Consistent Autoformalization for Constructing Mathematical Libraries

Anonymous ACL submission

Abstract

Autoformalization is the task of automatically translating mathematical content written in natural language to a formal language expression. The growing language interpretation capabilities of Large Language Models (LLMs), including in formal languages, are lowering the barriers for autoformalization. However, LLMs alone are not capable of consistently and reliably delivering autoformalization, in particular as the complexity and specialization of the target domain grows. As the field evolves into the direction of systematically applying autoformalization towards large mathematical libraries, the need to improve syntactic, terminological and semantic control increases. This paper proposes the coordinated use of three mechanisms, most-similar retrieval augmented generation (MS-RAG), denoising steps and auto-correction with syntax error feedback (Auto-SEF) to improve autoformalization quality. The empirical analysis, across different models, demonstrates that these mechanisms can deliver autoformalizaton results which are syntactically, terminologically and semantically more consistent. These mechanisms can be applied across different LLMs and have shown to deliver improve results across different model types.[1]

1 Introduction

Mathematical reasoning constitutes an essential aspect of human intelligence (Saxton et al., 2019; Lu et al., 2023). It centers on symbolic-level reasoning, as manifested through systematic, abstract and and step-wise logical inference. Mathematical reasoning has been clustered under two types of models: deep learning models (Hendrycks et al., 2021; Wei et al., 2022; Meadows and Freitas, 2023; Liu et al., 2023) and formal models (Polu and Sutskever, 2020; Wang and Deng, 2020; Han et al., 2022; Jiang et al., 2022, 2023b). Mathematical reasoning in Large Language Models (LLMs) predominantly utilizes statements expressed in informal mathematical statements. More recent models have aimed towards bridging both informal and formal mathematical reasoning (Wu et al., 2022; First et al., 2023b; Azerbayev et al., 2023; Quan et al., 2024a), where the material (content-based) inference strengths of LLMs are complemented by external formal/symbolic reasoning methods such as automated theorem provers (e.g. Isabelle (Paulson, 2000) and Lean (de Moura et al., 2015)), which can systematically assess the logical validity of the reasoning process (Wu et al., 2022), facilitating LLMs to perform controlled and consistent inference.

However, formal and verifiable mathematical reasoning with theorem provers requires the manual formalization of logical formulae from informal statements, in order to build the supporting mathematical libraries, knowledge bases (KBs) which express previous axioms, definitions, theorems and proofs, a process that demands considerable effort and domain-specific knowledge. A prototypical case in point is the liquid tensor experiment (Scholze, 2022), an initiative aimed at formalizing analytical geometry results from Scholze & Clausen, requiring a community coordinated effort of experts.

Contemporary LLMs have demonstrated considerable efficacy (Wu et al., 2022; Xin et al., 2023; First et al., 2023b) for supporting autoformalization efforts within an in-context learning paradigm, being largely evaluated in less specialized domains and tasks. Existing methods are still limited in delivering a method for systematically and consistently building large formal and specialized mathematical libraries. The essence of the challenge is twofold: (i) *specialization and out-of-distribution (OOD) drifts:* as one moves towards more specialized and newer domains to be autoformalized, models are progressively exposed to more chal-

[1] Code and datasets are available at anonimyzed_link

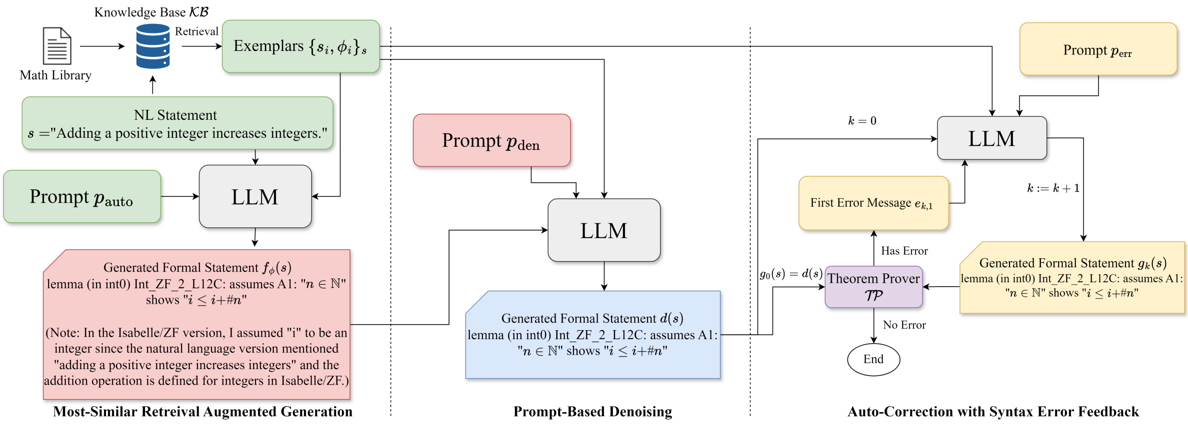

Figure 1: The overall framework consists of three stages: Stage 1 contains one round for retrieval augmented autoformalization; Stage 2 contains one round for denoising; Stage 3 is composed of several iterative rounds to refine the code based on syntax errors. For better illustration, we change \<in>, \<nat>, \<lsq>, $+$ to their LaTeX version $\in$, $\mathbb{N}$, $\leq$, $+$. The ground truth code is *lemma (in int0) Int_ZF_1_5_L7A: assumes "a\<in>\<int>" "b \<in>\<int>\<^sub>+"shows "a \<lsq>a\<ra>b" "a \<noteq>a\<ra>b" "a\<ra>b \<in>\<int>"* (assumes "$a \in \mathbb{Z}$" "$b \in \mathbb{Z}^+$" shows "$a \leq a + b$" "$a \neq a + b$" "$a + b \in \mathbb{Z}$").

lenging OOD cases, and (ii) *library consistency and coherence:* new formalized need to be consistently built-up on previously statements, cohering terminologically, syntactically and semantically.

This work targets this overarching research question, namely: 'how to systematically support the creation of consistent and coherent formal mathematical libraries from informal mathematical statements?'. In order to address this task, we decompose this broader aim into the following research questions: *RQ1: 'To what extent contemporary LLMs are capable of formalizing specialized mathematical statements into formal representations for mathematical libraries?'; RQ2: 'Which metrics can be used to assess the quality of this formalization?'; RQ3: 'Which mechanisms can be used to extend the autoformalization properties of LLMs to achieve better generative control and enhance terminological, syntactic and semantic consistency and coherence?'*. To address these research questions, we propose a novel framework (See Figure 1) that leverages LLMs with most-similar retrieval augmented generation (MS-RAG), denoising steps and iterative feedback-guided syntax error refinement cycles (Auto-SEF) to deliver a syntactically consistent and semantically coherent autoformalization.

To assess the effectiveness of our proposed framework, we construct a supporting dataset for the task of mathematical library autoformalization (MathLibForm) and build a supporting empirical analysis methodology guided by a critical selection of a set of automated metrics. We conduct a systematic empirical analysis with a diverse sample of state-of-the-art LLMs, in order to compare and contrast their autoformalization properties and the impact of the proposed library autoformalization mechanisms. Our results demonstrate that leveraging LLMs with MS-RAG and Auto-SEF, combined with denoising strategies, can significantly enhance the syntactic correctness of formalization results, reaching improvements from 5.47% to 33.58%. In summary, the contributions of the paper are:

1. Proposal of a novel neuro-symbolic framework targeting the autoformalization of mathematical libraries, which employs LLMs with MS-RAG, denoising and Auto-SEF to consistently and iteratively enhance and refine the formalization results;

2. Definition of a new task (formalization of mathematical libraries) and creation of a supporting dataset (MathLibForm);

3. Proposal of an evaluation methodology.

2 Proposed Approach

In this section, we start by defining the target task and then describe the proposed mechanisms for improving autoformalization.

Autoformalization: An autoformalization is a transformation function which maps an informal mathematical statement s in the domain of natural language and LaTeX symbols $\mathcal{S}$ into a formal mathematical statement ϕ, under a formal language $\mathcal{F}$, $f : \mathcal{S} \rightarrow \mathcal{F}$, such that for every $s \in \mathcal{S}$, there exists a $\phi \in \mathcal{F}$ where $f(s) = \phi$.

Semantic correctness: A transformation $f(s) = \phi$ is semantically correct if there exists a model $\mathcal{M}$ such that:

$$\exists \mathcal{M} \quad : \quad \mathcal{M} \models s \quad \text{and} \quad \mathcal{M} \models \phi,$$

where $\models$ denotes that the former item satisfies or correctly interprets the latter.

Library-based autoformalization: Given a Knowledge Base ($\mathcal{KB}$) of formalised mathematical statements under a formal language $\mathcal{F}$, a *library-based autoformalization transformation function* f_Φ is defined such that the generated statement ϕ is semantically consistent with the set of statements $\Phi \in \mathcal{KB}$.

Semantic consistency: A statement ϕ is semantically consistent with respect to $\mathcal{KB}$ if all terms in ϕ that have references in $\mathcal{KB}$ are used consistently with the terms in $\mathcal{KB}$. Formally, let ϕ be a statement and $\mathcal{KB}$ be a knowledge base. ϕ is semantically consistent with respect to $\mathcal{KB}$ if:

$$\forall t \in \text{terms}(\phi) \cap \text{references}(\mathcal{KB}), \quad t_\phi = t_{\mathcal{KB}},$$

where $\text{terms}(\phi)$ denotes the set of terms in ϕ and $\text{references}(\mathcal{KB})$ denotes the set of referenced terms in $\mathcal{KB}$.

2.1 Most-Similar Retrieval Augmented Generation (MS-RAG)

Under the aforementioned formal notations, autoformalization with LLMs defines the transformation function as:

$$f(s) = \text{LLM}(p_{\text{auto}}, \{(s_i, \phi_i)\}, s),$$

where p_{auto} is a prompt for autoformalization and $\{(s_i, \phi_i)\}$ is a set of exemplars. The initial attempt (Wu et al., 2022) defined subcategories $\mathcal{SC}_j$ in math and chose fixed examples $\{(s_i, \phi_i)\}_j \in \mathcal{SC}_j$ for each subcategory, where the transformation function becomes:

$$f(s) = \text{LLM}(p_{\text{auto}}, \{(s_i, \phi_i)\}_j, s), \text{if } s \in \mathcal{SC}_j.$$

However, fixed examples cannot reflect the usage of various novel definitions and notions in each subcategory. Therefore, with the assumption of the existence of $\mathcal{KB}$, we propose to first retrieve a set of samples based on a similarity relevance function $\mathcal{MS}(s) \in \mathcal{KB}$ and then define the transformation function as:

$$f_\phi(s) = \text{LLM}(p_{\text{auto}}, \{(s_i, \phi_i)\}_s, s),$$

where $(s_i, \phi_i) \in \mathcal{MS}(s)$.

2.2 Denoising Formalization Results

Bias inherited from instruction fine-tuning (Ouyang et al., 2022) causes LLMs during autoformalization to occasionally generate redundant texts not integral to the formal statement, thereby infusing the final output with noisy information. Consequently, the direct output of LLMs frequently fails to meet the criteria for a valid formal code. Please note that despite the fact that output conditions can be communicated on the initial prompt, typically the output behaviour of the models can be less controlled and nor fully enforceable. To alleviate this issue, we propose two types of denoising:

Code-Based Denoising (CBD). Definition of a set of post-processing rules $\mathcal{R}$ to remove irrelevant outputs such as *extra explanations* and *unsolicited proofs* , where a new formal statement is obtained: $d(s) = \mathcal{R}(f_\phi(s))$.

Prompt-Based Denoising (PBD). The rigidity of a CBD method can be contrasted to a post-hoc prompt-based approach for the same purpose. Hence, we propose to design a prompt p_{den} for LLMs to do the denoising of the autoformalization results. Denoising with only a prompt raises the risk of losing semantic consistency because of the bias in the training data of LLMs. Therefore, the set of retrieved items $\mathcal{MS}(s)$ from MS-RAG could be used to maintain semantic consistency. The denoising becomes: $d(s) = \text{LLM}(p_{\text{den}}, \{(s_i, \phi_i)\}_s, f_\phi(s))$.

Using reported syntax errors as a feedback have been established as a systematic mechanism for guiding the correction of formal models (Quan et al., 2024a,b) for LLMs potentially automatically correct the formalization results.

2.3 Auto-correction with Syntax Error Feedback (Auto-SEF)

The validity of any formal code ϕ can be checked by a theorem prover $\mathcal{TP}$ that supports its written formal language $\mathcal{F}$. If the formal code is not valid, the theorem prover can output a set of syntax errors $\{e_k\} = \mathcal{TP}(\phi)$. Using reported syntax errors as feedback has been established as a systematic mechanism for guiding the correction of formal models (Quan et al., 2024a,b), potentially allowing LLMs to automatically correcting the results of formalization. Hence, we design a prompt p_{err} to add an auto-correction component to let LLMs recognize previously produced errors and correct mistakes. To maintain semantic consistency, retrieved examples are also used and the generation becomes:

$$g(s) = \text{LLM}(p_{\text{err}}, \{(s_i, \phi_i)\}_s, \{e_k\}, d(s)).$$

where $\{e_k\} = \mathcal{TP}(d(s))$. Within this setting we propose an iterative process:

$$g_{k+1}(s) = \text{LLM}(p_{\text{err}}, \{(s_i, \phi_i)\}_s, e_{k,1}, g_k(s))$$

with initial state $g_0(s) = d(s)$ and $e_{k,1}$ is the first item in $\mathcal{TP}(g_k(s))$.

3 Evaluation Benchmark

3.1 MathLibForm

Formal mathematical datasets, such as miniF2F (Zheng et al., 2022), predominantly concentrate on distinct mathematical problems representing simpler mathematical solving tasks. In contrast, the creation of mathematical libraries demands the autoformalization fo statements which can be more specialized, conceptually more complex and potentially out-of-distribution. In this work we use *IsarMathLib*[2], as a reference setting within the environment of the Isabelle/ZF theorem prover framework. Formal statements in IsarMathLib are frequently accompanied by textual comments, which can serve as the natural language statements of the formal expressions. Mathematical items: *lemma*, *definition*, *corollary*, *theorem*, along with textual comments and proofs, were extracted with a script first. This leads to a total of 2,744 items, which were then randomly split into training and test sets in a 90% to 10% proportion, resulting in 2,470 training samples and 274 test samples for constructing the MathLibForm dataset. To enrich the information contained in MathLibForm, we also informalize formal statements with Mistral and add the generated textual descriptions. The training and testing sets are utilized to build the knowledge base $\mathcal{KB}$ and to evaluate methods, respectively.

3.2 Evaluation Metrics

The correctness of generated formal statements serves as the most crucial and direct metric for evaluating the performance of autoformalization. However, assessing correctness requires human evaluation, which is a time-consuming process and cannot be seamlessly integrated into an autonomous evaluation system. In this work, we proposed two distinct components to access code correctness: *semantic similarity* and *syntactic correctness*. Utilizing the ground truth as a reference, we measure semantic similarity using pairwise metrics, including BLEU (Papineni et al., 2002), ChrF (Popović, 2015), RUBY (Tran et al., 2019), and CodeBERTScore (CBS) (Zhou et al., 2023). The implementation details of these metrics are provided in *Appendix*. To assess syntactic correctness, we use Isabelle theorem prover to detect syntax errors in formal statements and use the *Pass* metric which represents the success rate at which the generated formal statement does not exhibit any syntax errors, as verified by the theorem prover. The integration between the transofrmer and Isabelle is done on a ToolFormer setting with the support of an Isabelle client[3] (Shminke, 2022).

4 Experiments and Analysis

4.1 Retrieval Augmented Autoformalization

We establish baselines in zero-shot and 3-shot settings on several state-of-the-art LLMs: Mistral (Jiang et al., 2023a), Llemma 7B (Azerbayev et al., 2024), Mixtral (Jiang et al., 2024a), GPT-3.5-Turbo (descriptions of the models can be found in *Appendix*). For MS-RAG, BM25 (Robertson et al., 1994) is used as the primary ranking function to retrieve Top-k (k=3) most similar samples for exemplars (BM25 will concentrate a terminological similarity function). Different settings are contrasted for querying and indexing the reference KB. There are two choices for query: 1. natural language textual description; 2. description along with zero-shot autoformalization result from Mistral. The choices for indexing KB elements combine three content sources: 1. natural language textual description; 2. informalization of formal statements; 3. formal statements. For this specific analysis, we constraint the foundation model to Mistral. All results are reported in Table 1.

MS-RAG can improve autoformalization in mathematical libraries settings. As shown in Table 1, for the same type of LLMs, using retrieved examples rather than fixed examples leads to an

[2] https://github.com/SKolodynski/IsarMathLib

[3] https://github.com/inpefess/isabelle-client

LLM	Method	BLEU-2	ChrF	RUBY	CBS	Pass
Baselines						
Mistral	Zero-Shot	0.30	17.14	16.13	51.13	0.0
Mistral	3-Shot	1.77	27.30	24.02	62.73	5.47
Llemma 7B	Zero-Shot	0.91	16.67	14.77	47.74	9.12
Llemma 7B	3-Shot	2.43	28.81	21.93	66.68	8.76
Mixtral	Zero-Shot	0.65	16.33	17.97	51.07	0.36
Mixtral	3-Shot	5.37	30.53	28.51	62.86	1.09
GPT-3.5-Turbo	Zero-Shot	2.15	17.81	21.93	51.69	40.51
GPT-3.5-Turbo	3-Shot	14.23	37.95	39.13	67.26	38.69
Retrieval Augmented Autoformalization						
Mistral	Query: **T** Index: **T**	10.05	51.38	44.82	76.93	21.53
Mistral	Query: **T** Index: **T+S**	9.96	50.79	43.92	76.21	19.71
Mistral	Query: **T** Index: **I+S**	5.65	36.92	32.23	67.47	8.76
Mistral	Query: **T** Index: **T+I+S**	10.53	49.61	43.28	75.17	**2**2.26
Mistral	Query: **T+ZS** Index: **T**	10.14	46.89	40.76	73.69	12.77
Mistral	Query: **T+ZS** Index: **T+S**	8.40	46.26	39.91	73.40	14.96
Mistral	Query: **T+ZS** Index: **I+S**	5.51	36.71	31.94	66.91	10.95
Mistral	Query: **T+ZS** Index: **T+I+S**	8.85	45.14	39.27	72.47	16.06
Llemma 7B	Query: **T** Index: **T**	4.18	36.93	28.68	69.93	12.77
Llemma 7B	Query: **T** Index: **T+S**	4.61	37.48	29.39	69.56	14.23
GPT-3.5-Turbo	Query: **T** Index: **T**	36.32	**59.63**	**58.51**	**79.14**	**64.60**
GPT-3.5-Turbo	Query: **T** Index: **T+S**	**37.11**	58.56	57.71	78.89	62.77

Table 1: Autoformalization results for different settings. BM25 retriever is used to retrieve Top-3 most similar samples for retrieval augmented autoformalization. Greedy decoding is used in generation for reproducibility. Code-based denoising is applied to all outputs. The query used to retrieve relevant exemplars includes: (**T**): natural language textual description; (**ZS**): zero-shot autoformalization result from Mistral. The index used for knowledge base has the following options: (**T**): natural language textual description; (**I**): informalization of formal statement generated from Mistral; (**S**): formal statement. The setting with highest scores is highlighted in **bold**.

improvement in both semantic similarity and syntactic correctness of the generated formal statements. This mechanism can lift the performance of smaller models: e.g. as a smaller model, Mistral (7B) with MS-RAG can outperform Mixtral (8×7B) with standard prompting across all metrics and is comparable to GPT-3.5 (175B) without MS-RAG according to some metrics such as RUBY.

Similarity-based few-shot outperforms zero-shot learning. For all LLMs, autoformalization results with 3-shot exemplars are generally better than those from the zero-shot setting in terms of semantic similarity metrics. For syntactic correctness, Llemma 7B and GPT-3.5 in the zero-shot setting have slightly higher pass rates compared to the 3-shot setting.

MS-RAG levels the playing field across models of different scales. As the largest LLM in our experiments, GPT-3.5 with MS-RAG significantly outperforms all other models. However, comparing its best performance with MS-RAG to its performance in the 3-shot setting, its relative change in syntactic correctness (67%) is much lower than that with Mistral (307%). The relative change for Llemma 7B is the smallest (62%). We attribute this to the fact that Llemma was not finetuned with instructions. These differences suggest that smaller LLM with instruction tuning benefits more from RAG.

Augmenting the index with auto-informalization or the query with zero-shot auto-formalization does not lead to better retrieval. Among all results in Table 1, GPT-3.5 with textual description query and textual description index achieves highest scores in four metrics except BLEU-2. Compared to other choices with Mistral, this choice also leads to highest scores in ChrF, RUBY and CBS and second highest scores in Pass, which suggests that it is the best choice. Incorporating zero-shot results from Mistral as queries generally yields worse

Metric	MS-RAG	PBD 1A	PBD 1B	PBD 1C	PBD 1D
BLEU-2	6.33 (+3.72)	8.88 (+1.61)	11.30 (+1.99)	15.21 (+1.49)	14.90 (**+2.42**)
ChrF	48.45 (**+2.93**)	38.27 (-0.35)	43.25 (-0.06)	44.52 (-0.23)	48.51 (+0.11)
RUBY	28.99 (+15.83)	38.23 (+2.12)	42.08 (+1.91)	44.59 (+0.79)	46.43 (**+0.98**)
CBS	76.40 (**+0.53**)	68.04 (-0.03)	70.51 (-0.07)	71.92 (+0.01)	74.07 (+0.03)
Pass	17.15 (+4.38)	6.57 (+0.00)	9.12 (+0.00)	13.50 (+0.37)	28.10 (**+0.00**)

Table 2: The effect of denoising on Mistral. The change of scores after applying CBD is recorded in round brackets. The setting with highest final scores is marked as **bold**.

NL Statement	The next lemma shows that open sets are members of (what we will prove later to be) the natural neighborhood system on $X = \bigcup T$.
Ground Truth	lemma open_are_neighs: assumes "U\<in>T" "x\<in>U" shows "x \<in> \<Union>T" and "U \<in> {V\<in>Pow(\<Union>T) .\<exists>U\<in>T. (x\<in>U \<and> U\<subseteq>V)}"
3-shot	lemma open_set_in_nhs: assumes "X = \<bigcup> T" "T \<in> Covers(X)" "U \<in> Open(X)"shows "U \<in> NHS(X)" where NHS(X) is a predicate representing the natural neighborhood system on X. Note: The definition of NHS(X) should be provided before using this lemma.
MS-RAG	lemma open_is_neighbors: assumes "U \<in> T" and "x \<in> U" shows "U \<in> ({neighborhood system of} T)`(x)" (* Note: This lemma assumes that T is a topology, but it is not explicitly stated in the natural language version.*)
PBD 1A	lemma open_is_neighbors: assumes "U :: set T" and "x :: T" shows "U :: ({neighborhood system of} T) x"
PBD 1B	lemma open_is_neighbors: assumes "U \<in> T" and "x \<in> U" shows "U \<in> ({neighborhood_system} T)`(x)"
PBD 1C	lemma open_is_neighbors: assumes "U \<in> T" "x \<in> U" shows "U \<in> ({neighborhood system of} T) x"
PBD 1D	lemma open_is_neighbors: assumes "U \<in> T" and "x \<in> U" shows "U \<in> ({neighborhood system of} T)`(x)"

Table 3: An example using Mistral shows that only MS-RAG and PBD 1D have no syntax errors of formalization.

results compared to its counterpart. This is probably caused by the low quality of zero-shot formalization results. Choosing informalized description for index also does not have a better performance.

4.2 Output Denoising

In this section, we investigate the impact of denoising. We select the result of MS-RAG (Query: T, Index: T) to apply PBD with four prompts: (**1A**) The prompt only contains instructions to remove explanations and proofs; (**1B**) 1A adds an additional instruction for *stylistic alignment* to declare that the final output after refinement should maintain the same syntactic style; (**1C**) Includes some fixed formal statement examples for the stylistic alignment instruction in 1B; (**1D**) Changes the fixed examples in 1C to retrieved examples from MS-RAG. We record the results of Mistral in Table 2.

Denoising significantly impacts the quality of the formal statements. Compared to results without denoising, using either denoising method can significantly improve BLEU and RUBY scores. Applying CBD to the original MS-RAG results can lead to an improvement across metrics. However, the effect of CBD decreases after we apply PBD to the results. For the Pass metric, performing CBD after PBD had no observable impact. This demonstrates the impact of PBD as a syntactic control mechanism. Our results suggest that a composition of PBD and CBD can yield the best performance in syntactic correctness while maintaining semantic similarity at the same or higher level.

Denoising can reduce the performance gap between smaller LLM and larger LLM. We also conducted similar experiments on GPT-3.5 (results in *Appendix*). Denoising methods have a comparatively lower effect on the results of GPT-3.5, serving more as a function of control for smaller models, approaching their performance to larger models.

Stylistic alignment is necessary when applying PBD. Without the explicit declaration of stylistic alignment (1A), the syntactic correctness drops 10.58% compared to the results of MS-RAG. The reason is that when we only ask Mistral to remove redundant strings, it tends to neglect the original syntactic style of the formal statements and rewrites them in a new style that it was trained on. However, merely specifying that the model should maintain such a style without giving explicit examples (1B) does not effectively communicate the intent to preserve the style. This is demonstrated by the higher performance of 1C compared to 1B. In addition, using retrieved examples (1D) rather than fixed examples (1C) can increase scores further.

Case Study We use an example in Table 3 to conduct a case study on the necessity of denoising. As shown in Table 3, both 3-shot and MS-RAG results include an additional textual description in the final output which does not form a formal statement. PBD 1A changes "\<in>" into "::" which is another way of expressing "∈" but this expression is not provided in its prompt, so this behaviour is highly likely to be the bias of Mistral. PBD 1B and 1C mitigate this behaviour but they also make other syntax errors, such as the missing word "of" or the special character "`". **Only PBD 1D maintains the validity of the formal statement because the retrieved examples have a similar usage of these elements and hence they are emphasized during generation.**

4.3 Iterative Symbolic Refinement

In this section, we mainly focus on answering the question on whether syntax errors can be corrected by LLMs in coordination with symbolic solvers. This process is iteratively run for up to nine cycles. To better illustrate the changes, we plot the scores of each iteration on the Pass metric in Figure 2.

Iterative Auto-SEF improves syntactic correctness of the formalization results. As shown in Figure 2, both GPT-3.5 and Mistral can receive improvements from the iterative Auto-SEF method. This result demonstrates that Auto-SEF can indeed enable LLMs to fix some syntactic errors. The first iteration brings the largest increase (2.56% for Mistral, 4.38% for GPT-3.5) in pass rate. After that, the change becomes smoother and iterative improvements are limited to a small number of cycles.

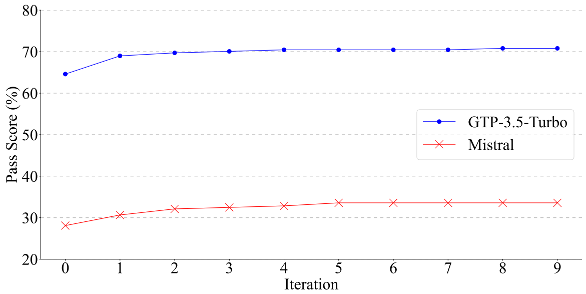

Figure 2: Pass rate of each iteration with Auto-SEF. Iteration 0 is the start point before applying Auto-SEF.

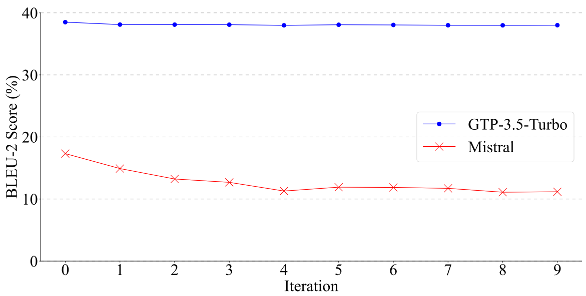

Figure 3: BLEU-2 scores of each Auto-SEF iteration.

Smaller LLM tends to trade-off semantic similarity for syntactic correctness when applying Auto-SEF. We select BLEU-2 as a proxy for semantic similarity and illustrate the scores of each iteration in Figure 3. The BLEU-2 scores for GPT-3.5 remain steady across different iterations, whereas for Mistral, the scores decrease in the first few iterations. Combining this result with the improvement in pass rate, we hypothesize that a trade-off occurs due to the comparatively lower capacity of Mistral to perform syntactic correction while controlling for semantic drifting during Auto-SEF prompting.

4.4 A Critique of the Metrics

Metrics for the evaluation of code generation can disagree with each other (Evtikhiev et al., 2023). We use all results with CBD to calculate the Pearson product-moment correlation coefficients between metrics, illustrating these coefficients with a heatmap in Figure 4.

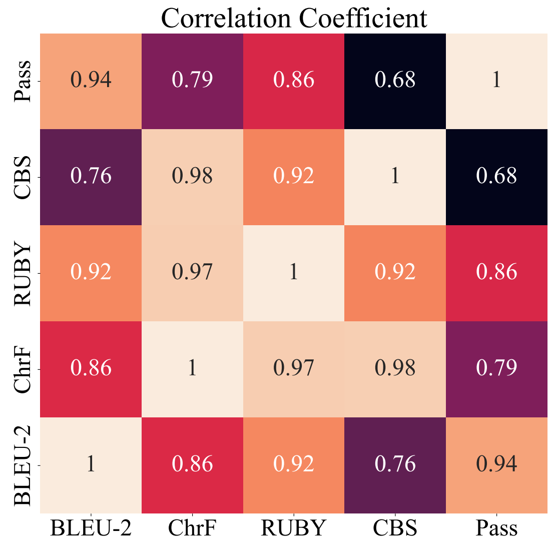

Figure 4: Correlation coefficients between metrics.

RUBY can serve as an initial metric when evaluating formalization results. All correlation coefficients are larger than 0.6. This suggests that all metrics are positively related to each other and that any one of them is a reasonable indicator for evaluating formalization results. Among these metrics, RUBY has the strongest correlation (> 0.85) with the other metrics.

Pass and BLEU metrics should be jointly used to prevent evaluation bias. Some zero-shot results in Table 1 lead to a high score on the Pass metric but lower scores on other metrics due toi internal LLM style biases. Syntactic correctness is one significant criteria in evaluation, but the aforementioned situation suggests that using Pass metric alone might include biases during evaluation. According to Figure 4, among metrics for semantic similarity, BLEU-2 has the strongest correlation with the Pass metric and hence can indicate syntactic correctness to some extent. We suggest considering both BLEU scores and Pass rate when comparing results.

5 Related Work

Automated Theorem Proving Automated Theorem Proving refers to the task of automatically generating a formal proof for a given mathematical statement (Wang and Deng, 2020). The typical approach to this task involves decomposing it into a multi-step generation problem, where at each step the model generates the next part of the proof given the current proof state (Polu and Sutskever, 2020; Wang and Deng, 2020; Han et al., 2022; Jiang et al., 2022). Our work on autoformalization supports such automated theorem proving efforts (Wu et al., 2022) by delivering a coherent formal representation that maintains the semantic integrity necessary for mathematical reasoning over mathematical libraries.

Retrieval Augmented Generation (Lewis et al., 2020) RAG has demonstrated improvements for code (Lu et al., 2022; Zhang et al., 2023). For formal language, Yang et al. (2023) trained a retrieval-augmented language model for formal premise selection and theorem proving. Meanwhile, our work focuses on utilizing RAG for the task of improving autoformalization performance and coherence with respect to mathematical libraries.

LLMs Refinement Through feedback-guided refinement strategies LLMs can self-correct (Pan et al., 2024). Recent studies (Madaan et al., 2023; Quan et al., 2024a) evaluate strategies using iterative feedback to refine LLM-generated answers for downstream tasks. Some work has utilized error messages generated by theorem provers for LLMs (Pan et al., 2023; Quan et al., 2024a; Jiang et al., 2024b; Quan et al., 2024b) or repair models (First et al., 2023a) to address syntactic or proof errors using these messages. Similarly, our work applies prompt-based refinement from external feedback error messages generated by Isabelle/ZF to iteratively refine the formalized logical forms with specific error code locations.

6 Conclusion

This paper examined the effects of using RAG for autoformalization with LLMs and explored methods to refine formalization results. Our experiments demonstrated the effectiveness of incorporating a retrieval process for autoformalization. Further refinement experiments indicated that denoising and iteratively refining syntax errors can enhance the formalization quality. We evaluated results on different LLMs and found that smaller LLMs with instruction fine-tuning benefited more from the proposed methods, pointing in the direction of serving as a mechanism for reducing the formal performance gaps between larger commercial models and smaller models. We also constructed a dataset and assessed metrics to evaluate autoformalization, which could serve as resources for formal mathematical reasoning tasks. We aim to develop more advanced prompting strategy and automated metrics for autoformalization as future directions.

Limitations

Some natural language statements in our dataset are too general or informal, failing to provide meaningful information for automated mathematics theorem proving. Although our proposed framework, Auto-SEF, enhances syntactic control in autoformalization, increasing iterations do not yield significant improvements in the Pass metric. This limitation is due to the inability of LLMs to generate syntactically correct complex formal representations.

References

Zhangir Azerbayev, Bartosz Piotrowski, Hailey Schoelkopf, Edward W. Ayers, Dragomir Radev, and Jeremy Avigad. 2023. Proofnet: Autoformalizing and formally proving undergraduate-level mathematics. *Preprint*, arXiv:2302.12433.

Zhangir Azerbayev, Hailey Schoelkopf, Keiran Paster, Marco Dos Santos, Stephen Marcus McAleer, Albert Q. Jiang, Jia Deng, Stella Biderman, and Sean Welleck. 2024. Llemma: An open language model for mathematics. In *The Twelfth International Conference on Learning Representations*.

Steven Bird and Edward Loper. 2004. NLTK: The natural language toolkit. In *Proceedings of the ACL Interactive Poster and Demonstration Sessions*, pages 214–217, Barcelona, Spain. Association for Computational Linguistics.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

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

Mikhail Evtikhiev, Egor Bogomolov, Yaroslav Sokolov, and Timofey Bryksin. 2023. Out of the bleu: How should we assess quality of the code generation models? *J. Syst. Softw.*, 203(C).

Emily First, Markus N. Rabe, Talia Ringer, and Yuriy Brun. 2023a. Baldur: Whole-proof generation and repair with large language models. *Preprint*, arXiv:2303.04910.

Emily C First, Markus Norman Rabe, Talia Ringer, and Yuriy Brun. 2023b. Baldur: Whole-proof generation and repair with large language models. *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*.

Jesse Michael Han, Jason Rute, Yuhuai Wu, Edward Ayers, and Stanislas Polu. 2022. Proof artifact co-training for theorem proving with language models. In *International Conference on Learning Representations*.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the MATH dataset. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023a. Mistral 7b. *Preprint*, arXiv:2310.06825.

Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, Gianna Lengyel, Guillaume Bour, Guillaume Lample, Lélio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, Szymon Antoniak, Teven Le Scao, Théophile Gervet, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2024a. Mixtral of experts. *Preprint*, arXiv:2401.04088.

Albert Qiaochu Jiang, Wenda Li, Szymon Tworkowski, Konrad Czechowski, Tomasz Odrzygóźdź, Piotr Mił oś, Yuhuai Wu, and Mateja Jamnik. 2022. Thor: Wielding hammers to integrate language models and automated theorem provers. In *Advances in Neural Information Processing Systems*, volume 35, pages 8360–8373. Curran Associates, Inc.

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

Dongwei Jiang, Marcio Fonseca, and Shay B. Cohen. 2024b. Leanreasoner: Boosting complex logical reasoning with lean. *Preprint*, arXiv:2403.13312.

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. 2020.

Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Advances in Neural Information Processing Systems*, volume 33, pages 9459–9474. Curran Associates, Inc.

Jiayu Liu, Zhenya Huang, ChengXiang Zhai, and Qi Liu. 2023. Learning by applying: A general framework for mathematical reasoning via enhancing explicit knowledge learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(4):4497–4506.

Pan Lu, Liang Qiu, Wenhao Yu, Sean Welleck, and Kai-Wei Chang. 2023. A survey of deep learning for mathematical reasoning. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 14605–14631, Toronto, Canada. Association for Computational Linguistics.

Shuai Lu, Nan Duan, Hojae Han, Daya Guo, Seung-won Hwang, and Alexey Svyatkovskiy. 2022. ReACC: A retrieval-augmented code completion framework. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6227–6240, Dublin, Ireland. Association for Computational Linguistics.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: Iterative refinement with self-feedback. *Preprint*, arXiv:2303.17651.

Jordan Meadows and André Freitas. 2023. Introduction to mathematical language processing: Informal proofs, word problems, and supporting tasks. *Transactions of the Association for Computational Linguistics*, 11:1162–1184.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*, volume 35, pages 27730–27744. Curran Associates, Inc.

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

Liangming Pan, Michael Saxon, Wenda Xu, Deepak Nathani, Xinyi Wang, and William Yang Wang. 2024. Automatically correcting large language models: Surveying the landscape of diverse automated correction strategies. *Transactions of the Association for Computational Linguistics*, 11:484–506.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

Lawrence C. Paulson. 2000. Isabelle: The next 700 theorem provers. *Preprint*, arXiv:cs/9301106.

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

Maja Popović. 2015. chrF: character n-gram F-score for automatic MT evaluation. In *Proceedings of the Tenth Workshop on Statistical Machine Translation*, pages 392–395, Lisbon, Portugal. Association for Computational Linguistics.

Xin Quan, Marco Valentino, Louise Dennis, and André Freitas. 2024a. Enhancing ethical explanations of large language models through iterative symbolic refinement. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1–22, St. Julian's, Malta. Association for Computational Linguistics.

Xin Quan, Marco Valentino, Louise A. Dennis, and André Freitas. 2024b. Verification and refinement of natural language explanations through llm-symbolic theorem proving. *Preprint*, arXiv:2405.01379.

Stephen E. Robertson, Steve Walker, Susan Jones, Micheline Hancock-Beaulieu, and Mike Gatford. 1994. Okapi at trec-3. In *TREC*, volume 500-225 of *NIST Special Publication*, pages 109–126. National Institute of Standards and Technology (NIST).

David Saxton, Edward Grefenstette, Felix Hill, and Pushmeet Kohli. 2019. Analysing mathematical reasoning abilities of neural models. *ArXiv*, abs/1904.01557.

Peter Scholze. 2022. Liquid tensor experiment. *Experimental Mathematics*, 31(2):349–354.

Boris Shminke. 2022. Python client for isabelle server. *Preprint*, arXiv:2212.11173.

Ngoc Tran, Hieu Tran, Son Nguyen, Hoan Nguyen, and Tien N. Nguyen. 2019. Does bleu score work for code migration? In *Proceedings of the 27th International Conference on Program Comprehension*, ICPC '19, page 165–176. IEEE Press.

Mingzhe Wang and Jia Deng. 2020. Learning to prove theorems by learning to generate theorems. In *Advances in Neural Information Processing Systems*, volume 33, pages 18146–18157. Curran Associates, Inc.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pages 24824–24837. Curran Associates, Inc.

Yuhuai Wu, Albert Qiaochu Jiang, Wenda Li, Markus Norman Rabe, Charles E Staats, Mateja Jamnik, and Christian Szegedy. 2022. Autoformalization with large language models. In *Advances in Neural Information Processing Systems*.

Huajian Xin, Haiming Wang, Chuanyang Zheng, Lin Li, Zhengying Liu, Qingxing Cao, Yinya Huang, Jing Xiong, Han Shi, Enze Xie, Jian Yin, Zhenguo Li, Xiaodan Liang, and Heng Liao. 2023. Lego-prover: Neural theorem proving with growing libraries. *ArXiv*, abs/2310.00656.

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

Kun Zhang, Xiexiong Lin, Yuanzhuo Wang, Xin Zhang, Fei Sun, Cen Jianhe, Hexiang Tan, Xuhui Jiang, and Huawei Shen. 2023. ReFSQL: A retrieval-augmentation framework for text-to-SQL generation. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 664–673, Singapore. Association for Computational Linguistics.

Kunhao Zheng, Jesse Michael Han, and Stanislas Polu. 2022. minif2f: a cross-system benchmark for formal olympiad-level mathematics. In *International Conference on Learning Representations*.

Shuyan Zhou, Uri Alon, Sumit Agarwal, and Graham Neubig. 2023. CodeBERTScore: Evaluating code generation with pretrained models of code. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 13921–13937, Singapore. Association for Computational Linguistics.

A Large Language Models

We describe large language models used in our experiments in this section.

Mistral (Jiang et al., 2023a) Mistral is a large language model with 7 billion parameters and finetuned on instruction datasets which are publicly available on the HuggingFace repository. It balances the trade-off of performance and efficiency and has the strongest performance among LLMs which have the similar scale with it. Mistral has shown strong performance in code, mathematics, and reasoning benchmark.

Llemma (Azerbayev et al., 2024) Llemma is an open large language model finetuned specifically for mathematics. It is pretrained on Proof-Pile-2 which is a diverse mixture of math-related text and code. However, it has not been trained to follow instructions. Llemma has two scales in 7B and 34B. We only use the 7B model in our experiments.

Mixtral (Jiang et al., 2024a) Mixtral is a large language model with sparse mixture of experts method and instruction finetuning. It has the same architecture as Mistral 7B but each layer of it consists of 8 feed-forward blocks. This makes it a 8×7B size. However, during inference, only 13B parameters are activated.

GPT-3.5-Turbo GPT-3.5-Turbo is one large language models of OpenAI GPT-3.5 series. It shares the same architecture as GPT-3 (Brown et al., 2020) and is finetuned with instructions. The number of parameters in GPT-3.5-Turbo is 175 billions.

B Evaluation Metrics

We describe the implementation of metrics to measure semantic similarity in this section.

BLEU (Papineni et al., 2002) The autoformalization task is a translation task so the most common metric in translation tasks, BLEU, is used as one evaluation metric for autoformalization. This metric is also used in (Wu et al., 2022). We use the implementation from NLTK (Bird and Loper, 2004).

ChrF (Popović, 2015) ChrF is another n-gram metric in translation task that focuses on characters instead of words in BLEU. We leverage this character level metric in NLTK to take character-level aspect into account.

RUBY (Tran et al., 2019) The autoformalization task is also a code generation task. RUBY is a metric designed specific for code generation evaluation that uses edit distance to calculate the score. If program dependence graph (PDG) or abstract syntax tree (AST) is provided, it calculates graph similarity based on graph edit distance or tree similarity based on tree edit distance. Otherwise, it calculates string edit distance to determine the string similarity between reference code and candidate code as the score. In our experiments, because of the difficulty of obtaining PDG or AST of formal statements, we use string edit distance from NLTK to calculate string similarity as the score. This implementation focuses on characters rather than tokens as in the original paper but it still makes the score a reasonable indicator of performance.

Translate the following Isabelle/ZF code: {statement} into a natural language version statement as brief as possible:

Table 4: Prompt for informalization.

Natural language version: {Natural Language Text} Translate the natural language version to an Isabelle/ZF version without any additional text and do not give any proof: {Formal Statement}

Table 5: Prompt for autoformalization.

CodeBERTScore (Zhou et al., 2023) CodeBERTScore is a model-based metric to evaluate performance on code generation. It uses token representations of reference code and candidate code to determine a final score. The original paper trained different models for different programming languages to get representations but Isabelle is not one of them. Therefore, we use a mathematical specific model Llemma 7B (Azerbayev et al., 2024) as the model to obtain representations. Although this model is not a BERT-based model, it can still generate meaningful representations for score calculation.

C Prompts

We provide prompts for informalization, autoformalization, denoising, and Auto-SEF in Table 4, 5, 6, 7, respectively.

D Detailed Results

We provide the exact number of scores of denoising in Table 8 and Auto-SEF in Table 9.

	Prompt
PBD 1A	You are an expert in Isabelle theorem prover. You will be provided with an Isabelle/ZF code generated by a language model. Your task is to clean the provided Isabelle/ZF code with following instructions. Instructions: 1. The provided code might contain several lemmas or definitions or theorems. The cleaned code must only keep the best one lemma or definition or theorem. 2. Do not write any proof and if there is a proof in the provided code, remove it from the cleaned code. 3. You should only output tokens that compose the cleaned code. Anything else, including but not limited to note, description, explanation and comment, must be removed from the final answer. Giving any additional text is prohibited. Strictly follow the instructions that I have claimed. Provided Isabelle/ZF Code: {isabelle code} Cleaned Code:
PBD 1B	1A + An additional instruction: 4. The cleaned code must have the same style and usage of operators as the original provided code. Operators usually start with “\” such as “\<in>”, “\<cdot>”.
PBD 1C	1A + An additional instruction: 4. The cleaned code must have the same style and usage of operators as the original provided code. Operators usually start with “\” such as “\<in>”, “\<cdot>”. Here are some additional Isabelle/ZF code examples which have the same style as the original provided code: {fixed 3-shot formal statements}
PBD 1D	1A + An additional instruction: 4. The cleaned code must have the same style and usage of operators as the original provided code. Operators usually start with “\” such as “\<in>”, “\<cdot>”. Here are some additional Isabelle/ZF code examples which have the same style as the original provided code: {retrieved 3-shot formal statements}

Table 6: Prompts for informalization.

You are an expert in Isabelle theorem prover. You will be provided with an Isabelle/ZF code generated by a language model. The provided code has some Isabelle/ZF syntax errors according to the Isabelle prover. You will also be provided with the error details and where the error code is located in the code. Your task is to fix related errors in the provided Isabelle/ZF code with following instructions. Instructions: 1. Only refine the code part which is related to provided error details. You must keep other code parts unchanged. 2. The syntax errors might cause by the mismatch of brackets, incorrect using of operators or invalid representation of Isabelle/ZF code. You should only refine the error codes based on the error details by rewriting, fixing or removing error codes. 3. You should only output tokens that compose the cleaned code. Anything else, including but not limited to note, description, explanation and comment, must be removed from the final answer. Giving any additional text is prohibited. 4. The cleaned code must have the same style and usage of operators as the original provided code. Operators usually start with “\” such as “\<in>”, “\<cdot>”. Here are some additional Isabelle/ZF code examples which have the same style as the original provided code: {retrieved 3-shot formal statements} Strictly follow the instructions that I have claimed. Provided Isabelle/ZF Code: {isabelle code} {first syntax error details} Refined Code:

Table 7: Auto-SEF prompt.

LLM	Method	BLEU-2	ChrF	RUBY	CBS	Pass
Mistral	Retrieval 3-shot	6.33	48.45	28.99	76.40	17.15
Mistral	Retrieval 3-shot+CBD	10.05	**51.38**	44.82	**76.93**	21.53
Mistral	PBD 1A	8.88	38.27	38.23	68.04	6.57
Mistral	PBD 1A+CBD	10.49	37.92	40.35	68.01	6.57
Mistral	PBD 1B	11.30	43.25	42.08	70.51	9.12
Mistral	PBD 1B+CBD	13.29	43.19	43.99	70.44	9.12
Mistral	PBD 1C	15.21	44.52	44.59	71.92	13.50
Mistral	PBD 1C+CBD	16.70	44.29	45.38	71.93	13.87
Mistral	PBD 1D	14.90	48.51	46.43	74.07	28.10
Mistral	PBD 1D+CBD	**17.32**	48.62	**47.41**	74.10	28.10
GPT-3.5-Turbo	Retrieval 3-shot	36.06	**59.70**	**58.56**	**79.34**	**64.96**
GPT-3.5-Turbo	Retrieval 3-shot+CBD	36.32	59.63	58.51	79.14	64.60
GPT-3.5-Turbo	PBD 1A	**38.60**	57.90	58.16	78.79	63.87
GPT-3.5-Turbo	PBD 1A+CBD	38.59	57.86	58.12	78.63	63.87
GPT-3.5-Turbo	PBD 1B	36.49	57.08	57.79	78.27	62.04
GPT-3.5-Turbo	PBD 1B+CBD	36.49	57.08	57.79	78.27	62.04
GPT-3.5-Turbo	PBD 1C	37.10	57.28	57.83	78.62	63.50
GPT-3.5-Turbo	PBD 1C+CBD	37.10	57.28	57.83	78.62	63.50
GPT-3.5-Turbo	PBD 1D	38.50	58.09	58.17	78.99	64.60
GPT-3.5-Turbo	PBD 1D+CBD	38.50	58.09	58.17	78.99	64.60

Table 8: The effect of denoising.

LLM	Method	BLEU-2	ChrF	RUBY	CBS	Pass
Mistral	Iteration1	14.91	45.69	44.16	72.22	30.66
Mistral	Iteration2	13.23	44.84	43.72	72.04	32.12
Mistral	Iteration3	12.69	44.10	42.19	71.63	32.48
Mistral	Iteration4	11.29	44.18	42.30	71.53	32.85
Mistral	Iteration5	11.91	43.57	41.72	71.06	33.58
Mistral	Iteration6	11.87	43.48	41.69	71.09	33.58
Mistral	Iteration7	11.72	43.64	41.26	70.91	33.58
Mistral	Iteration8	11.10	43.24	41.55	71.00	33.58
Mistral	Iteration9	11.17	43.09	40.85	70.80	33.58
GPT-3.5-Turbo	Iteration1	38.11	57.66	57.45	78.71	68.98
GPT-3.5-Turbo	Iteration2	38.10	57.55	57.55	78.47	69.71
GPT-3.5-Turbo	Iteration3	38.09	57.55	57.57	78.48	70.07
GPT-3.5-Turbo	Iteration4	37.99	57.54	57.50	78.45	70.44
GPT-3.5-Turbo	Iteration5	38.08	57.58	57.62	78.51	70.44
GPT-3.5-Turbo	Iteration6	38.05	57.57	57.46	78.47	70.44
GPT-3.5-Turbo	Iteration7	38.00	57.53	57.39	78.49	70.44
GPT-3.5-Turbo	Iteration8	37.99	57.57	57.37	78.50	70.80
GPT-3.5-Turbo	Iteration9	38.01	57.55	57.42	78.48	70.80

Table 9: Auto-SEF results with CBD applied.