Improving Autoformalization Using Direct Dependency Retrieval

Anonymous ACL submission

Abstract

Statement autoformalization, a crucial first step in formal verification, aims to transform informal descriptions of math problems into machine-verifiable formal representations but remains a significant challenge. The core difficulty lies in the fact that existing language models hallucinate formal dependencies, including missing or incorrect definitions, lemmas, and theorems. Current dependency retrieval approaches exhibit poor precision and recall, and lack the scalability to leverage ever-growing public datasets. To bridge this gap, we propose a novel retrieval-augmented framework based on Direct Dependency Retrieval (DDR). DDR directly generates candidate formal dependencies from natural-language mathematical descriptions and verifies their existence in the formal library via an efficient Suffix Array Check (SAC). Built on a SAC-constructed dependency retrieval dataset of over 500,000 samples, a high-precision DDR model is fine-tuned and shown to significantly outperform state-of-the-art methods in both retrieval precision and recall, leading to superior advantage in the autoformalization tasks. SAC also contributes in assessing formalization difficulty and enabling explicit quantification of the hallucination in In-Context Learning (ICL).

1 Introduction

Theorem provers, such as Lean (Moura and Ullrich, 2021), Coq (Bertot and Castéran, 2013), and Isabelle (Nipkow et al., 2002), are symbolic systems designed for formal verification whose soundness and completeness are proven. Grounded in specific foundational logics, these systems employ proprietary formal languages to encode mathematical statements, definitions, and proofs. The logical kernel within each system can then verify the validity of these formalized objects with complete rigor (Nawaz et al., 2019). However, this process eschews the flexibility of natural language, demanding that users not only master the strict formal syntax but also possess a profound understanding of the semantic mapping between mathematical ideas and their formal representations. Consequently, the effective utilization of these theorem provers presents a formidable challenge to researchers (Li et al., 2024).

Statement autoformalization refers to the process of automatically translating mathematical statements from natural language into a formal language comprehensible to theorem provers (Liu et al., 2025b). As a foundational step in the formal verification, the correctness of its output is paramount. It determines whether the proof target is precisely aligned with the intent of the original statement. Beyond this core function, statement autoformalization exhibits significant potential for broader applications and innovation. These applications include synthesizing training data for neural theorem provers (Lin et al., 2025), automated program verification (Lin et al., 2024), and curating and filtering informal reasoning datasets (Zhang et al., 2025). , and evaluating and enhancing the informal reasoning capabilities of LLMs (e.g., through techniques like rejection sampling (Zhou et al., 2024)).

Current mainstream research in autoformalization primarily encompasses three direction. First, constructing larger and more challenging datasets for Supervised Fine-Tuning (SFT) (Cao et al., 2025; Peng et al., 2025; Yu et al., 2025; Miranda et al., 2025; Wang et al., 2025a; Ying et al., 2024a; Zheng et al., 2021; Azerbayev et al., 2023); second, encoding and indexing the libraries of theorem provers to support retrieval (Liu et al., 2025b; Yang et al., 2023; Wang et al., 2025b); and third, employing methods like ICL to enable Large Language Models (LLMs) to directly translate informal statements into formal code (Wu et al., 2022; Zhang et al., 2024).

However, these approaches face two critical lim-

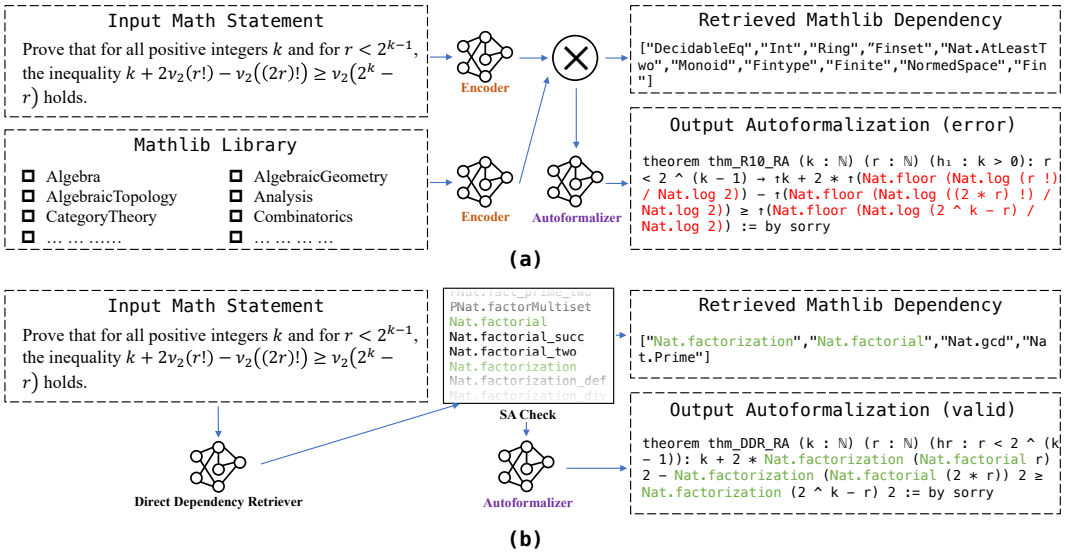

Figure 1: Comparison of two paradigms for formal dependency retrieval. The $v_2(\cdot)$ in input indicates 2-adic valuation. (a) Select-based retrieval using embedding cosine similarity: both the input statement and library items are encoded into embeddings, and the top-k items with highest cosine similarity are retrieved as dependencies. (b) DDR generates potential dependencies based on context, followed by SAC verification step.

itations. First, relying solely on generative models to directly synthesize formal code often overlooks the existing definitions and lemmas within the formal library (Yang et al., 2023; Zhang et al., 2024). This ignorance is a primary reason for the model to hallucinate non-existent formal objects or generate syntactically incorrect code (Wu et al., 2022; Liu et al., 2025b). Results presented in Table 2 demonstrate that even with a two-stage ICL approach, where dependencies are retrieved prior to the main task, the problem of hallucination in model outputs remains significant. Second, the retrieval strategies for library dependencies in existing methods are rudimentary, yielding results with weak semantic relevance to the informal statement (Azerbayev et al., 2023; Yang et al., 2023). The libraries of mainstream theorem provers are vast and continually growing; for instance, Lean 4's mathlib contains over 240,000 entries, including many with similar descriptions and usage. Simple retrieval methods struggle to achieve both high efficiency and precision. This challenge is confirmed by the experiments in Liu et al. (2025b) as well as our own results presented in Table 3.

To address the aforementioned challenges, we propose a novel dependency retrieval method DDR, and the corresponding framework for statement autoformalization. In this context, dependency retrieval aims to identify relevant formal objects, such as definitions and theorems, from a theorem prover's library based on an informal mathematical statement (Liu et al., 2025b). DDR leverages the contextual understanding of LLMs to directly generate potential formal dependencies. To empower the LLM with this capability, we have designed SAC for both data generation and dependency verification. Specifically, this process begins by constructing an efficient database from the theorem prover's library using a suffix array. Subsequently, leveraging existing informal-formal statement pairs from public datasets, we extract candidate dependencies from the formal code and verify them against the database via the efficient binary search. This produces high-quality labels for fine-tuning the DDR model. The same verification mechanism is reused at inference time to validate the dependencies generated by the DDR model. Experimental results demonstrate that DDR significantly improves both recall and precision in dependency retrieval, leading to superior advantage in the autoformalization tasks.

Our main contributions are summarized as follows:

1. We propose DDR as a novel generation-plus-verification dependency retrieval paradigm in autoformalization tasks, and SAC as a flexible dependency verification method.

2. We provide a quantitative analysis of ICL for dependency retrieval, revealing its inherent and

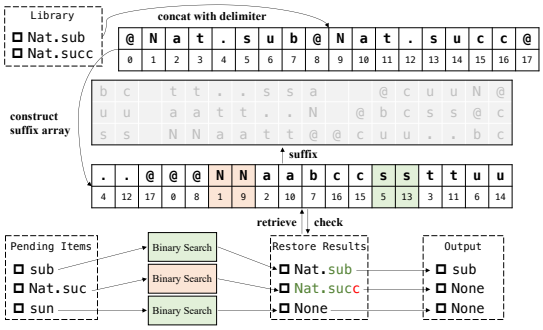

Figure 2: Schematic diagram of SAC process. All library items are concatenated into a single string with delimiters, followed by suffix array construction. Each pending item is then queried via binary search to determine its match status (exact, partial, or none). This mechanism supports both dataset construction and DDR output verification.

severe hallucination problems.

3. We conduct comprehensive experiments on statement autoformalization. The results validate that DDR can serve as a plug-and-play module to enhance the performance of existing autoformalizers and ICL methods.

4. The proposed SAC provides a potential sound mechanism for assessing formalization difficulty, overcomes the inherent subjectivity and stochasticity of LLM-based graders.

5. We have constructed and open-sourced a library dependency dataset with over 500,000 samples. The methodology for constructing this dataset is generalizable and can be applied to other formal corpora, fostering more diverse research in statement autoformalization.

2 Related Work

Formal mathematics aims to translate mathematical statements and proofs into machine-verifiable formal languages based on rigorous logical calculus. In recent years, with the advancement of deep learning, this field has become an active area of research, primarily encompassing two core branches: autoformalization (Murphy et al., 2024; Liu et al., 2025b; Poiroux et al., 2024; Azerbayev et al., 2023; Wang et al., 2020), and automated theorem proving (Lin et al., 2025; Wang et al., 2025a; Chen et al., 2025; Xin et al., 2024a,b; Ren et al., 2025; Liu et al., 2025a).

Automated theorem proving is dedicated to automatically generating proof scripts for given formal goals. The central challenge lies in designing efficient proof search strategies (Liu et al., 2025a). Current cutting-edge research in this direction focuses on combining LLMs with reinforcement learning to emulate the heuristic reasoning processes of human mathematicians in exploring the proof space (Chen et al., 2025), with the future prospect of assisting in the discovery of new theorems. Prominent examples of such systems include Seed-Prover (Chen et al., 2025) and DeepSeek-Prover (Xin et al., 2024a,b; Ren et al., 2025). In contrast, autoformalization targets the automatic translation of informal mathematical content, typically expressed in natural language, into a formal language. Research in this area concentrates on establishing reliable evaluation methodologies to assess the semantic fidelity of the translation (Liu et al., 2025b; Poiroux et al., 2024) and on developing higher-performance autoformalization models (Liu et al., 2025b; Poiroux et al., 2024). This paper focuses on the task of statement autoformalization. We propose a method to mitigate model hallucination and enhance model performance by offering better dependency retrieval results in the RAG framework.

Progress in autoformalization research has led to the creation of large-scale informal-to-formal parallel corpora (Mahdavi et al., 2025). These corpora, such as FormalMath (Yu et al., 2025), CriticBench (Peng et al., 2025), and FineLeanCorpus (Peng et al., 2025), have expanded in scale from a few hundred samples to tens of thousands and are primarily generated through rule-based methods and by leveraging the ICL capabilities of LLMs (Azerbayev et al., 2023; Gao et al., 2024). However, current autoformalization research has not effectively utilized this data to discover the intricate connection between informal mathematical statements and their required library dependencies. To address this gap, we propose DDR, a method designed to enhance a model's ability to infer and utilize library dependencies during statement formalization.

Research on evaluating the fidelity of autoformalization has explored various methods, such as manual expert review (Wang et al., 2018; Peng et al., 2025), LLM review (Yang et al., 2023; Sidhu et al.), auxiliary metrics like Perplexity (Wang et al., 2018), BLEU (Papineni et al., 2002; Wang et al., 2018), and type checking (Poiroux et al., 2024) within a theorem prover. However, these approaches have inherent limitations in capturing the deep semantic equivalence between informal and formal mathematical statements. They may

overlook semantic inconsistencies between the two modalities or produce false negatives by incorrectly rejecting semantically valid formalizations (Liu et al., 2025b). To overcome this, our work adopts Bidirectional Extender Definitional Equivalence (BEq) as the primary evaluation metric (Liu et al., 2025b). This choice is motivated by BEq's high alignment with human expert judgment in assessing semantic equivalence. The core principle of BEq is to define two formal statements as equivalent if they are mutually transformable through a predefined, finite, and verifiable set of transformation rules (Aczel et al., 2013).

3 Methodology

3.1 Direct Dependency Retrieval

To prevent hallucination during generation, existing dependency retrieval methods select items from a pre-existing library to form their output (Yang et al., 2023; Liu et al., 2025b), as illustrated in Figure 1 (a). However, this selected-based paradigm suffers from two primary limitations. First, training an encoder to produce effective embeddings for every formal object in the library to maximize retrieval performance is a challenge in itself (Xiong et al., 2020; Gao et al., 2023). Second, the method is sensitive to superficial variations in the input: semantically equivalent mathematical statements may yield different results in maximum cosine similarity retrieval due to minor differences in their syntactic structure or notational representation (Asl et al., 2024). Our empirical results in Table 3 confirm that such embedding-based retrieval methods exhibit low recall and precision. Inaccurate retrieval results don't contribute to the language model in autoformalization tasks. Figure 1 (a) illustrates this failure mode: in the absence of a usable dependency from the retrieved Mathlib candidates, the model attempts to construct a local function functionally equivalent to `factorization`, leading to an erroneous output. Furthermore, this approach lacks scalability, as it cannot effectively leverage the growing volume of available datasets to continuously enhance model capabilities.

To address the challenge of LLMs hallucinating non-existent or irrelevant library dependencies during autoformalization, we propose a generation-plus-verification pipeline, with DDR as its core component. DDR is a SFT model to operate in an end-to-end fashion, directly extracting potential library dependencies from an input mathematical statement. The SFT based paradigm enables it to effectively absorb and leverage continuously growing training data, thereby demonstrating excellent scalability. In contrast to conventional two-stage retrieval methods that require additional encoder model (Liu et al., 2025b), the design of DDR streamlines the architecture, leading to a simpler, more efficient, and better-performing approach. Accurate dependency retrieval reduces the complexity of the subsequent formalization task, thereby enhancing the overall performance of the autoformalizer. As illustrated in Figure 1 (b), providing the model with precise dependencies enables it to more reliably generate the correct formal output, thus increasing the overall success rate.

Difficulty	Num	Depend Rate	Depend Length
0	835	0.188	0.267
1	120040	0.555	1.061
2	113816	0.784	1.863
3	49841	0.807	2.079
4	69882	0.795	1.958
5	67547	0.523	1.106
6	41312	0.811	2.230
7	36675	0.754	2.108
8	8682	0.820	2.393
9	708	0.825	2.612

Table 1: Dataset statistics of FineLeanCorpus. **Depend Rate** denotes the percentage of samples that have explicit mathlib dependencies within each difficulty level. **Depend Length** denotes the mean number of explicit mathlib dependency items for samples within each difficulty level. Higher **Depend Rate** and **Depend Length** values typically indicate more challenging tasks.

The effective application of the DDR method entails two primary challenges: first, constructing a high-quality dataset for model fine-tuning, and second, implementing an output filtering mechanism to validate the generated dependencies. Such challenges can be addressed by the following SAC method.

3.2 Suffix Array Check

To accurately and efficiently verify the existence of dependency objects generated by LLMs within a formal library, we introduce a checking method based on suffix arrays (Liu et al., 2024). By definition, a suffix array is an array of integers that stores the starting indices of all suffixes of the original string, sorted in lexicographical order (Kärkkäinen et al., 2006). The sorted nature of the suffix array facilitates efficient matching via binary search.

Method	Diff01		Diff23		Diff45		Diff67		Diff89	
	$\textbf{Hall}_{\textbf{m}}$	$\textbf{Hall}_{\textbf{std}}$	$\textbf{Hall}_{\textbf{m}}$	$\textbf{Hall}_{\textbf{std}}$	$\textbf{Hall}_{\textbf{m}}$	$\textbf{Hall}_{\textbf{std}}$	$\textbf{Hall}_{\textbf{m}}$	$\textbf{Hall}_{\textbf{std}}$	$\textbf{Hall}_{\textbf{m}}$	$\textbf{Hall}_{\textbf{std}}$
Claude-3.5-Sonnet	0.35	0.32	0.30	0.26	0.28	0.26	0.28	0.22	0.28	0.22
DeepSeek-R1	0.23	0.28	0.33	0.31	0.27	0.29	0.31	0.30	0.30	0.29
GPT-4o	0.29	0.28	0.30	0.27	0.29	0.27	0.29	0.28	0.30	0.26
Ling-flash-2.0	0.33	0.32	0.39	0.31	0.34	0.30	0.33	0.28	0.36	0.28
Qwen-Max	0.23	0.30	0.27	0.31	0.21	0.24	0.24	0.26	0.25	0.24
R@5	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
R@10	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
DDR	0.02	0.08	0.01	0.04	0.00	0.03	0.00	0.02	0.01	0.09

Table 2: Hallucination rates in retrieved dependency results. $\textbf{Hall}_{\textbf{m}}$ measures the mean proportion of hallucinated items per sample in retrieved results; $\textbf{Hall}_{\textbf{std}}$ measures the standard deviation. **R@k** denotes the method from Liu et al. (2025b), where the top-k retrieved results are used as dependency outputs.y

SAC first concatenates the identifiers of all defined objects in the library, such as pre-proven theorems, into a single long string using a specific delimiter. Subsequently, a suffix array is constructed from this concatenated long string in linear complexity (Lee et al., 2022; Kärkkäinen et al., 2006). Figure 2 illustrates the SAC process with a simplified example, assuming the library contains only two objects and there are three pending items. For each pending item, we can locate the range of all suffixes that have the item as a prefix within the suffix array. This process enables the rapid retrieval of all library objects corresponding to the pending items, thereby completing their existence verification.

SAC is an efficient method for verification, capable of meeting the real-time requirements of large-scale requests (Liu et al., 2024). For a formal library containing N objects with an average length of d, and M pending items with an average length of s, conventional string matching methods typically involve brute-force comparison, resulting in a time complexity of $\mathcal{O}\left(sdMN\right)$. In contrast, the SAC operates on a suffix array constructed over the entire library. For each pending item, it utilizes binary search on the suffix array to locate matches. The overall time complexity of this approach is $\mathcal{O}(sM\log(Nd))$. This significant reduction in computational complexity enables real-time responsiveness (Liu et al., 2024).

3.3 Constructing High-quality Training Dataset

Fine-tuning a precise and efficient DDR model necessitates a dataset of pairs, each comprising a mathematical statement and its corresponding library dependencies. However, no such public dataset is currently available, and its construction is primarily challenged by the difficulty of accurately extracting and efficiently verifying library dependencies from existing formal statements. To address this, we employ the above SAC method to process a large-scale autoformalization corpus to generate the required training data. We select FineLeanCorpus (Peng et al., 2025) as the source corpus, which contains over 500,000 samples spanning a wide range of mathematical domains and difficulty levels. After partitioning the corpus into training and test sets based on problem difficulty, we leverage the efficiency of SAC to process the entire training partition in under two hours. This process yield a high-quality dataset suitable for fine-tuning the DDR model. This resulting dataset is anonymously available at anonymous github. We anticipate that this contribution will foster further research on the field of formal mathematical proving.

4 Direct Dependency Retrieval-augmented Autoformalization

This section presents a comprehensive evaluation of the proposed method. Our evaluation is twofold: first, we directly assess the performance of DDR on the dependency retrieval task. Second, we measure the improvements it brings by evaluating the pass@8 success rate on the downstream autoformalization task. Furthermore, the experiments provide additional insights, such as a discussion on hallucinations in ICL methods and an analysis of the difficulty of autoformalization datasets.

4.1 Dataset Overview and Difficulty Analysis

The difficulty attribute is a key metric in FineLeanCorpus designed to quantify the challenge of for-

Method	Diff01		Diff23		Diff45		Diff67		Diff89	
	Precision	Recall	Precision	Recall	Precision	Recall	Precision	Recall	Precision	Recall
Claude-3.5-Sonnet	0.49	0.52	0.28	0.38	0.21	0.34	0.28	0.43	0.27	0.33
DeepSeek-R1	0.53	0.56	0.24	0.35	0.19	0.32	0.23	0.37	0.26	0.37
GPT-4o	0.54	0.55	0.28	0.32	0.20	0.25	0.26	0.32	0.27	0.29
Ling-flash-2.0	0.45	0.47	0.17	0.24	0.15	0.22	0.21	0.32	0.18	0.25
Qwen-Max	0.48	0.51	0.26	0.29	0.23	0.28	0.26	0.33	0.27	0.32
R@5	0.02	0.07	0.07	0.15	0.05	0.12	0.08	0.17	0.09	0.14
R@10	0.02	0.10	0.04	0.19	0.03	0.15	0.06	0.27	0.06	0.21
DDR	**0.84**	**0.82**	**0.90**	**0.87**	**0.91**	**0.92**	**0.91**	**0.88**	**0.83**	**0.83**

Table 3: Comparison of the retrieval dependency results. Results shown are filtered (i.e., without hallucinations).

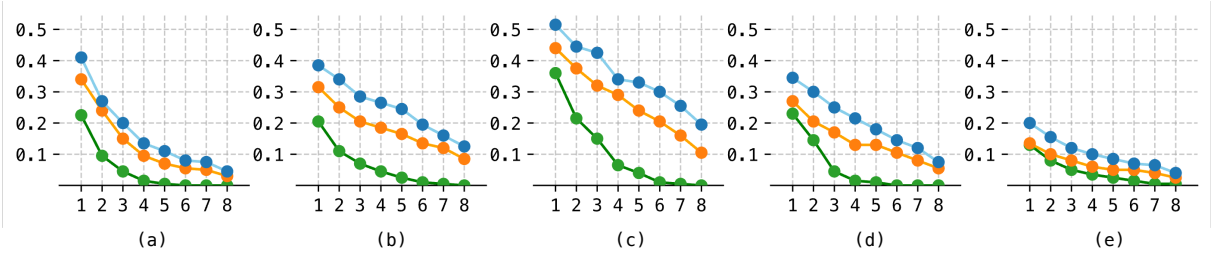

Figure 3: LLM consistency across multiple attempts. Subplots (a–e) correspond to the performance of DeepSeek-R1 on datasets Diff01, Diff23, Diff45, Diff67, and Diff89, respectively. Each subplot illustrates the proportion of problems (y axis) for which at least k (from 1 to 8, x axis) out of 8 attempts successfully pass the BEq verification. Dependency retrieval method: DDR (blue); ICL (orange); N/A (green).

malization, rated on an 11-point scale from 0 to 10. In our experimental setup, we consolidated samples with a difficulty of 10 into the difficulty 9 category. The statistical results of Mathlib dependency on datasets are summarized in Table 1.

We randomly sampled 100 statements from each difficulty category and then combined the samples from every two adjacent difficulty categories as the test set. This process resulted in five test sets, named **Diff01**, **Diff23**, **Diff45**, **Diff67**, and **Diff89**, where **Diff*ij*** denotes the combined set of samples from difficulty levels ***i*** and ***j***, with each test set containing 200 statements. All remaining samples in the dataset were used to fine-tune the DDR model.

The original difficulty ratings for the FineLeanCorpus (Peng et al., 2025) datasets, assigned by an LLM grader, exhibit a notable discrepancy with our experimental results in Table 4 and Table 5. To investigate this inconsistency, we propose assessing the actual formalization difficulty using two alternative metrics: **Depend Rate** and **Depend Length**. Based on the data in Table 1, we can calculate the weighted averages of **Depend Rate** and **Depend Length** for the **Diff23**, **Diff45**, **Diff67**, and **Diff89** subsets. The results are as follows: **Diff45** has the lowest values (0.662, 1.539), the values for **Diff23** (0.791, 1.929) and **Diff67** (0.784, 2.173) are higher, and **Diff89** has the highest values (0.820, 2.410). These metrics reveal a difficulty ranking of **Diff89 > Diff67 > Diff23 > Diff45**. This ranking aligns closely with our experimental observations, suggesting that **Depend Rate** and **Depend Length**, derived via our DDR and SAC, can serve as a potential indicator of formalization difficulty along with LLM-based graders. The **Diff01** subset is excluded from this analysis because its imprecise informal descriptions introduce extraneous formalization challenges, making its difficulty hard to assess with these metrics. Detailed discussion are presented in appendix B

4.2 Setup

Due to space constraints, we defer the detailed description of our experimental setup to the appendix C. Our evaluation utilizes two primary metrics: Type Checking (TypeC) and Bidirectional Extended Definitional Equality (BEq). TypeC examines the syntactic correctness of expressions. BEq ensures semantic equivalence between the output and ground truth. We compare our method against a comprehensive set of baseline methods, including MMA (Jiang et al., 2023), DeepSeek-R1 (Guo et al., 2025), Ling-flash-2.0 (AI et al., 2025), Claude-3.5-Sonnet (Anthropic, 2024), GPT-4o (Achiam et al., 2023), Qwen-Max (Qwen et al., 2025), as well as RA+R@5 and RA+R@10 (Liu

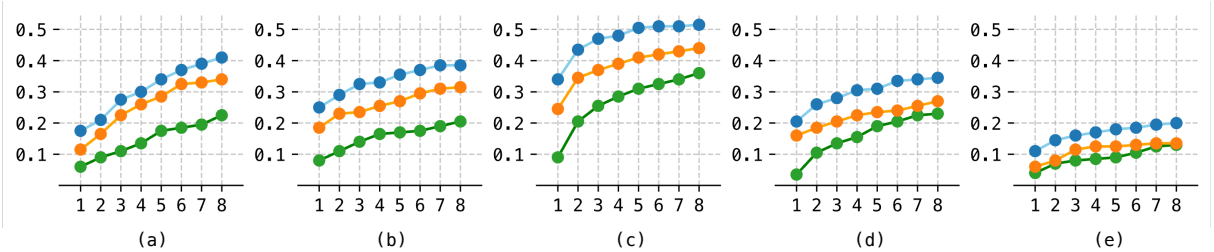

Figure 4: LLM efficiency across multiple attempts. Subplots (a–e) correspond to the performance of DeepSeek-R1 on datasets Diff01, Diff23, Diff45, Diff67, and Diff89, respectively. Each subplot illustrates the intermediate pass@k score (y axis) out of 8 attempts using BEq verification, k range from 1 to 8 (x axis). Dependency retrieval method: DDR (blue); ICL (orange); N/A (green).

Method	Diff01		Diff23		Diff45		Diff67		Diff89	
	TypeC	BEq	TypeC	BEq	TypeC	BEq	TypeC	BEq	TypeC	BEq
MMA	0.470	0.095	0.455	0.110	0.600	0.190	0.500	0.100	0.385	0.065
RA	0.670	0.115	0.730	0.170	0.765	0.295	0.735	0.185	0.610	0.105
RA+R@5	0.860	0.185	0.795	0.180	0.805	0.145	0.825	0.115	0.765	0.135
RA+R@10	**0.880**	0.205	0.860	0.200	**0.910**	0.255	0.825	0.160	0.765	**0.150**
RA+DDR	0.865	**0.235**	**0.900**	**0.215**	**0.910**	**0.305**	**0.865**	**0.225**	**0.770**	**0.150**

Table 4: Comparison of the autoformalization results using SFT methods. **MMA** denotes the LLM fine-tuned on MMA's Lean subset. **RA** refers to the pure autoformalization method from Liu et al. (2025b), while **R@5** and **R@10** denote its retrieval-augmented variants using the top-5 and top-10 retrieved results, respectively. **RA+DDR** denotes the new retrieval-augmented variant incorporating the proposed DDR.

et al., 2025b).

4.3 Hallucination Study

Prior to the introduction of DDR, a sound mechanism for assessing dependency hallucinations in the outputs of LLMs on autoformalization tasks was lacking. The proposed DDR framework, through the integrated SAC, provides a practical and efficient solution to this problem. This method determines whether each generated dependency exists within the formal library, thereby enabling the quantification of hallucination severity. Define the hallucination rate (**Hall**) as the proportion of hallucinatory items in a given set of dependencies. For instance, if a model outputs (`Real.div`, `Real.max`, `Nat.succ`) and only `Nat.succ` is authentic, the **Hall** is 2/3. $\textbf{Hall}_{\textbf{m}}$ and $\textbf{Hall}_{\textbf{std}}$ denote the mean and standard deviation of this rate across the test set, respectively. The hallucination rates for various mainstream dependency retrieval methods are presented in Table 2.

The analysis reveals that R@5 and R@10, as select-based methods, are inherently free from hallucination by design, since their outputs are drawn directly from the formal library. In contrast, all generative retrieval methods are susceptible to hallucination. Our proposed DDR method exhibits outstanding performance, with a mean hallucination rate ($\textbf{Hall}_{\textbf{m}}$) below 0.02 and a standard deviation ($\textbf{Hall}_{\textbf{std}}$) below 0.09, indicating nearly non-existent hallucination. In stark contrast, other ICL-based methods suffer from severe hallucination, with both the mean and standard deviation of their **Hall** values hovering around 0.30. This indicates that the performance of the ICL method in dependency retrieval is highly polarized: for some samples, it generates valid dependencies, whereas for others, its outputs consist predominantly of hallucinations.

4.4 Retrieval Results

Although R@5 and R@10 are hallucination-free, their retrieval performance is questionable. To isolate retrieval quality from the issue of hallucination and enable a fair comparison of the methods' true capabilities, we first filtered out all hallucinatory items from each method's output and then recalculated their precision and recall. The performance comparison after filtering is shown in Table 3.

After filtering, the results show substantial performance disparities among the models. Despite having no hallucinations, R@5 and R@10 yield the lowest precision (0.02-0.10) and recall (0.10-0.20),

Method	RAG	Diff01		Diff23		Diff45		Diff67		Diff89	
		TypeC	BEq	TypeC	BEq	TypeC	BEq	TypeC	BEq	TypeC	BEq
Claude-3.5-Sonnet	N/A	0.830	0.415	0.740	0.330	**0.775**	0.385	**0.680**	0.205	0.530	0.100
Claude-3.5-Sonnet	ICL	0.740	0.370	0.515	0.245	0.640	0.330	0.455	0.145	0.385	0.110
Claude-3.5-Sonnet	DDR	**0.905**	**0.460**	**0.770**	**0.335**	0.765	**0.425**	0.625	**0.220**	**0.630**	**0.130**
DeepSeek-R1	N/A	0.910	0.225	0.745	0.205	0.820	0.360	0.780	0.230	0.660	0.130
DeepSeek-R1	ICL	0.880	0.340	0.755	0.315	0.825	0.440	0.760	0.270	0.570	0.135
DeepSeek-R1	DDR	**0.940**	**0.410**	**0.875**	**0.385**	**0.940**	**0.515**	**0.875**	**0.345**	**0.695**	**0.200**
GPT-4o	N/A	0.905	0.440	0.760	0.295	0.800	0.400	0.745	0.240	0.575	0.135
GPT-4o	ICL	0.880	**0.460**	0.675	0.255	0.730	0.345	0.625	0.210	0.405	0.120
GPT-4o	DDR	**0.940**	**0.460**	**0.775**	**0.320**	**0.850**	**0.455**	**0.780**	**0.275**	**0.630**	**0.150**
Ling-flash-2.0	N/A	**0.850**	**0.405**	**0.610**	0.210	**0.665**	0.300	0.510	0.190	0.380	0.085
Ling-flash-2.0	ICL	0.775	0.355	0.420	0.170	0.590	0.280	0.395	0.170	0.295	0.085
Ling-flash-2.0	DDR	0.775	0.400	0.595	**0.225**	0.630	**0.360**	**0.535**	**0.205**	**0.440**	**0.115**
Qwen-Max	N/A	0.795	**0.375**	0.605	0.220	**0.665**	**0.360**	0.620	0.205	0.440	0.105
Qwen-Max	ICL	0.730	0.315	0.445	0.160	0.550	0.260	0.445	0.165	0.350	0.115
Qwen-Max	DDR	**0.830**	**0.375**	**0.660**	**0.250**	0.560	0.285	**0.695**	**0.255**	**0.545**	**0.120**

Table 5: Comparison of the autoformalization results using ICL methods. **RAG** column indicates the dependency retrieval strategy used in autoformalization: N/A denotes no retrieval, ICL denotes prompt-based retrieval, and DDR refers to the proposed method.

indicating their retrieval capability is severely limited. The filtered prompt-based methods achieve moderate performance, with precision scores ranging from 0.2 to 0.5 and recall scores from 0.3 to 0.5. Our proposed DDR method demonstrates the most superior performance, achieving a high precision of 0.8-0.9 and a recall of 0.8-0.9, significantly outperforming all other baseline methods.

4.5 Autoformalization Results

The efficacy of DDR is further substantiated by its performance improvements in the downstream autoformalization task. As shown in Table 4 and Table 5, integrating DDR yields significant performance gains under both SFT and ICL frameworks. We specifically evaluated the combination of DDR with the RA model, an advanced model fine-tuned for autoformalization in Liu et al. (2025b). A format mismatch exists, as the RA model requires a comprehensive input format (full dependency name, informal description, and code), whereas DDR outputs only abbreviated dependency names. Consequently, the R@5 and R@10 retrievers from Liu et al. (2025b), designed for RA, are more format-compatible. Despite this mismatch, the combination of RA and DDR consistently outperforms all baseline methods. This success is primarily attributed to DDR's superior retrieval precision and recall, highlighting its excellent generalization and robustness. In contrast, the performance of R@5 and R@10 reflects a trade-off between precision and recall, consistent with their respective retrieval characteristics.

To evaluate DDR's potential as a plug-and-play component, we integrated it with ICL methods. Table 5 compares three experimental setups: (1) direct autoformalization, (2) two-stage autoformalization using a prompt-based dependency retrieval, and (3) two-stage autoformalization using a DDR-based dependency retrieval. The results indicate that the DDR-based RAG is the superior strategy in most test scenarios. Experiments on Deepseek-R1, illustrated in Figure 3, further reveal DDR's stability advantage: the DDR-integrated method not only achieves a higher total pass@8 success count but also exhibits a significantly better distribution of successful attempts across 8 independent trials compared to the baseline. Furthermore, Figure 4 shows that DDR can complete more tasks with fewer attempts.

5 Conclusion

We propose DDR, a method that leverages the contextual understanding of LLMs to directly generate candidate dependencies for a given informal statement in autoformalization tasks. This is complemented by SAC, which utilize suffix array for efficient dependency verification. Experimental results validate the effectiveness of our approach and demonstrate its potential in ICL hallucinations analysis and in assessing the difficulty of autoformalization datasets.

6 Limitations

A key limitation of DDR is its inability to precisely locate the generated items within the library. Specifically, whereas the full retrieval results in methods like Liu et al. (2025b) typically include complete metadata such as the item's name, code usage, and documentation, DDR's potential to generate abbreviated or shortcut names precludes the retrieval of this supplementary information. This discrepancy renders DDR not directly compatible with the retrieval-augmented autoformalizer framework in Liu et al. (2025b).

DDR also exhibits limitations in handling certain special cases, which necessitate dedicated mechanisms. For instance, some core constructs (such as `MOD`), despite being crucial and frequently used in formalization, may not exist as standard library items

Formal mathematics libraries are continuously evolving, with dependency items being added, removed, or renamed across different versions. Such changes require corresponding maintenance of our system to ensure its continued functionality and effectiveness.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Peter Aczel, Benedikt Ahrens, Thorsten Altenkirch, Steve Awodey, Bruno Barras, Andrej Bauer, Yves Bertot, Marc Bezem, Thierry Coquand, Eric Finster, and 1 others. 2013. *Homotopy type theory: univalent foundations of mathematics*. The Univalent Foundations Program Institute for Advanced Study.

Inclusion AI, Bowen Ma, Cheng Zou, Canxiang Yan, Chunxiang Jin, Chunjie Shen, Dandan Zheng, Fudong Wang, Furong Xu, GuangMing Yao, and 1 others. 2025. Ming-flash-omni: A sparse, unified architecture for multimodal perception and generation. *arXiv preprint arXiv:2510.24821*.

Anthropic. 2024. Claude 3.5 sonnet. Technical report, Anthropic.

Javad Rafiei Asl, Prajwal Panzade, Eduardo Blanco, Daniel Takabi, and Zhipeng Cai. 2024. Robustsentembed: Robust sentence embeddings using adversarial self-supervised contrastive learning. *arXiv preprint arXiv:2403.11082*.

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

Yves Bertot and Pierre Castéran. 2013. *Interactive theorem proving and program development: Coq'Art: the calculus of inductive constructions*. Springer Science & Business Media.

Jialun Cao, Yaojie Lu, Meiziniu Li, Haoyang Ma, Haokun Li, Mengda He, Cheng Wen, Le Sun, Hongyu Zhang, Shengchao Qin, and 1 others. 2025. From informal to formal–incorporating and evaluating llms on natural language requirements to verifiable formal proofs. *arXiv preprint arXiv:2501.16207*.

Luoxin Chen, Jinming Gu, Liankai Huang, Wenhao Huang, Zhicheng Jiang, Allan Jie, Xiaoran Jin, Xing Jin, Chenggang Li, Kaijing Ma, and 1 others. 2025. Seed-prover: Deep and broad reasoning for automated theorem proving. *arXiv preprint arXiv:2507.23726*.

Bofei Gao, Feifan Song, Zhe Yang, Zefan Cai, Yibo Miao, Qingxiu Dong, Lei Li, Chenghao Ma, Liang Chen, Runxin Xu, and 1 others. 2024. Omni-math: A universal olympiad level mathematic benchmark for large language models. *arXiv preprint arXiv:2410.07985*.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yixin Dai, Jiawei Sun, Haofen Wang, and Haofen Wang. 2023. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*, 2(1).

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, and 1 others. 2025. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, and 1 others. 2024. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*.

Albert Q Jiang, Wenda Li, and Mateja Jamnik. 2023. Multilingual mathematical autoformalization. *arXiv preprint arXiv:2311.03755*.

Juha Kärkkäinen, Peter Sanders, and Stefan Burkhardt. 2006. Linear work suffix array construction. *Journal of the ACM (JACM)*, 53(6):918–936.

Katherine Lee, Daphne Ippolito, Andrew Nystrom, Chiyuan Zhang, Douglas Eck, Chris Callison-Burch, and Nicholas Carlini. 2022. Deduplicating training data makes language models better. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8424–8445.

Zhaoyu Li, Jialiang Sun, Logan Murphy, Qidong Su, Zenan Li, Xian Zhang, Kaiyu Yang, and Xujie Si. 2024. A survey on deep learning for theorem proving. *arXiv preprint arXiv:2404.09939*.

Xiaohan Lin, Qingxing Cao, Yinya Huang, Haiming Wang, Jianqiao Lu, Zhengying Liu, Linqi Song, and Xiaodan Liang. 2024. Fvel: Interactive formal verification environment with large language models via theorem proving. *Advances in Neural Information Processing Systems*, 37:54932–54946.

Yong Lin, Shange Tang, Bohan Lyu, Ziran Yang, Jui-Hui Chung, Haoyu Zhao, Lai Jiang, Yihan Geng, Jiawei Ge, Jingruo Sun, and 1 others. 2025. Goedel-prover-v2: Scaling formal theorem proving with scaffolded data synthesis and self-correction. *arXiv preprint arXiv:2508.03613*.

Haoxiong Liu, Jiacheng Sun, Zhenguo Li, and Andrew C Yao. 2025a. Proofaug: Efficient neural theorem proving via fine-grained proof structure analysis. *arXiv preprint arXiv:2501.18310*.

Jiacheng Liu, Sewon Min, Luke Zettlemoyer, Yejin Choi, and Hannaneh Hajishirzi. 2024. Infini-gram: Scaling unbounded n-gram language models to a trillion tokens. *arXiv preprint arXiv:2401.17377*.

Qi Liu, Xinhao Zheng, Xudong Lu, Qinxiang Cao, and Junchi Yan. 2025b. Rethinking and improving autoformalization: towards a faithful metric and a dependency retrieval-based approach. In *The Thirteenth International Conference on Learning Representations*.

Sadegh Mahdavi, Muchen Li, Kaiwen Liu, Christos Thrampoulidis, Leonid Sigal, and Renjie Liao. 2025. Leveraging online olympiad-level math problems for llms training and contamination-resistant evaluation. *arXiv preprint arXiv:2501.14275*.

Brando Miranda, Zhanke Zhou, Allen Nie, Elyas Obbad, Leni Aniva, Kai Fronsdal, Weston Kirk, Dilara Soylu, Andrea Yu, Ying Li, and 1 others. 2025. Veribench: End-to-end formal verification benchmark for ai code generation in lean 4. In *2nd AI for Math Workshop@ ICML 2025*.

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

Logan Murphy, Kaiyu Yang, Jialiang Sun, Zhaoyu Li, Anima Anandkumar, and Xujie Si. 2024. Autoformalizing euclidean geometry. *arXiv preprint arXiv:2405.17216*.

M Saqib Nawaz, Moin Malik, Yi Li, Meng Sun, and M Lali. 2019. A survey on theorem provers in formal methods. *arXiv preprint arXiv:1912.03028*.

Tobias Nipkow, Markus Wenzel, and Lawrence C Paulson. 2002. *Isabelle/HOL: a proof assistant for higher-order logic*. Springer.

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.

Zhongyuan Peng, Yifan Yao, Kaijing Ma, Shuyue Guo, Yizhe Li, Yichi Zhang, Chenchen Zhang, Yifan Zhang, Zhouliang Yu, Luming Li, and 1 others. 2025. Criticlean: Critic-guided reinforcement learning for mathematical formalization. *arXiv preprint arXiv:2507.06181*.

Auguste Poiroux, Gail Weiss, Viktor Kunčak, and Antoine Bosselut. 2024. Improving autoformalization using type checking. *arXiv preprint arXiv:2406.07222*.

Qwen, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, and 1 others. 2025. Qwen2.5 technical report. *Preprint*, arXiv:2412.15115.

ZZ Ren, Zhihong Shao, Junxiao Song, Huajian Xin, Haocheng Wang, Wanjia Zhao, Liyue Zhang, Zhe Fu, Qihao Zhu, Dejian Yang, and 1 others. 2025. Deepseek-prover-v2: Advancing formal mathematical reasoning via reinforcement learning for subgoal decomposition. *arXiv preprint arXiv:2504.21801*.

Jasdeep Sidhu, Shubhra Mishra, Aryan Gulati, Devanshu Ladsaria, and Brando Miranda. An evaluation benchmark for autoformalization in lean4. In *The Second Tiny Papers Track at ICLR 2024*.

Qwen Team and 1 others. 2024. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2(3).

Haiming Wang, Mert Unsal, Xiaohan Lin, Mantas Baksys, Junqi Liu, Marco Dos Santos, Flood Sung, Marina Vinyes, Zhenzhe Ying, Zekai Zhu, Jianqiao Lu, Hugues de Saxcé, Bolton Bailey, Chendong Song, Chenjun Xiao, Dehao Zhang, Ebony Zhang, Frederick Pu, Han Zhu, and 21 others. 2025a. Kimina-prover preview: Towards large formal reasoning models with reinforcement learning. *arXiv preprint*.

Hanyu Wang, Ruohan Xie, Yutong Wang, Guoxiong Gao, Xintao Yu, and Bin Dong. 2025b. Aria: An agent for retrieval and iterative autoformalization via dependency graph. *arXiv preprint arXiv:2510.04520*.

Qingxiang Wang, Chad Brown, Cezary Kaliszyk, and Josef Urban. 2020. Exploration of neural machine translation in autoformalization of mathematics in mizar. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, pages 85–98.

Qingxiang Wang, Cezary Kaliszyk, and Josef Urban. 2018. First experiments with neural translation of informal to formal mathematics. In *International Conference on Intelligent Computer Mathematics*, pages 255–270. Springer.

Yuhuai Wu, Albert Qiaochu Jiang, Wenda Li, Markus Rabe, Charles Staats, Mateja Jamnik, and Christian Szegedy. 2022. Autoformalization with large language models. *Advances in neural information processing systems*, 35:32353–32368.

Huajian Xin, Daya Guo, Zhihong Shao, Zhizhou Ren, Qihao Zhu, Bo Liu, Chong Ruan, Wenda Li, and Xiaodan Liang. 2024a. Deepseek-prover: Advancing theorem proving in llms through large-scale synthetic data. *arXiv preprint arXiv:2405.14333*.

Huajian Xin, ZZ Ren, Junxiao Song, Zhihong Shao, Wanjia Zhao, Haocheng Wang, Bo Liu, Liyue Zhang, Xuan Lu, Qiushi Du, and 1 others. 2024b. Deepseek-prover-v1. 5: Harnessing proof assistant feedback for reinforcement learning and monte-carlo tree search. *arXiv preprint arXiv:2408.08152*.

Lee Xiong, Chenyan Xiong, Ye Li, Kwok-Fung Tang, Jialin Liu, Paul Bennett, Junaid Ahmed, and Arnold Overwijk. 2020. Approximate nearest neighbor negative contrastive learning for dense text retrieval. *arXiv preprint arXiv:2007.00808*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Kaiyu Yang, Aidan Swope, Alex Gu, Rahul Chalamala, Peiyang Song, Shixing Yu, Saad Godil, Ryan J Prenger, and Animashree Anandkumar. 2023. Leandojo: Theorem proving with retrieval-augmented language models. *Advances in Neural Information Processing Systems*, 36:21573–21612.

Huaiyuan Ying, Zijian Wu, Yihan Geng, Jiayu Wang, Dahua Lin, and Kai Chen. 2024a. Lean workbook: A large-scale lean problem set formalized from natural language math problems. *Advances in Neural Information Processing Systems*, 37:105848–105863.

Huaiyuan Ying, Shuo Zhang, Linyang Li, Zhejian Zhou, Yunfan Shao, Zhaoye Fei, Yichuan Ma, Jiawei Hong, Kuikun Liu, Ziyi Wang, and 1 others. 2024b. Internlm-math: Open math large language models toward verifiable reasoning. *arXiv preprint arXiv:2402.06332*.

Zhouliang Yu, Ruotian Peng, Keyi Ding, Yizhe Li, Zhongyuan Peng, Minghao Liu, Yifan Zhang, Zheng Yuan, Huajian Xin, Wenhao Huang, and 1 others. 2025. Formalmath: Benchmarking formal mathematical reasoning of large language models. *arXiv preprint arXiv:2505.02735*.

Lan Zhang, Xin Quan, and Andre Freitas. 2024. Consistent autoformalization for constructing mathematical libraries. *arXiv preprint arXiv:2410.04194*.

Ziyin Zhang, Jiahao Xu, Zhiwei He, Tian Liang, Qiuzhi Liu, Yansi Li, Linfeng Song, Zhenwen Liang, Zhuosheng Zhang, Rui Wang, and 1 others. 2025. Deeptheorem: Advancing llm reasoning for theorem proving through natural language and reinforcement learning. *arXiv preprint arXiv:2505.23754*.

Yuze Zhao, Jintao Huang, Jinghan Hu, Xingjun Wang, Yunlin Mao, Daoze Zhang, Zeyinzi Jiang, Zhikai Wu, Baole Ai, Ang Wang, and 1 others. 2025. Swift: a scalable lightweight infrastructure for fine-tuning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 29733–29735.

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

Jin Peng Zhou, Charles Staats, Wenda Li, Christian Szegedy, Kilian Q Weinberger, and Yuhuai Wu. 2024. Don't trust: Verify–grounding llm quantitative reasoning with autoformalization. *arXiv preprint arXiv:2403.18120*.

Name	RAG	Pass	Failed
Goedel-Formalizer-V2-8B	N/A	180	120
Goedel-Formalizer-V2-8B	DDR	**183**	**117**
Kimina-Autoformalizer-7B	N/A	105	195
Kimina-Autoformalizer-7B	DDR	**128**	**172**

Table 6: Comparison of different formalizers on 300 Omni-math problems.

A Additional Results

A.1 Full Figures

While Figure 3 and Figure 4 primarily presented the experimental results on the Deepseek-R1 model, this section provides a more comprehensive evaluation by integrating the DDR method with various other ICL methods in Figure 5 and Figure 6. The experimental data demonstrates that incorporating DDR brings significant performance improvements to these baseline methods in the vast majority of cases.

A.2 Out of Domain Study

To evaluate the generalization ability of DDR, we conduct autoformalization experiments on the Omni-math dataset (Gao et al., 2024), adopting the experimental setup from Lin et al. (2025). We compare DDR against the baseline methods Goedel-Formalizer-V2-8B (Lin et al., 2025) and Kimina-Autoformalizer-7B (Wang et al., 2025a). To evaluate the correctness of the generated formalizations, we employ the method proposed in Lin et al. (2025), which combines Lean4 TypeC with semantic judgments from LLMs. The pass@8 results are presented in Table 6. The results demonstrate that DDR enhances the performance of existing autoformalizers, even those not specifically designed for dependency retrieval.

B Examples Illustration of Diff01

A notable observation is that samples categorized as difficulty 0 and difficulty 1 present a counter-intuitively high formalization challenge. This is primarily because the natural language statements in this category often do not represent abstract descriptions of the problem's mathematical essence, as illustrated by examples in Figure 7. Our subsequent experimental results corroborate this finding: for **Diff01** dataset, the performance of autoformalization across all methods are worse than **Diff23** and **Diff45**, yet still better than that of **Diff89**.

Figure 7 presents several examples from the **Diff01** dataset. These problems are primarily formulated in a natural language style rather than as rigorous formal mathematical statements. Such presentation brings additional challenge in autoformalization, and renders difficulty metrics that rely on **Depend Rate** and **Depend Length** inapplicable, thereby impeding an accurate quantification of the autoformalization task difficulty on this dataset. Furthermore, because these problem descriptions lack an abstract representation of their mathematical essence, all methods evaluated in Table 4 and Table 5 exhibit a significant degradation in performance on the **Diff01** dataset.

C Experiment Settings

C.1 SAC Details

We employ precision and recall as quantitative metrics to evaluate the performance of the dependency retrieval results. A key challenge in this evaluation arises from the discrepancy between the outputs of DDR, which directly generate dependency short names, and the ground-truth labels, which are Mathlib object identifiers extracted from source code. This leads to name ambiguity, as source code often contains unqualified or partially qualified names. To accurately identify true positives for our evaluation, we introduce a suffix-based matching criterion. Specifically, we represent each identifier as a sequence of dot-separated components. A retrieved identifier is considered a match for a ground-truth label if and only if two conditions are met: (1) the string representation of one identifier is a substring of the other; and (2) when both identifiers are split by the dot character (`'.'`) into lists of their components, one list is a sublist of the other.

C.2 Metrics

To evaluate the semantic equivalence between the formal statements predicted by the model and the ground-truth formal statements, we employ two criteria: Type Checking (TypeC) and Bidirectional Extended Definitional Equality (BEq (Liu et al., 2025b). TypeC ensures the syntactic correctness of expressions, whereas BEq is utilized to determine their semantic equivalence. Building upon these criteria, we define the TypeC@8 and BEq@8 as our final evaluation metrics shown in Table 4 and Table 5. These metrics measure the percentage of problems for which at least one of the top-8 predicted statements is deemed equivalent to the

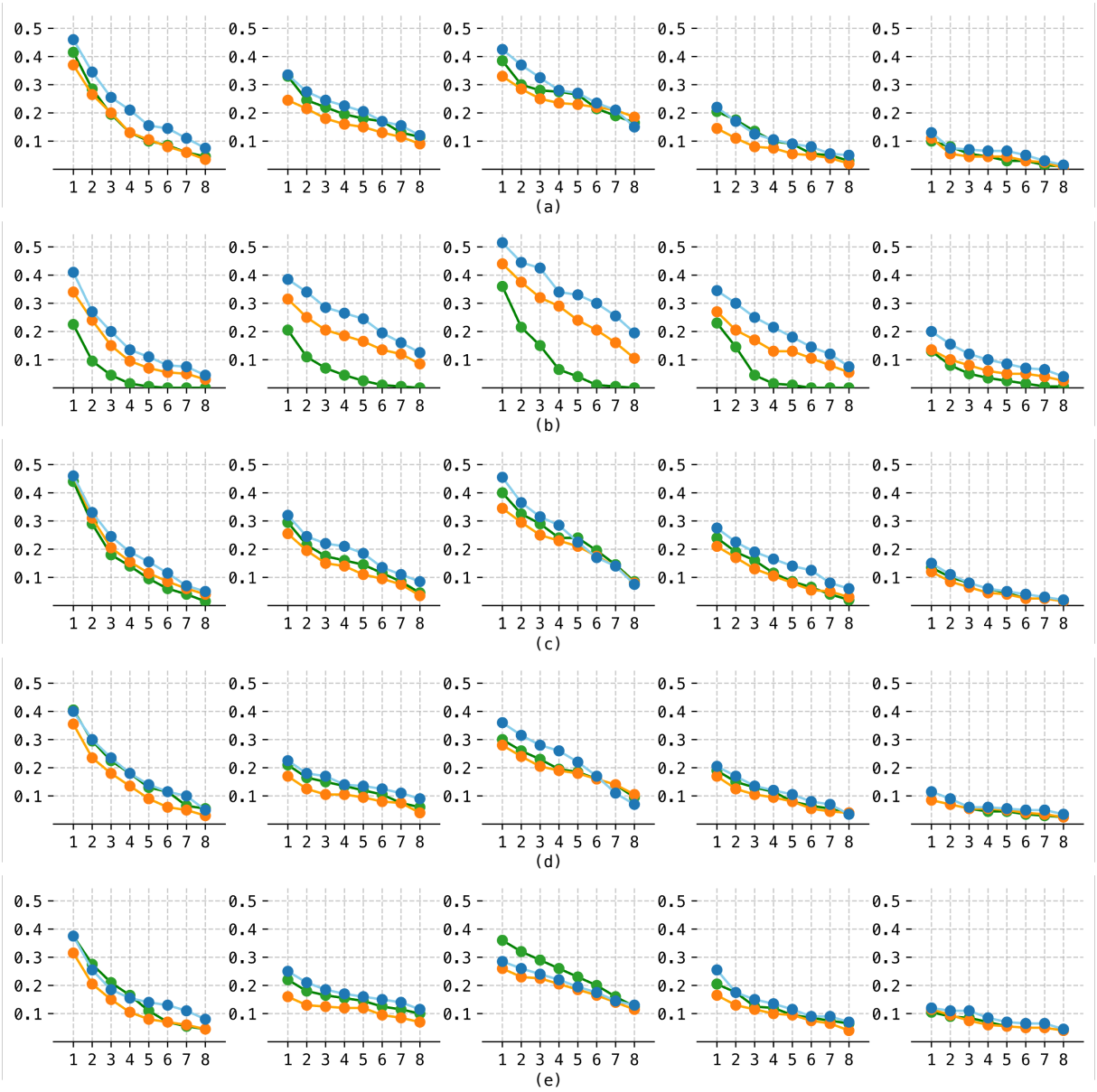

Figure 5: LLM consistency across multiple attempts. From **left** to **right** correspond to Diff01, Diff23, Diff45, Diff67, and Diff89; from **top** to **bottom** correspond to Claude-3.5-Sonnet, DeepSeek-R1, GPT-4o, Ling-flash-2.0, and Qwen-Max. Each subplot illustrates the proportion of problems (y axis) for which at least k (from 1 to 8, x axis) out of 8 attempts successfully pass the BEq verification. Dependency retrieval method: DDR (blue); ICL (orange); N/A (green).

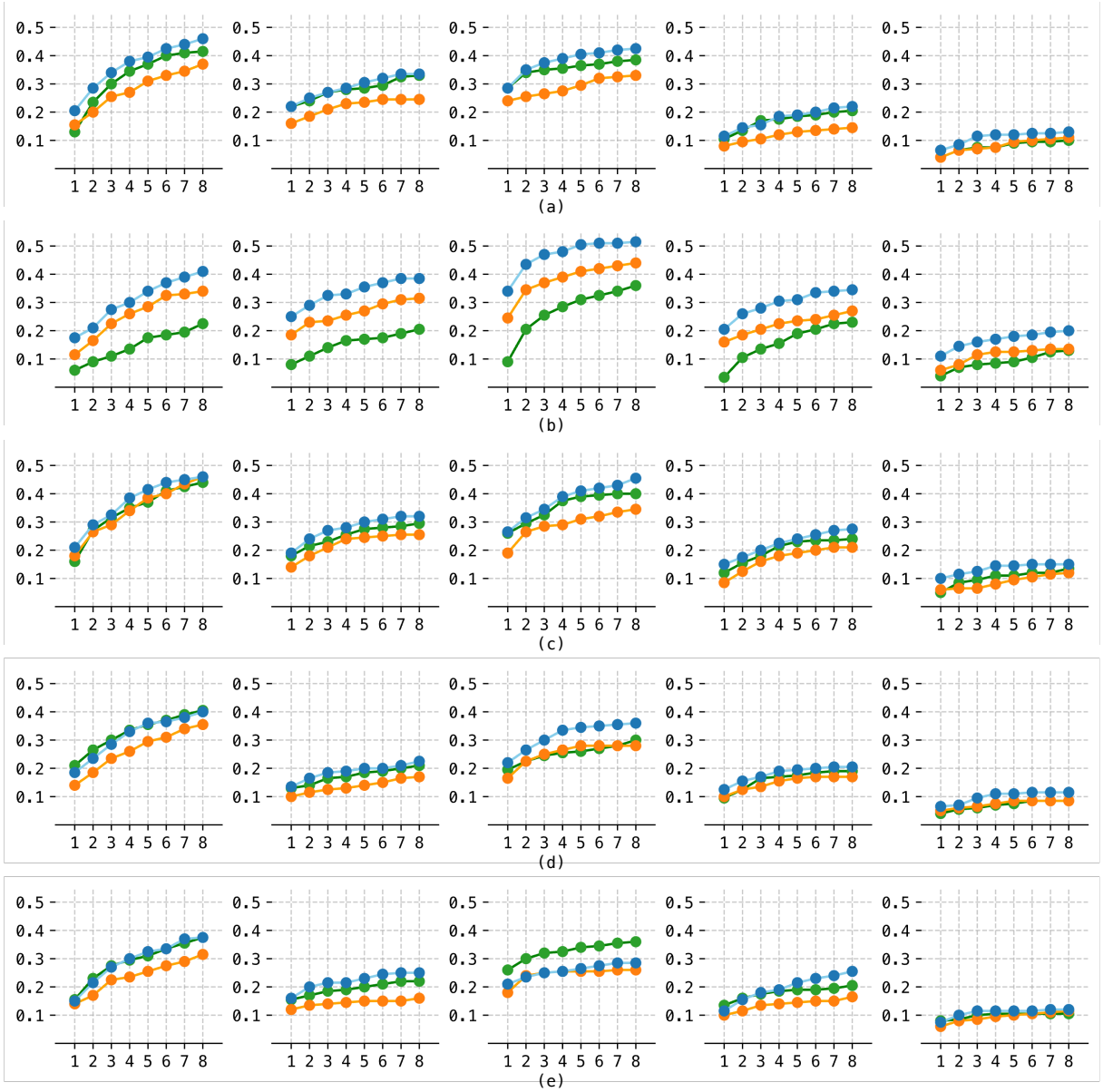

Figure 6: LLM efficiency across multiple attempts. From **left** to **right** correspond to Diff01, Diff23, Diff45, Diff67, and Diff89; from **top** to **bottom** correspond to Claude-3.5-Sonnet, DeepSeek-R1, GPT-4o, Ling-flash-2.0, and Qwen-Max. Each subplot illustrates the intermediate pass@k score (y axis) out of 8 attempts using BEq verification, k range from 1 to 8 (x axis). Dependency retrieval method: DDR (blue); ICL (orange); N/A (green).

Informal statement example 1:
Prove that the total number of wooden blocks Thomas used to make the five stacks is 55.

Formal statement example 1:
```
theorem thm_P
(a : ℕ) (h₀ : a > 0) (h₁ : a + (a + 1) + (a + 2) + (a + 3) + (a + 4) = 55) :
a = 5 := by sorry
```

Informal statement example 2:
Prove that the time it takes for Usain to run a whole lap around the school stadium is 42

Formal statement example 2:
```
theorem thm_P :
∃ t, t > 0 ∧ t = 42 := by sorry
```

Informal statement example 3:
Prove that the number of ways n tourists can distribute themselves among n cinemas so that each cinema has exactly one tourist is $n!$.

Formal statement example 3:
```
theorem thm_P
(n : ℕ) : {f : Fin n → Fin n | f.Injective}.ncard = n! := by sorry
```

Informal statement example 4:
Prove that the probability that James wins when he and his sister each spin a modified spinner with six congruent sectors numbered from 1 to 6, and James wins if the absolute difference of their numbers is 2 or less, is $\frac{2}{3}$.

Formal statement example 4:
```
theorem thm_P :
(Set.ncard {x : Fin 6 × Fin 6 | x.1 - x.2 ∈ Finset.Icc (-2) 2}) / 36
= 2 / 3 := by sorry
```

Figure 7: Examples from the FineLeanCorpus with *difficulty* levels 0 and 1.

ground-truth statement under the corresponding criterion.

C.3 Hallucination Details

It is important to note that the hallucination metric, **Hall**, is computed exclusively over the set of effective samples. We define an effective sample as an instance for which the model retrieves one or more dependencies from the library. Some autoformalization tasks are simple enough to be solved using only the given context and basic syntax, thereby not requiring any explicit library dependencies. For such cases, our model is designed to output a predefined token, "None" (or "No usage" in our implementation), to indicate that no dependency retrieval is necessary. Since these instances do not involve an actual retrieval action, the concept of hallucination, retrieving an incorrect or non-existent item, is not applicable. Therefore, they are excluded from the metric's calculation. For example, in a dataset of 200 retrieved dependency samples, if 50 are resolved with a "None" output, the effective sample size for computing the mean hallucination rate ($\mathbf{Hall_m}$) and its standard deviation ($\mathbf{Hall_{std}}$) is 150.

C.4 Baseline Details

To comprehensively evaluate our proposed method, we established several baseline models and conducted ablation studies. For the ablation studies, we employed the RAutoformalizer (RA) from Liu et al. (2025b) as a pure formalizer baseline, and RA+R@k as an autoformalizer that incorporates the top-k retrieved dependencies using the method in Liu et al. (2025b). The RA was fine-tuned based on InternLM2-Math-Base-7B (Ying et al., 2024b). Our baselines cover multiple aspects: first, we evaluated a range of state-of-the-art general LLMs via ICL, including DeepSeek-R1 (Guo et al., 2025), Ling-flash-2.0 (AI et al., 2025), Claude-3.5-Sonnet (claude-3-5-sonnet-20241022) (Anthropic, 2024), GPT-4o (gpt-4o-2024-08-06) (Hurst et al., 2024), and Qwen-Max (qwen-max-2024-09-19) (Team et al., 2024); second, we included an InternLM2-Math-Base-7B (Ying et al., 2024b) model fine-tuned on the MMA Lean subset (Jiang et al., 2023). For our proposed DDR method, we selected Qwen3-32B (Yang et al., 2025) as its base model. This choice highlights a core advantage of DDR: its flexibility in selecting a base dependency retrieval model without being restricted

to specific embedding models.

During the decoding stage for all experiments, the temperature was set to 0.7, and we generated 8 samples for each problem. To ensure a fair comparison, the dependency retrieval step for the DDR method was performed only once, and its results were reused for all subsequent Lean 4 statement formalization generations. It is noteworthy that none of the autoformalization models discussed in the main text, including the fine-tuned ones, have undergone SFT on the FineLeanCorpus (Peng et al., 2025).

For a fair comparison, we adopt an experimental setup identical to that of Liu et al. (2025b). We used the same prompt as described in Section A.7.1 of Liu et al. (2025b) for evaluation on BEq. Similarly, the prompt for the ICL method is also consistent with that in Section A.7.3 of Liu et al. (2025b). Gemini 2.5 pro is utilized as an auxiliary tool for BEq. Gemini 2.5 pro is also the LLM judge used in section A.2.

Our proposed DDR method is fine-tuned using the Swift (Zhao et al., 2025) framework with the following hyperparameters:

- Max Sequence Length: 8192
- Batch size: 1
- Gradient Accumulation: 64
- Training Devices: 8
- Train Epochs: 6
- Optimizer: AdamW with learning rate 1×10^{-4}, $\beta = (0.9, 0.95)$, weight decay 0.1, maximal gradient norm 1.0, no warmup.
- Learning Rate Scheduler: Cosine.
- LoRA: rank 32, alpha 32, dropout 0.05.

D Prompt Templates

D.1 Prompt Template for Dependency Retrieval

```
You are an expert Lean 4 ‘Mathlib‘ dependency extractor.
Your task is to analyze a given mathematical proposition and identify the key mathematical concepts that would require specific definitions, theorems, or notations from the Lean 4 ‘Mathlib‘ library.
**CRITICAL INSTRUCTIONS:**
1. Your output **MUST** be a single, valid JSON object.
2. The JSON object **MUST** have one key: "dependency".
3. The value for the "dependency" key **MUST** be a list of strings. Each string is a potential ‘Mathlib‘ dependency.
4. The dependency names can be fully qualified (e.g., ‘Nat.minFac‘) or common shortcuts (e.g., ‘ncard‘, ‘Icc‘, ‘MOD‘).
5. If there is no potential ‘Mathlib‘ library dependency, output {"dependency":["No usage"]}
6. Do **NOT** write any Lean code. Do **NOT** provide any explanations or text outside of the JSON object. Your entire response must be only the JSON.
Here are some examples of the required input and output:

**Example 1**
**Math Proposition:**
Prove that the remainder when the number N of ordered 2011 - tuples of positive integers (a_1, a_2, \ldots, a_{2011}) with 1 \le a_1, a_2, \ldots, a_{2011} \le 2011^2 such that there exists a polynomial f of degree 4019 satisfying the following conditions:
1. f(n) is an integer for every integer n;
2. 2011^2 \mid f(i) - a_i for i = 1, 2, \ldots, 2011;
3. 2011^2 \mid f(n + 2011) - f(n) for every integer n;
is divided by 1000 is equal to 211.
**Output:**
``` {
dependency:
[Polynomial,degree,Fin,eval,Set.ncard]
}
```

Example 2
Math Proposition:
Prove that the set of integers (n \ge 1) for which the smallest prime divisor of ((n!)^n + 1) is at most (n + 2015) is finite.
Output:
```

```{
dependency:[Nat.factorial,Finite,
Nat.minFac]
 }
```

Example 3
Math Proposition:
Prove that for a positive odd integer n, the arithmetic mean of the fractional parts \left{\frac{k^{2n}}{p}\right} for k = 1, \ldots, \frac{p - 1}{2} is \frac{1}{2} for infinitely many primes p satisfying p \equiv 2 \pmod{n} and p \equiv 1 \pmod{4}.
Output:
``` {
dependency:[MOD,Nat.Prime,Odd,Finset.Icc]
 }
```

Example 4
Math Proposition:
Prove that if t represents the number ten, then the value of the expression (t + t + t + t) is 40.
Output:
``` {
dependency:[No usage]
 }
```
Now, perform this task for the following proposition.
Math Proposition:
{input}
Output:
```

```

D.2 Prompt Template for Autoformalization with Dependency Retrieval

Please translate mathematical propositions into Lean 4 theorems. ‘Mathlib‘ is the MUST import.
 DO NOT try to prove the theorem, ONLY translate it.
 Use ‘sorry‘ as placeholder. Use the given potential ‘Mathlib‘ dependency items.
 Here are some examples:

Math Proposition:
Suppose f : R \rightarrow R is continuous and for every \alpha>0, \lim_{n\rightarrow \infty}f(n\alpha) = 0. Prove that \lim_{x\rightarrow \infty}f(x) = 0. The potential ‘Mathlib‘ dependency are
[Real,Continuous,Tendsto,Nat,atTop].
Lean Theorem:
```
import Mathlib
theorem exercise
(f : Real \r Real)
(hf : Continuous f \and \all \alpha > 0,
Tendsto (fun n : Nat \ma f (n * \alpha))
atTop (\MCN 0)):
(Tendsto f atTop (\MCN 0)) := sorry
```

Math Proposition:
Let G be a finite group (with a multiplicative operation), and A be a subset of G that contains more than half of G’s elements. Prove that every element of G can be expressed as the product of two elements of A. The potential ‘Mathlib‘ dependency are
[Group,Finite,Set,ncard,Nat.card,Rat].
Lean Theorem:
```
import Mathlib
open Filter Topology
theorem exercise
[Group G]
(hG : Finite G)
(A : Set G)
(hA : A.ncard > (Nat.card G : Rat)/2):
\all g : G, \ex x \in A, \ex y \in A, g
= x * y := sorry
```

Math Proposition:
Let p be a prime number greater than 5. Let f(p) denote the number of infinite sequences a_1,a_2,a_3,... such that a_n \in {1,2,...,p-1} and a_{n}a_{n+2} \equiv 1 + a_{n+1} (mod p) for all n \ge 1. Prove

````
that f(p) is congruent to 0 or 2 (mod 5).
The potential ‘Mathlib‘ dependency are
[Nat,Nat.Prime,ZMod,ncard,MOD].
Lean Theorem:
```
import Mathlib
theorem exercise
(p : Nat)
(hp : Nat.Prime p \and p > 5)
(f : Nat := {a : Nat \r (ZMod p) | \all
n : Nat, a n \neq 0 \and a n * a (n + 2)
= 1 + a (n + 1)}.ncard):
f \== 0 [MOD 5] \or f \== 2 [MOD 5] :=
sorry
```


Math Proposition:
Let p be 10, and q be 15, prove that
p+b equals 25. The potential ‘Mathlib‘
dependency are
[No usage].
Lean Theorem:
  ```
import Mathlib
theorem exercise
(p : Nat)
(q : Nat)
(hp : p = 10)
(hq : q = 15):
p + q = 25 := sorry
```


Please translate the following
proposition:
Math Proposition:
{input}
The potential ‘Mathlib‘ dependency are
{retrieved-dependency}
Lean Theorem: ```


```
````

D.3 Other Templates

Other ICL templates employed in this paper, including those for autoformalization without dependency retrieval and BEq verification, adhere to the corresponding settings in Liu et al. (2025b).
````