	90.16	97.54	87.09	55.64	38.52	21.51	94.49	98.57	92.47	20.19	15.59
Qwen2.5-Coder-7B Judges												
0	-	57.48	62.57	34.22	20.94	4.10	-	56.85	56.99	41.67	5.92	2.69
1	-	59.53	62.70	35.66	24.74	4.92	-	57.53	56.81	41.67	8.14	3.76
2	-	59.73	62.70	37.09	26.88	4.92	-	60.48	58.60	42.74	8.84	3.76
3	-	59.53	63.93	36.89	28.26	4.92	-	60.08	57.89	42.74	9.78	3.76
4	-	59.43	63.80	36.07	28.17	4.92	-	60.75	58.24	44.09	10.07	3.76
5	-	59.22	63.66	35.86	29.01	4.92	-	61.83	58.60	44.35	10.42	3.76

Table 6: Performance of iterative self-refinement (ISR) with error feedback from the theorem prover and no acceptance policy.

	miniF2F-Test						ProofNet-Test					
t	π_{FV}	$\widehat{\pi}_{\mathrm{LP}}$	$\widehat{\pi}_{\mathrm{MC}}$	$\widehat{\pi}_{\mathrm{FQ}}$	$\widehat{J}$	$\widehat{J}=1$	π_{FV}	$\widehat{\pi}_{\mathrm{LP}}$	$\widehat{\pi}_{\mathrm{MC}}$	$\widehat{\pi}_{\mathrm{FQ}}$	$\widehat{J}$	$\widehat{J}=1$
ISR-DeepSeek-Prover-V2-7B												
0	50.41	47.75	80.05	45.08	32.23	6.97	14.52	41.94	69.71	38.17	7.75	1.08
1	47.13	48.16	78.28	43.03	29.47	6.15	8.06	40.59	68.64	32.53	5.46	2.15
2	48.77	49.90	81.42	40.57	30.87	5.74	9.14	39.38	69.18	29.57	6.91	2.15
3	49.59	51.33	80.05	39.34	31.74	5.74	8.06	40.19	69.53	34.95	5.76	2.69
4	48.77	50.20	78.83	37.70	30.56	4.92	7.53	42.20	72.58	30.38	5.48	2.69
5	50.41	52.46	81.01	40.78	33.06	6.15	6.99	42.20	71.68	31.72	5.26	2.15
ISR-Goedel-Prover-V2-8B												
0	65.16	46.62	75.82	47.54	39.12	2.46	22.04	33.47	58.60	36.02	9.93	1.61
1	60.66	40.06	69.67	48.98	36.04	2.05	17.74	25.40	53.05	35.48	8.91	2.69
2	60.66	38.52	68.44	46.31	35.08	1.64	18.28	26.08	54.84	33.33	7.60	2.15
3	59.43	38.32	65.98	44.88	34.37	2.46	21.51	27.55	53.76	36.02	9.20	1.61
4	57.38	37.60	65.44	48.98	32.70	2.05	20.97	25.13	52.51	36.56	8.38	1.08
5	55.74	37.30	65.98	46.72	30.72	1.64	23.66	23.52	52.33	35.22	10.21	1.08
ISR-GPT-5.4												
0	30.74	80.94	94.54	72.54	26.84	14.75	9.68	90.32	97.31	73.92	9.04	3.76
1	44.67	80.43	95.22	78.48	39.32	21.31	10.75	86.69	94.09	75.81	9.99	6.45
2	49.18	81.66	93.99	75.20	42.93	22.13	15.05	86.56	96.06	77.42	14.02	8.60
3	53.28	80.84	94.26	76.64	46.28	22.54	17.74	84.68	96.06	71.51	16.60	11.83
4	52.46	79.82	95.49	74.18	45.89	23.77	18.82	88.71	95.70	77.42	17.54	11.83
5	54.92	81.25	95.08	76.23	48.01	23.36	18.82	86.56	95.70	75.00	17.63	12.37

Table 7: Costs of different settings measured in LLM calls. Mean and standard deviation are computed on a per-sample basis. "Total" denotes the aggregate over all samples in the datasets.

	Generator Calls			Judge Calls		
	Mean	Std	Total	Mean	Std	Total
Best-of-k, $k=8$						
DeepSeek-Prover-V2-7B	8.00	-	3,440	55.95	17.99	24,057
Goedel-Prover-V2-8B	8.00	-	3,440	57.20	17.21	24,597
GPT-5.4	8.00	-	3,440	66.06	12.99	28,404
Iterative Self-Refinement (6 Iterations)						
Per Iteration	1.00	-	430	9.00	-	3,870
Overall	6.00	-	2,580	54.00	-	23,220
The Monotonic Process – Single (GPT-5.4)						
Iteration 0	1.78	0.41	766	9.00	0.00	3,870
Iteration 1	1.63	0.70	701	7.51	3.34	3,231
Iteration 2	1.52	0.80	655	6.70	3.93	2,880
Iteration 3	1.46	0.84	627	6.43	4.07	2,763
Iteration 4	1.41	0.87	608	6.01	4.24	2,583
Iteration 5	1.39	0.89	599	5.90	4.28	2,538
Overall	9.20	4.11	3,956	41.55	17.48	17,865
The Monotonic Process – Ensemble						
Iteration 0	4.42	1.54	1,900	28.88	6.93	12,420
Iteration 1	11.78	3.92	5,067	49.37	32.54	21,231
Iteration 2	10.17	5.47	4,371	34.93	32.86	15,021
Iteration 3	9.56	5.90	4,112	29.74	31.56	12,789
Iteration 4	8.89	6.25	3,823	25.81	30.79	11,097
Iteration 5	8.53	6.38	3,666	22.81	29.14	9,810
Overall	53.35	26.48	22,939	191.55	135.43	82,368

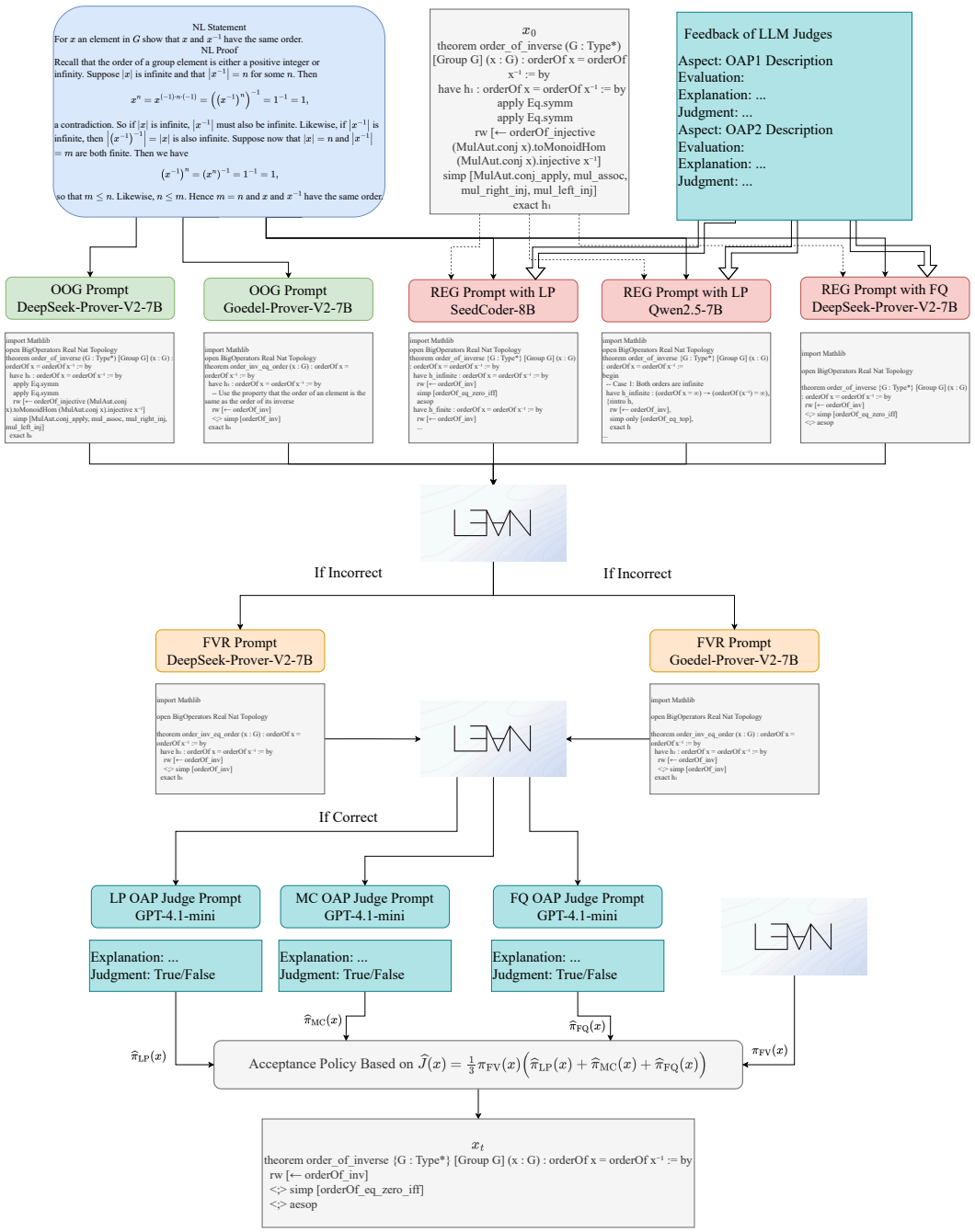


Figure 7: Illustrative example of a single step in the monotonic process. The monotonic process enables the construction of high-quality formalizations that are both formally valid, as verified by the theorem prover, and semantically aligned, as evaluated by LLM judges.

Table 8: Performance of refinement with feedback. All results are based on zero-shot autoformalization.

	Generator Type	LLM	Feedback	π_{FV}	$\widehat{\pi}_{\mathrm{LP}}$	$\widehat{\pi}_{\mathrm{MC}}$	$\widehat{\pi}_{\mathrm{FQ}}$	$\widehat{J}$
miniF2F-Test								
Baseline	One-Off Generator	DeepSeek-Prover-V2-7B	-	50.41	55.02	80.87	51.02	35.15
	Recurrent Generator	Qwen2.5-7B-Instruct	LP	4.10	56.05	86.20	34.84	3.53
	Recurrent Generator	Qwen2.5-7B-Instruct	MC	4.92	55.43	85.52	39.34	4.17
	Recurrent Generator	Qwen2.5-7B-Instruct	FQ	7.79	50.72	84.84	49.59	6.89
	Recurrent Generator	Qwen2.5-7B-Instruct	ALL	4.10	53.89	87.30	43.44	3.72
	Recurrent Generator	Seed-Coder-8B-Instruct	LP	22.54	64.86	89.34	45.90	19.51
	Recurrent Generator	Seed-Coder-8B-Instruct	MC	31.97	57.07	87.70	52.25	26.14
	Recurrent Generator	Seed-Coder-8B-Instruct	FQ	25.41	56.66	82.65	56.15	21.94
	Recurrent Generator	Seed-Coder-8B-Instruct	ALL	28.28	55.53	81.83	49.39	22.42
	Recurrent Generator	DeepSeek-Prover-V2-7B	LP	50.82	59.02	84.29	51.64	36.04
	Recurrent Generator	DeepSeek-Prover-V2-7B	MC	49.18	56.15	84.43	51.43	35.44
	Recurrent Generator	DeepSeek-Prover-V2-7B	FQ	51.23	59.53	84.15	53.07	37.45
	Recurrent Generator	DeepSeek-Prover-V2-7B	ALL	50.82	56.86	83.06	51.43	36.79
	Recurrent Generator	Goedel-Prover-V2-8B	LP	50.00	44.98	73.50	48.57	33.45
	Recurrent Generator	Goedel-Prover-V2-8B	MC	51.23	47.44	75.82	48.77	35.36
	Recurrent Generator	Goedel-Prover-V2-8B	FQ	53.28	52.66	78.96	53.48	37.93
	Recurrent Generator	Goedel-Prover-V2-8B	ALL	52.46	41.91	71.58	45.70	32.50
Baseline	One-Off Generator	Goedel-Prover-V2-8B	-	58.61	47.34	73.63	49.80	35.61
	Recurrent Generator	Qwen2.5-7B-Instruct	LP	0.82	51.23	83.06	28.69	0.82
	Recurrent Generator	Qwen2.5-7B-Instruct	MC	2.05	46.82	83.61	35.25	1.64
	Recurrent Generator	Qwen2.5-7B-Instruct	FQ	10.66	44.88	81.01	45.08	9.10
	Recurrent Generator	Qwen2.5-7B-Instruct	ALL	8.20	51.74	82.79	39.14	7.50
	Recurrent Generator	Seed-Coder-8B-Instruct	LP	17.62	52.77	81.56	39.96	13.62
	Recurrent Generator	Seed-Coder-8B-Instruct	MC	27.05	50.00	82.92	47.34	19.92
	Recurrent Generator	Seed-Coder-8B-Instruct	FQ	25.00	44.98	76.64	52.46	17.88
	Recurrent Generator	Seed-Coder-8B-Instruct	ALL	27.46	47.44	79.10	48.57	19.77
	Recurrent Generator	DeepSeek-Prover-V2-7B	LP	56.56	49.28	77.19	50.61	35.83
	Recurrent Generator	DeepSeek-Prover-V2-7B	MC	55.74	48.77	77.32	47.95	34.57
	Recurrent Generator	DeepSeek-Prover-V2-7B	FQ	57.38	49.08	75.68	50.82	35.54
	Recurrent Generator	DeepSeek-Prover-V2-7B	ALL	56.97	45.90	74.86	49.18	35.04
	Recurrent Generator	Goedel-Prover-V2-8B	LP	52.46	40.27	68.31	48.16	32.17
	Recurrent Generator	Goedel-Prover-V2-8B	MC	54.51	40.57	70.22	48.77	32.91
	Recurrent Generator	Goedel-Prover-V2-8B	FQ	54.92	45.59	73.63	50.61	35.30
	Recurrent Generator	Goedel-Prover-V2-8B	ALL	59.02	40.37	67.90	45.90	33.53
ProofNet-Test								
Baseline	One-Off Generator	DeepSeek-Prover-V2-7B	-	11.83	41.80	67.38	35.75	7.14
	Recurrent Generator	Qwen2.5-7B-Instruct	LP	0.54	40.46	75.99	22.04	0.31
	Recurrent Generator	Qwen2.5-7B-Instruct	MC	2.15	42.07	74.55	22.85	1.82
	Recurrent Generator	Qwen2.5-7B-Instruct	FQ	0.54	40.86	74.73	23.12	0.54
	Recurrent Generator	Qwen2.5-7B-Instruct	ALL	0.54	43.55	76.34	22.58	0.54
	Recurrent Generator	Seed-Coder-8B-Instruct	LP	6.45	45.83	76.16	31.45	4.69
	Recurrent Generator	Seed-Coder-8B-Instruct	MC	6.45	42.34	73.48	31.45	5.29
	Recurrent Generator	Seed-Coder-8B-Instruct	FQ	4.84	42.20	72.22	33.60	4.26
	Recurrent Generator	Seed-Coder-8B-Instruct	ALL	7.53	40.86	72.22	29.30	5.29
	Recurrent Generator	DeepSeek-Prover-V2-7B	LP	11.29	44.62	73.12	35.22	6.66
	Recurrent Generator	DeepSeek-Prover-V2-7B	MC	10.22	43.95	73.48	39.52	6.54
	Recurrent Generator	DeepSeek-Prover-V2-7B	FQ	11.83	46.24	72.76	40.86	7.50
	Recurrent Generator	DeepSeek-Prover-V2-7B	ALL	12.90	45.30	72.94	40.05	8.23
	Recurrent Generator	Goedel-Prover-V2-8B	LP	20.43	37.50	62.01	41.67	12.40
	Recurrent Generator	Goedel-Prover-V2-8B	MC	18.82	32.53	60.93	44.35	11.81
	Recurrent Generator	Goedel-Prover-V2-8B	FQ	11.83	40.46	67.03	39.52	8.12
	Recurrent Generator	Goedel-Prover-V2-8B	ALL	16.67	32.53	63.08	38.17	10.26
Baseline	One-Off Generator	Goedel-Prover-V2-8B	-	17.74	31.59	63.44	29.84	6.89
	Recurrent Generator	Qwen2.5-7B-Instruct	LP	1.61	33.33	72.58	13.44	1.30
	Recurrent Generator	Qwen2.5-7B-Instruct	MC	1.08	33.74	71.86	15.59	1.08
	Recurrent Generator	Qwen2.5-7B-Instruct	FQ	1.61	31.85	70.07	18.28	1.57
	Recurrent Generator	Qwen2.5-7B-Instruct	ALL	1.61	35.62	70.61	17.47	1.52
	Recurrent Generator	Seed-Coder-8B-Instruct	LP	3.23	32.66	68.64	18.55	2.76
	Recurrent Generator	Seed-Coder-8B-Instruct	MC	4.30	33.60	70.43	22.58	3.29
	Recurrent Generator	Seed-Coder-8B-Instruct	FQ	3.23	32.12	66.67	25.27	2.66
	Recurrent Generator	Seed-Coder-8B-Instruct	ALL	4.84	30.91	63.98	24.46	3.54
	Recurrent Generator	DeepSeek-Prover-V2-7B	LP	15.05	35.48	67.56	32.80	7.97
	Recurrent Generator	DeepSeek-Prover-V2-7B	MC	14.52	34.14	68.46	30.11	6.82
	Recurrent Generator	DeepSeek-Prover-V2-7B	FQ	14.52	35.35	67.20	34.14	7.26
	Recurrent Generator	DeepSeek-Prover-V2-7B	ALL	16.13	34.95	66.31	33.87	7.57
	Recurrent Generator	Goedel-Prover-V2-8B	LP	16.13	21.91	53.41	34.14	8.65
	Recurrent Generator	Goedel-Prover-V2-8B	MC	19.35	21.51	51.25	34.68	9.98
	Recurrent Generator	Goedel-Prover-V2-8B	FQ	17.74	25.27	58.60	34.68	8.74
	Recurrent Generator	Goedel-Prover-V2-8B	ALL	18.82	24.19	56.81	36.83	8.96