Beyond Relevance: Counterfactual Utility Filtering for Retrieval-Augmented Autoformalization

Anonymous ACL submission

Abstract

Autoformalization bridges large language models and trustworthy reasoning systems by translating informal mathematics into proof-assistant-checkable statements. Although relevance-based retrieval can improve autoformalization, it can also be counterproductive: semantically related knowledge may fail to support verifier-accepted formalization. This reveals a key mismatch between conventional RAG and autoformalization: what matters is whether retrieved knowledge helps instantiate the correct formal anchors—proof-assistant-level objects, type constraints, and library declarations required for verifier-accepted formalization. We propose a counterfactual-driven, verifier-guided framework for *utility-aware retrieval-augmented autoformalization*. Our method controls when knowledge should be used and applies *counterfactual utility filtering* to identify which retrieved knowledge truly supports formal anchoring. Concretely, it deletes concept clusters to synthesize counterfactual subproblems and uses Lean compilation and semantic alignment feedback to select high-utility knowledge. Experiments on FormL4, PutnamAxiom, and Gaokao-Formal show that our method solves hard cases left unresolved by relevance-based RAG. In **FAR@10**, counterfactual utility filtering achieves a maximum relative gain of 84.71% over the +RAG stage across different datasets and autoformalizers. These results suggest that retrieval-augmented autoformalization benefits not from retrieving more knowledge, but from selecting knowledge that improves formal anchoring.

1 Introduction

Autoformalization translates informal mathematical statements into proof-assistant-checkable formal language, providing an important interface between large language models and trustworthy reasoning systems. Unlike standard mathematical question answering, autoformalization requires the model to generate formal statements that can be compiled, type-checked, and further consumed by proof assistants such as Lean (de Moura et al., 2015), Isabelle/HOL (Nipkow et al., 2002), or Coq (Bertot and Castéran, 2004). The key difficulty is not merely producing Lean-like syntax, but instantiating the proof-assistant-level objects, type constraints, and library declarations required by the verifier. We refer to these structures as *formal anchors*: the objects, types, and declarations that anchor an informal statement to a verifier-accepted formal representation.

Retrieval-augmented generation (RAG) is therefore a natural way to improve autoformalization, since retrieved definitions, theorems, lemmas, type structures, and library declarations can provide useful formalization cues. Recent work also shows that retrieval augmentation can improve formalization success rates (Zhang et al., 2024; Lu et al., 2026). However, our central observation is that relevance is not equivalent to *knowledge utility for formal anchoring*. A retrieved snippet may be semantically related to the problem, yet still fail to help the model instantiate the correct formal anchors, or even distract it toward incompatible Lean objects, type constraints, or library constructions. Thus, the key question for retrieval-augmented autoformalization is not simply whether retrieved knowledge is relevant, but whether it supports verifier-accepted formal anchoring.

Figure 1 illustrates this mismatch. The original problem asks whether the mapping induced by a function on measurable spaces is monotone, and direct autoformalization fails. Relevance-based retrieval returns several related concepts, including preimages, image/preimage properties, set monotonicity, and sigma-algebra-related concepts. Although these concepts are all semantically related, relevance alone does not distinguish which concepts are merely surface-related and which ones provide the formal anchors needed to resolve the

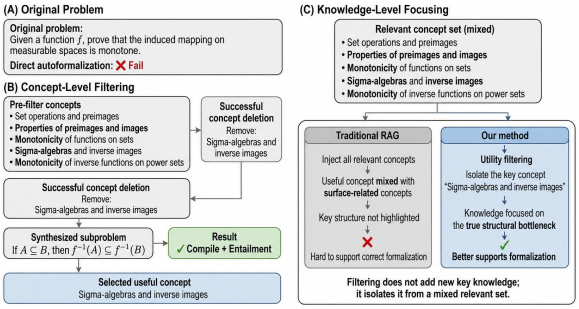

Figure 1: A case illustrating why relevance-based knowledge injection is insufficient for autoformalization.

failure. This shows that useful knowledge may already exist in the retrieved candidate pool, but it is mixed with weakly useful or distracting context.

To estimate knowledge utility, we use counterfactual deletion as a verifier-guided diagnostic signal. The intuition is that if deleting or simplifying a concept cluster yields a counterfactual subproblem that becomes compilable and semantically aligned, then the removed cluster is likely associated with the formalization bottleneck of the original problem. In Figure 1, deleting the cluster involving *sigma-algebras and inverse images* produces a verifier-accepted subproblem, indicating that knowledge related to this cluster is more useful for recovering the missing formal anchors. Counterfactual deletion therefore turns verifier feedback into a utility signal for selecting knowledge that supports formal anchoring.

Motivated by this observation, we propose *utility-aware retrieval-augmented autoformalization*, a verifier-guided framework that controls both when knowledge should be used and which knowledge is useful for formal anchoring. At the instance level, a route decision model predicts whether a problem should follow the DIRECT route or the KNOWLEDGE route, reducing unnecessary retrieval for problems that are already directly formalizable. At the concept level, *counterfactual utility filtering* identifies utility-relevant concept clusters by synthesizing counterfactual subproblems through concept deletion and evaluating them with Lean compilation and semantic alignment. The resulting utility signal is used to construct a focused knowledge context, rather than injecting all superficially related retrieval results.

In summary, this paper argues that the retrieval bottleneck in autoformalization is not merely a knowledge-coverage problem, but a knowledge-utility problem. Autoformalization benefits not from retrieving more knowledge, but from selecting knowledge that supports correct *formal anchoring*. We make the following contributions:

- We introduce *knowledge utility for formal anchoring* as the central criterion for retrieval-augmented autoformalization, shifting retrieval from surface relevance estimation to support for verifier-accepted formal anchors.
- We propose *counterfactual utility filtering*, a verifier-guided method that identifies high-utility mathematical concept clusters through concept deletion and verifier-accepted subproblem analysis.
- We develop a framework for *utility-aware retrieval-augmented autoformalization* that decides when knowledge is needed and filters retrieved knowledge by utility. Experiments on FormL4 (Lu et al., 2024), PutnamAxiom (Azim et al., 2023), and Gaokao-Formal (Xuejun et al., 2025) show that our method recovers cases left unresolved by relevance-based RAG, achieving a maximum relative gain rate of 84.71% in FAR@10 over the +RAG stage.

2 Method

We propose a utility-aware retrieval-augmented autoformalization framework driven by verifier feedback. As shown in Figure 2, the framework first predicts whether retrieval is needed, then applies counterfactual utility filtering to hard cases unresolved by relevance-based RAG, and finally constructs a utility-aware context for verifier-gated generation. All route labels, counterfactual interventions, and accepted outputs are defined using the same criterion based on Lean compilation and semantic alignment.

2.1 Task Formulation and Verifier-Guided Acceptance Criterion

Given an informal mathematical problem q, an autoformalization model F_θ is expected to generate Lean formal statements. We denote by κ the external knowledge context provided to the model. When no external knowledge is used, we write $\kappa = \varnothing$. Under the pass@m setting, the model produces a set of candidate formalizations:

$$\mathcal{Z}_m(q;\kappa) = \{z_1, \ldots, z_m\} \sim F_\theta(q;\kappa), \quad (1)$$

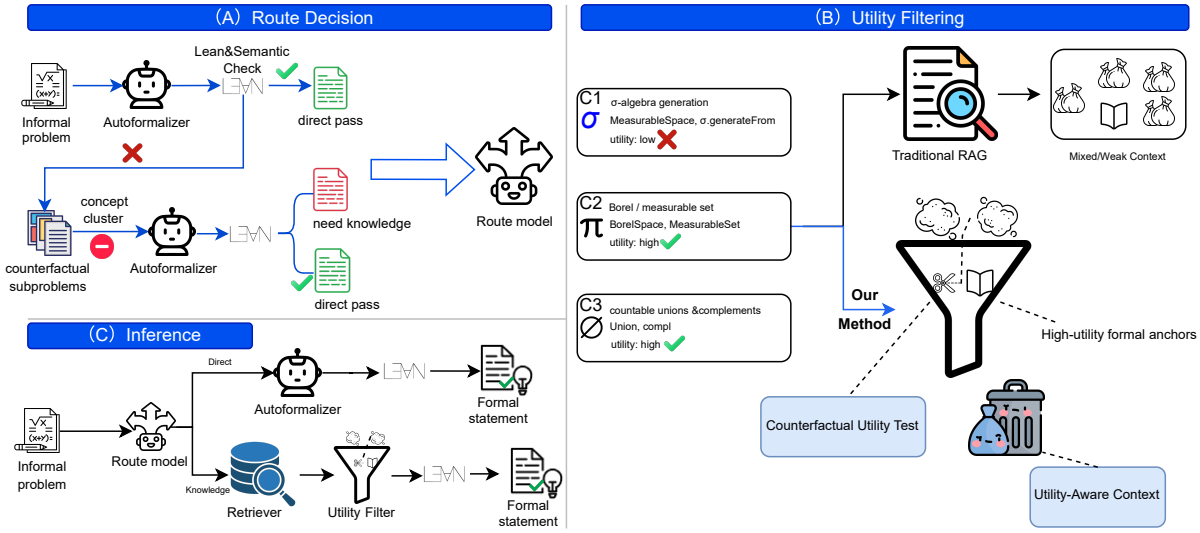

Figure 2: Overview of our utility-aware retrieval-augmented autoformalization framework. (A) The route supervision module learns when external knowledge should be invoked from verifier-derived labels and counterfactual boundary augmentation. (B) The counterfactual utility filtering module identifies utility-selected concept clusters and constructs a high-utility knowledge context. (C) The inference pipeline combines route decision and utility filtering through a verifier-gated cascade.

where each z_i is a Lean formal statement candidate.

We evaluate autoformalization success through a verifier-guided criterion. A generated statement is accepted only if it is both syntactically and type-theoretically valid in Lean, and semantically aligned with the original informal problem. Accordingly, we define a verifier $\mathcal{V}$ that combines Lean compilation and semantic alignment:

$$\mathcal{V}(q, z) = \begin{cases} 1, & \begin{array}{l}\mathrm{LeanCompile}(z) = 1, \\ \mathrm{SemAlign}(q, z) = 1,\end{array} \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

Here, $\mathrm{LeanCompile}(z) = 1$ indicates that the generated formal statement z successfully passes Lean compilation, and $\mathrm{SemAlign}(q, z) = 1$ indicates that z is semantically aligned with the original informal problem q.

Based on this verifier, we define the pass@m acceptance function under knowledge context κ:

$$\mathrm{Accept}_m(q; \kappa) = \begin{cases} 1, & \begin{array}{l}\exists z \in \mathcal{Z}_m(q; \kappa) \\ \text{s.t. } \mathcal{V}(q, z) = 1,\end{array} \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

Thus, the direct route corresponds to $\mathrm{Accept}_m(q; \varnothing)$, while the knowledge route corresponds to $\mathrm{Accept}_m(q; \kappa)$ for some retrieved knowledge context κ. All subsequent route labels, counterfactual data construction, and knowledge utility estimation are grounded in this verifier-guided acceptance criterion. This ensures that the supervision signals used in our framework are aligned with the final goal of autoformalization, rather than with surface-level textual similarity or retrieval relevance alone.

2.2 Route Decision Model

Given the verifier-guided acceptance criterion, we train a route decision model to assign each informal problem to either the direct route or the knowledge route. Figure 3 illustrates the three-stage training pipeline: seed route labeling, counterfactual boundary augmentation, and route decision model training.

Seed route labeling. As shown in Figure 3(A), we first construct coarse route supervision from direct autoformalization. For each informal problem q, we run the autoformalizer without external knowledge and evaluate the generated Lean candidates with the verifier defined in Section 2.1. If the direct route is accepted, the problem is labeled as `Direct`; otherwise, it is labeled as `Knowledge`:

$$y_{\mathrm{seed}}(q) = \begin{cases} \mathrm{Direct}, & \mathrm{Accept}_m(q; \emptyset) = 1, \\ \mathrm{Knowledge}, & \mathrm{Accept}_m(q; \emptyset) = 0. \end{cases} \tag{4}$$

The `Knowledge` label should be interpreted as a route label rather than a success label. It does not mean that knowledge augmentation is guaranteed to solve the problem. Instead, it only indicates that,

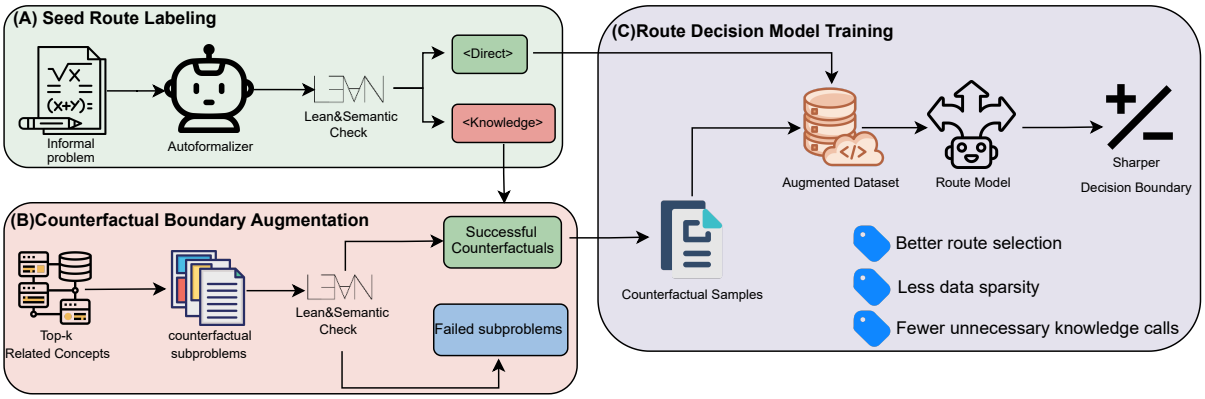

Figure 3: Training the route decision model. (A) Seed route labels are obtained from direct autoformalization and verifier feedback. (B) For direct-failure problems, concept-conditioned counterfactual subproblems are synthesized and verified; successful counterfactuals are collected as counterfactual route samples. (C) The augmented route dataset is used to train a route decision model with a sharper Direct-vs-Knowledge decision boundary.

under the current autoformalizer, the direct route is insufficient. These seed labels provide the initial supervision for learning the target autoformalizer's Direct-vs-Knowledge boundary.

Counterfactual boundary augmentation. Seed labels alone are coarse. Direct-success and direct-failure problems may differ in superficial properties such as length, domain, notation density, or symbolic complexity. A route model trained only on such labels may therefore learn shallow difficulty cues rather than the actual boundary of when the target autoformalizer requires external knowledge.

To obtain more informative supervision, we augment direct-failure problems with counterfactual samples, as shown in Figure 3(B). For each seed `Knowledge` problem, we extract k related mathematical concepts, with $k = 5$ in our experiments, and synthesize concept-conditioned counterfactual subproblems by removing or simplifying selected concept clusters. Each counterfactual subproblem is then evaluated through the direct route using the same verifier.

If a counterfactual subproblem is accepted by the verifier, it is collected as a successful counterfactual sample and labeled as `Direct`. Together with its original direct-failure problem labeled as `Knowledge`, it forms a local counterfactual pair around the Direct-vs-Knowledge boundary. These counterfactual pairs provide local boundary supervision around direct-failure cases.

Route decision model training. As shown in Figure 3(C), the final training set combines the original seed labels and the verifier-accepted counterfactual samples:

$$\mathcal{D}_{\text{route}} = \mathcal{D}_{\text{seed}} \cup \mathcal{D}_{\text{cf}}, \tag{5}$$

where $\mathcal{D}_{\text{seed}}$ contains the seed route labels obtained from direct autoformalization, and $\mathcal{D}_{\text{cf}}$ contains verifier-accepted counterfactual subproblems generated from direct-failure problems.

We train a binary route decision model D_ϕ on $\mathcal{D}_{\text{route}}$ to predict whether an input problem should follow the direct route or the knowledge route. After training, D_ϕ acts as an instance-level knowledge controller at inference time. It sends problems that are likely to be directly formalizable to the direct route, while routing knowledge-requiring problems to the knowledge route. By learning from successful counterfactual samples around failed problems, the route model obtains a sharper Direct-vs-Knowledge boundary, reduces unnecessary knowledge invocation, and mitigates the scarcity of route-level supervision.

2.3 Counterfactual Utility Filtering

The route decision model decides whether external knowledge should be invoked, but not which knowledge should be injected. Standard RAG injects top-ranked snippets by relevance, yet relevance does not necessarily imply utility for formal anchoring. We therefore estimate knowledge utility through counterfactual verification.

Figure 4 illustrates the proposed utility filtering process. Given a hard problem and a knowledge library, we first retrieve a broad candidate pool containing different types of formalization-related knowledge, such as theorems, lemmas, explanations, and code snippets. Instead of injecting this

pool directly, we perform counterfactual interventions over concept clusters and use verifier feedback to separate high-utility knowledge from low-utility knowledge. The selected knowledge is then used to construct a focused utility-aware context for final autoformalization.

Let $\mathcal{K}_{\text{rel}}(q)$ denote the relevance-based knowledge context retrieved for problem q. We first define the direct-route and relevance-based-route acceptance indicators as:

$$\begin{aligned} a_{\text{dir}}(q) &= \text{Accept}_m(q; \emptyset), \\ a_{\text{rel}}(q) &= \text{Accept}_m(q; \mathcal{K}_{\text{rel}}(q)). \end{aligned} \tag{6}$$

Counterfactual utility filtering is activated for hard cases where both direct formalization and relevance-based RAG fail. For these cases, the retrieved candidate pool may already contain useful knowledge, but the useful part is often mixed with weakly useful or distracting context. The goal is therefore not to retrieve more knowledge, but to identify which concepts in the candidate pool are truly associated with the formalization bottleneck.

We conduct counterfactual intervention over concept clusters rather than isolated concepts, because Lean-relevant structures often couple definitions, type constraints, notations, and library constructions. Cluster-level deletion provides a stronger diagnostic intervention for exposing whether a coupled structure constitutes the bottleneck of the original failure.

For each concept cluster g_t, we construct a counterfactual subproblem by deleting or simplifying the mathematical structure associated with that cluster:

$$c_t = \mathcal{T}_{\text{cf}}(q, g_t), \tag{7}$$

where $\mathcal{T}_{\text{cf}}$ denotes the counterfactual rewriting operation conditioned on the original problem q and the concept cluster g_t. We then evaluate whether this counterfactual subproblem can be formalized through the direct route:

$$u_t = \text{Accept}_m(c_t; \emptyset). \tag{8}$$

If $u_t = 1$, the counterfactual subproblem becomes directly formalizable after the structure associated with g_t is removed or simplified. This indicates that g_t corresponds to a structural bottleneck in the original problem. Consequently, knowledge related to g_t is likely to provide useful formal anchors for recovering the missing Lean objects, type constraints, or library constructions. We define the utility-selected concept clusters as:

$$\mathcal{G}_{\text{util}}(q) = \{g_t \in \mathcal{G}_{\text{rel}}(q) \mid u_t = 1\}. \tag{9}$$

The counterfactual subproblems are not treated as solutions to the original problem. Instead, they serve as diagnostic probes. Their verifier outcomes identify which concept clusters are most associated with the observed formalization failure. In this way, knowledge filtering is shifted from relevance estimation to verifier-guided utility estimation.

2.4 Utility-Aware Knowledge Construction

After obtaining the utility-selected concept clusters, we construct a compact knowledge context focused on these clusters. For each $g_t \in \mathcal{G}_{\text{util}}(q)$, we retrieve formal knowledge that can help anchor the corresponding mathematical structure in Lean. The resulting utility-aware knowledge context is defined as:

$$\mathcal{K}_{\text{util}}(q) = \text{Retrieve}\left(q, \mathcal{G}_{\text{util}}(q)\right). \tag{10}$$

The retrieved context may include theorems, lemmas, Lean declarations, type signatures, formal definitions, concise natural-language explanations, and code snippets. The goal is not to maximize the amount of retrieved knowledge, but to provide the autoformalizer with the formal anchors most likely to address the detected bottleneck. Compared with injecting all relevance-based snippets, utility-aware knowledge construction produces a more focused context and reduces the risk that weakly useful concepts distract the generation process.

If no utility-selected concept cluster is identified, we fall back to the relevance-based context $\mathcal{K}_{\text{rel}}(q)$. This fallback keeps the system robust while preserving the main principle of our method: when counterfactual evidence is available, knowledge injection is guided by utility rather than relevance alone.

2.5 Decision-Guided and Verifier-Gated Inference

Figure 2(C) summarizes the inference-time cascade. Given a new problem q, the route decision model predicts whether it should follow the DIRECT route or the KNOWLEDGE route:

$$\hat{y} = D_\phi(q), \quad \hat{y} \in \{\text{Direct}, \text{Knowledge}\}. \tag{11}$$

If $\hat{y} = \text{Direct}$, the system first attempts direct autoformalization without external knowledge. If

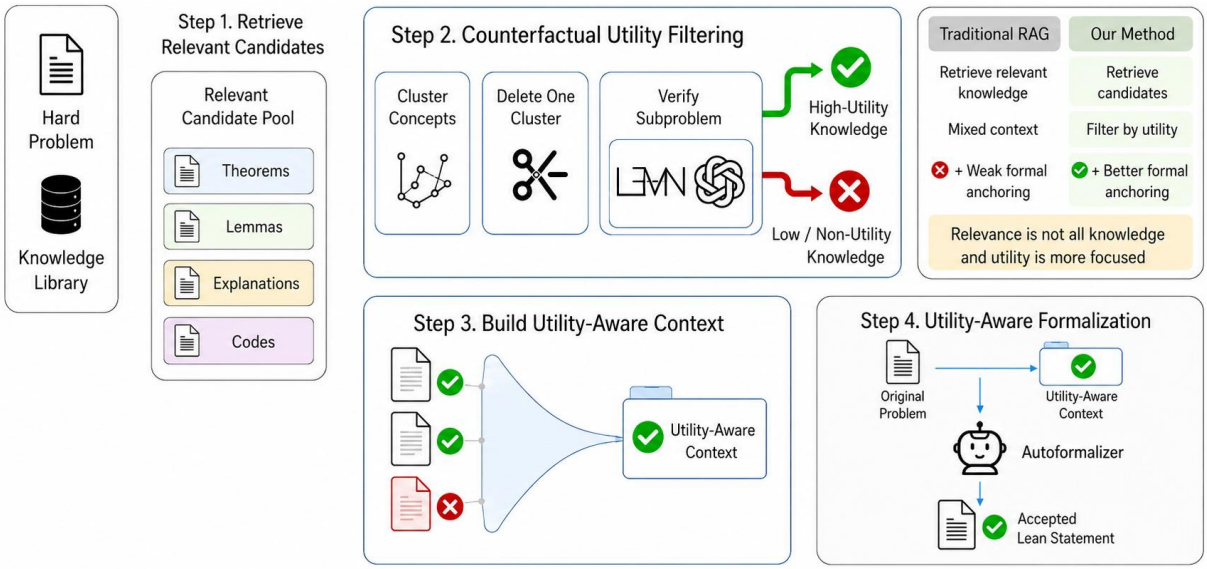

Figure 4: Counterfactual utility filtering for formal anchoring. Relevance-based retrieval produces a mixed candidate pool, while verifier-guided counterfactual filtering selects high-utility knowledge and builds a focused context for autoformalization.

one of the generated candidates is accepted by the verifier, the accepted formal statement is returned. If the Direct route fails, the system falls back to the knowledge route, which reduces the risk caused by route misclassification.

If $\hat{y} = \text{Knowledge}$, the system first performs conventional relevance-based retrieval and attempts the Knowledge route using the relevance-based context $\mathcal{K}_{\text{rel}}(q)$. If the verifier accepts a generated candidate, the system returns the accepted formal statement. Only when relevance-based knowledge injection still fails do we activate counterfactual utility filtering and construct $\mathcal{K}_{\text{util}}(q)$ from the utility-selected concept clusters. The autoformalizer then regenerates candidates under the utility-aware knowledge context.

This design follows a low-cost to high-cost cascade. Easy problems are solved directly, avoiding unnecessary retrieval. Problems that require external information are first handled with relevance-based RAG. Only difficult cases that fail under both the direct route and the relevance-based RAG stage trigger counterfactual utility filtering. Since every output must satisfy the same verifier-guided acceptance criterion, the system prevents unverified generations from being accepted. Overall, the framework addresses two central challenges in retrieval-augmented autoformalization: deciding when knowledge is needed and selecting what knowledge is actually useful for formalization success.

3 Experiments

3.1 Experimental Protocol

Following the evaluation protocol of retrieval-augmented autoformalization (Lu et al., 2026), we report two metrics: **Compilation Pass Rate@10** (CPR@10) and **Formalization Accuracy Rate@10** (FAR@10). CPR@10 measures whether at least one of the top-10 generated Lean candidates passes compilation, while FAR@10 further requires semantic alignment with the original informal problem.

We compare three inference strategies. **Direct** performs autoformalization without external knowledge. **+RAG** uses the retrieval pipeline of (Lu et al., 2026), a specialized concept-driven retrieval framework for autoformalization involving hybrid retrieval, contextual query augmentation, and reranking after Direct. **+Ours** applies our counterfactual utility filtering on top of Direct+RAG. Since our framework follows a cascade design, the reported scores are cumulative: +RAG denotes the performance after adding relevance-based retrieval to Direct, while +Ours denotes the performance after applying counterfactual utility filtering on top of Direct+RAG. Following prior retrieval-augmented autoformalization work, we also report relative gain (RG) over the preceding stage, which measures how much each stage improves upon the previous one.

Dataset	Model	Direct	+RAG	RG_{RAG}	+Ours	RG_{CF}
FormL4	DS-V3	28.10	51.06	81.71	57.14	11.91
	Gemini	24.22	45.72	88.77	54.99	20.28
	GPT-4o	15.80	24.13	52.72	34.13	41.44
	Kimina	86.08	91.75	6.59	97.52	6.29
PutnamAxiom	DS-V3	26.05	35.75	37.24	44.85	25.45
	Gemini	33.14	42.14	27.16	74.13	75.91
	GPT-4o	23.37	29.07	24.39	30.07	3.44
	Kimina	93.30	95.98	2.87	98.08	2.19
	Globel	81.61	86.02	8.05	89.08	5.56
Gaokao-Formal	DS-V3	54.40	73.50	35.11	83.50	13.61
	Gemini	16.16	48.88	202.48	58.46	19.60
	GPT-4o	15.10	42.10	178.81	46.10	9.50
	Kimina	98.38	98.78	0.41	99.59	1.02
	Globel	89.90	91.72	2.02	99.40	8.37

Table 1: CPR@10 results. Direct, +RAG, and +Ours are cumulative scores. RG_{RAG} denotes the relative gain of +RAG over Direct, and RG_{CF} denotes the relative gain of +Ours over +RAG. All values are percentages.

Dataset	Model	Direct	+RAG	RG_{RAG}	+Ours	RG_{CF}
FormL4	DS-V3	26.90	48.95	81.97	54.72	11.79
	Gemini	20.72	37.36	80.31	45.09	20.69
	GPT-4o	13.60	21.12	55.29	30.52	44.51
	Kimina	62.16	66.38	6.79	70.81	6.67
PutnamAxiom	DS-V3	15.32	21.82	42.43	29.22	33.91
	Gemini	18.00	23.74	31.89	43.85	84.71
	GPT-4o	11.11	14.17	27.54	15.17	7.06
	Kimina	44.83	50.00	11.53	58.43	16.86
	Globel	43.24	49.14	13.65	54.09	10.07
Gaokao-Formal	DS-V3	51.10	63.22	23.72	72.62	14.87
	Gemini	13.93	36.96	165.33	45.00	21.75
	GPT-4o	8.40	29.41	250.12	33.41	13.60
	Kimina	78.18	80.80	3.35	88.22	9.18
	Globel	85.05	87.06	2.36	90.91	4.41

Table 2: FAR@10 results. Direct, +RAG, and +Ours are cumulative scores. FAR@10 requires both Lean compilation and semantic alignment. RG_{RAG} denotes the relative gain of +RAG over Direct, and RG_{CF} denotes the relative gain of +Ours over +RAG. All values are percentages.

3.2 Autoformalization Performance

Tables 1 and 2 report the main results. Relevance-based RAG improves over Direct in most settings, confirming that external mathematical knowledge can recover many direct formalization failures. However, counterfactual utility filtering further improves upon the Direct+RAG stage across almost all settings, showing that the remaining failures are not merely caused by the absence of relevant knowledge. Rather, useful formal anchors are often mixed with surface-related or weakly useful context, and verifier-guided filtering makes the effective part of retrieved knowledge more salient.

The gains are especially clear in FAR@10, where semantic alignment is required in addition to Lean compilation. For example, on PutnamAxiom with Gemini-2.5, +Ours improves FAR@10 from 23.74 to 43.85 after the RAG stage, yielding an 84.71% relative gain. The same pattern also appears for stronger autoformalizers: on PutnamAxiom with Kimina, FAR@10 improves from 50.00 to 58.43, and on Gaokao-Formal with Kimina, it improves from 80.80 to 88.22. For Globel, the gains are more moderate but consistent: on PutnamAxiom, CPR@10 improves from 81.61 to 86.02 with +RAG and further to 89.08 with +Ours, while FAR@10 improves from 43.24 to 49.14 and then to 54.09. These results suggest that utility filtering is beneficial not only for weak base models, but also for stronger autoformalizers whose remaining errors often come from subtle formal-anchor mismatches.

In contrast, CPR@10 is close to saturation in some settings, such as Gaokao-Formal with Kimina and Globel. This indicates that once compilation becomes relatively easy, the main bottleneck shifts from producing syntactically valid Lean statements to selecting knowledge that supports semantically aligned formalization. Overall, the results support treating retrieval-augmented autoformalization as verifier-guided knowledge-utility control rather than default retrieve-and-inject generation.

3.3 Route Decision Model

We further evaluate the route decision model on the FormL4 test set. The model predicts whether a problem should follow the direct route or enter the knowledge route. The route labels are constructed from the real outcomes of the Kimina autoformalizer: if Kimina can directly formalize a problem without external knowledge, the problem is labeled as Direct; otherwise, it is labeled as Knowledge. Therefore, the router does not learn abstract problem difficulty. Instead, it learns the direct-vs-knowledge boundary of the target autoformalizer under the current autoformalization setting.

The model is trained using FormL4 (Lu et al., 2024) training data as the seed set. We then apply counterfactual boundary augmentation to expand the route training set to **2.67**$\times$ its original size. This augmentation is not intended to simply increase the number of training examples. Rather, it constructs local counterfactual variants around direct-failure problems, helping the router explore the Direct-vs-Knowledge boundary of the target autoformal-

Model	Recall	F1	Accuracy
qwen3-4B	0.9239	0.8113	0.7031
qwen2.5-0.5B	0.9194	0.8026	0.6876
qwen2.5-0.5B-base	0.1493	0.2121	0.2340
qwen3-4B-base	0.1791	0.2503	0.2588

Table 3: Route decision model performance on the FormL4 test set. Labels are constructed from the real Direct/Knowledge outcomes of the Kimina formalizer. Base models are supervised fine-tuned without counterfactual boundary augmentation.

izer. The base models in Table 3 are supervised fine-tuned models trained without counterfactual boundary augmentation. All route models use the same input format and training objective; the only difference is whether counterfactual boundary augmentation is applied during route data construction.

Table 3 shows that counterfactual boundary augmentation substantially improves route prediction. The augmented qwen3-4B model reaches 0.8113 F1 and 0.7031 accuracy, while its base counterpart obtains only 0.2503 F1 and 0.2588 accuracy. A similar pattern appears for qwen2.5-0.5B, where F1 increases from 0.2121 to 0.8026. These results indicate that route selection is not a capability that can be reliably obtained from ordinary supervised fine-tuning alone; it benefits from verifier-derived boundary supervision.

This result also explains why counterfactual boundary augmentation is useful. Direct and Knowledge examples are not reliably separable by surface difficulty. A problem may look complex but still be directly formalizable by the target autoformalizer, while another may appear simple but require a specific Lean definition, type constraint, or library construction. By constructing counterfactual variants around direct-failure problems, the training process provides local boundary examples that help the router learn when external knowledge should be invoked.

The route decision model also improves efficiency and robustness. Compared with an all-RAG strategy, where every instance receives retrieved context, our router invokes retrieval only for samples predicted as Knowledge. Thus, the route model addresses the "when to retrieve" problem, while counterfactual utility filtering addresses the "what to retrieve" problem.

4 Related Work

Retrieval augmentation and retrieval control. Retrieval-augmented generation has evolved from simply adding external documents to controlling when and how retrieved context should be used. Prior work studies adaptive retrieval, retrieval critique, query augmentation, and post-retrieval ranking (Shi et al., 2024a; Asai et al., 2024; Jeong et al., 2024; Shi et al., 2024b; Li et al., 2025; Yu et al., 2024). Other studies show that retrieved context can be noisy, irrelevant, or conflicting with model knowledge (Yoran et al., 2024; Wang et al., 2025). Our work follows this line of retrieval control, but targets a different notion of usefulness: in autoformalization, retrieved knowledge is useful only if it helps construct verifier-accepted formal anchors, including formal objects, type constraints, and library references.

Autoformalization and theorem proving. Recent autoformalization work improves informal-to-formal translation through domain structure, multilingual data, mathematical-library construction, and evaluation on realistic definitions (Murphy et al., 2024; Jiang et al., 2024; Zhang et al., 2024; Poiroux et al., 2025; Zhang et al., 2025). Lean-centered theorem proving systems further show the value of large-scale formal data, specialized training, and proof-assistant feedback (Ying et al., 2024; Wang et al., 2024; Xin et al., 2024; Lin et al., 2025; Xin et al., 2025). These works mainly improve the formalizer or prover itself, while our work studies how external knowledge should be controlled for retrieval-augmented autoformalization.

5 Conclusion

We present utility-aware retrieval-augmented autoformalization, which shifts retrieval from surface relevance to knowledge utility for formal anchoring. The framework decides when external knowledge is needed and uses counterfactual utility filtering to identify retrieved knowledge that supports verifier-accepted formalization. Experiments across three benchmarks show that our method recovers cases left unresolved by relevance-based RAG, demonstrating that effective retrieval for autoformalization depends not on retrieving more knowledge, but on selecting knowledge that improves formal anchoring.

6 Limitations

This work focuses on utility-aware retrieval-augmented autoformalization, and its limitations mainly arise from the verifier-guided and counterfactual nature of the framework. First, the quality of route labels and utility estimates depends on the acceptance criterion used throughout the pipeline. Lean compilation provides a reliable signal for syntactic and type-theoretic validity, but semantic alignment is still an approximate judgment. As a result, errors in semantic alignment may affect seed route labeling, counterfactual sample selection, and the identification of utility-selected concept clusters.

Second, counterfactual utility filtering relies on the construction of meaningful counterfactual subproblems. Although deleting or simplifying concept clusters provides a useful diagnostic signal, the resulting subproblem may not always isolate a single formalization bottleneck. Mathematical concepts are often tightly coupled with definitions, type constraints, notations, and library declarations, so cluster-level deletion may either remove too much structure or fail to fully remove the relevant dependency. Therefore, the utility signal should be interpreted as verifier-derived evidence that a concept cluster is associated with a bottleneck, rather than as a complete causal explanation.

Third, the framework introduces additional computational cost for hard cases. The inference cascade avoids applying counterfactual utility filtering to all instances, but when both direct formalization and relevance-based RAG fail, the system needs to synthesize and verify multiple counterfactual subproblems before constructing the utility-aware context. This makes the method more suitable for cases where relevance-based retrieval is insufficient, rather than as a uniformly cheap replacement for standard RAG.

Finally, the route decision model is tied to the behavior of the target autoformalizer. A problem labeled as DIRECT or KNOWLEDGE reflects whether the current autoformalizer succeeds without external knowledge, rather than an intrinsic property of the mathematical problem itself. When the underlying autoformalizer, verifier setting, or knowledge library changes, the Direct-vs-Knowledge boundary may shift, and the route model may need to be updated accordingly.

References

Akari Asai, Zeqiu Wu, Yizhong Wang, Avi Sil, and Hannaneh Hajishirzi. 2024. Self-rag: Learning to retrieve, generate, and critique through self-reflection. In *International Conference on Learning Representations*, volume 2024, pages 9112–9141.

Tarek Azim, Iddo Drori, and 1 others. 2023. Putnambench: A benchmark for formalized mathematics at the olympiad level. *arXiv preprint arXiv:2304.03215*.

Yves Bertot and Pierre Castéran. 2004. *Interactive Theorem Proving and Program Development: Coq'Art: The Calculus of Inductive Constructions*. Texts in Theoretical Computer Science. Springer.

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*, volume 9195 of *Lecture Notes in Computer Science*, pages 378–388. Springer.

Soyeong Jeong, Jinheon Baek, Sukmin Cho, Sung Ju Hwang, and Jong C. Park. 2024. Adaptive-RAG: Learning to adapt retrieval-augmented large language models through question complexity. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 7036–7050, Mexico City, Mexico. Association for Computational Linguistics.

Albert Q. Jiang, Wenda Li, and Mateja Jamnik. 2024. Multi-language diversity benefits autoformalization. In *Advances in Neural Information Processing Systems*, volume 37, pages 83600–83626. Curran Associates, Inc.

Rui Li, Liyang He, Qi Liu, Zheng Zhang, Heng Yu, Yuyang Ye, Linbo Zhu, and Yu Su. 2025. UniRAG: Unified query understanding method for retrieval augmented generation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 14163–14178, Vienna, Austria. Association for Computational Linguistics.

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

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

Wangyue Lu, Lun Du, Sirui Li, Ke Weng, Haozhe Sun, Hengyu Liu, Minghe Yu, Tiancheng Zhang,

and Ge Yu. 2026. Automated formalization via conceptual retrieval-augmented llms. *Preprint*, arXiv:2508.06931.

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

Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. 2002. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer.

Auguste Poiroux, Gail Weiss, Viktor Kunčak, and Antoine Bosselut. 2025. Reliable evaluation and benchmarks for statement autoformalization. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 17947–17969, Suzhou, China. Association for Computational Linguistics.

Weijia Shi, Sewon Min, Michihiro Yasunaga, Minjoon Seo, Richard James, Mike Lewis, Luke Zettlemoyer, and Wen-tau Yih. 2024a. REPLUG: Retrieval-augmented black-box language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 8371–8384, Mexico City, Mexico. Association for Computational Linguistics.

Zhengliang Shi, Shuo Zhang, Weiwei Sun, Shen Gao, Pengjie Ren, Zhumin Chen, and Zhaochun Ren. 2024b. Generate-then-ground in retrieval-augmented generation for multi-hop question answering. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7339–7353, Bangkok, Thailand. Association for Computational Linguistics.

Fei Wang, Xingchen Wan, Ruoxi Sun, Jiefeng Chen, and Sercan Ö. Arik. 2025. Astute RAG: Overcoming imperfect retrieval augmentation and knowledge conflicts for large language models. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics*.

Ruida Wang, Jipeng Zhang, Yizhen Jia, Rui Pan, Shizhe Diao, Renjie Pi, and Tong Zhang. 2024. Theorem-Llama: Transforming general-purpose LLMs into Lean4 experts. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 11953–11974. Association for Computational Linguistics.

Huajian Xin, Daya Guo, Zhihong Shao, Zhizhou Ren, Qihao Zhu, Bo Liu, Chong Ruan, Wenda Li, and Xiaodan Liang. 2024. DeepSeek-Prover: Advancing theorem proving in LLMs through large-scale synthetic data. In *NeurIPS 2024 Workshop on MATH-AI*.

Huajian Xin, Zhizhou Ren, Junxiao Song, Zhihong Shao, Wanjia Zhao, Haocheng Wang, Bo Liu, Liyue Zhang, Xuan Lu, Qiushi Du, Wenjun Gao, Haowei Zhang, Qihao Zhu, Dejian Yang, Zhibin Gou, Zhihong Wu, Fuli Luo, and Chong Ruan. 2025. DeepSeek-Prover-V1.5: Harnessing proof assistant feedback for reinforcement learning and monte-carlo tree search. In *International Conference on Learning Representations*.

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

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

Ori Yoran, Tomer Wolfson, Ori Ram, and Jonathan Berant. 2024. Making retrieval-augmented language models robust to irrelevant context. In *International Conference on Learning Representations*.

Yue Yu, Wei Ping, Zihan Liu, Boxin Wang, Jiaxuan You, Chao Zhang, Mohammad Shoeybi, and Bryan Catanzaro. 2024. RankRAG: Unifying context ranking with retrieval-augmented generation in LLMs. In *Advances in Neural Information Processing Systems*.

Lan Zhang, Xin Quan, and Andre Freitas. 2024. Consistent autoformalization for constructing mathematical libraries. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 4020–4033, Miami, Florida, USA. Association for Computational Linguistics.

Lan Zhang, Marco Valentino, and Andre Freitas. 2025. Autoformalization in the wild: Assessing LLMs on real-world mathematical definitions. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 1720–1738, Suzhou, China. Association for Computational Linguistics.

A Appendix

A.1 Case Study of Counterfactual Utility Filtering

We provide a representative case study to illustrate why relevance-based retrieval is insufficient for autoformalization and how counterfactual utility filtering identifies knowledge that better supports formal anchoring.

In this example, the failure of relevance-based retrieval does not arise because the retrieved concepts are irrelevant. Instead, many retrieved concepts only describe surface mathematical semantics, such as geometric interpretation and coordinate descriptions, while weakly constraining the

proof-assistant-level structures required for verifier-accepted formalization.

The original problem asks for the intersection point between a line and the curve defined by $x^y = y^x$. Relevance-based retrieval returns concepts such as `graph`, `first quadrant`, and `intersection point`, which are all semantically related to the problem. However, these concepts do not directly constrain the Lean objects, type structures, or formal relations needed for verifier-accepted formalization. As a result, relevance-based retrieval expands the semantic context without sufficiently narrowing the formal construction space.

Counterfactual deletion exposes this mismatch. When the concept cluster {`graph`, `first quadrant`, `coordinates`, `intersection point`} is removed, the resulting subproblem becomes compilable and semantically aligned. This indicates that the deleted concepts have low utility for formal anchoring: although semantically related, they weakly constrain the verifier-required formal structures. After removing these concepts, the remaining concepts focus more directly on the formal relation $x^y = y^x$, allowing the model to recover the corresponding Lean-level formal anchors.

Table 4 summarizes the overall counterfactual utility filtering process, including the retrieved concepts, the deleted concept cluster, the utility-selected concepts, and the verifier outcomes.

A.2 Quality of Counterfactual Subproblem Construction

We further examine whether the synthesized counterfactual subproblems are meaningful mathematical problems rather than arbitrary corruptions of the original statements. A high-quality counterfactual subproblem should preserve the mathematical core of the original problem while removing the targeted concept cluster. This property is important because counterfactual utility filtering relies on verifier feedback from these subproblems: if the synthesized subproblem is poorly formed, its verifier outcome would not provide a reliable utility signal.

A counterfactual subproblem is meaningful if it satisfies two conditions. First, it preserves the main mathematical object or reasoning structure of the original problem, so that verifier feedback remains informative for the original formalization task. Second, it removes the targeted concept cluster rather than arbitrarily changing the problem, so that success or failure on the subproblem can be interpreted as evidence about the utility of that cluster.

The counterfactual subproblems shown in Table 5 satisfy these properties. In the trigonometric case, the same function f_N and sine-sum construction are preserved after removing the extremal inequality and optimal constant. In the geometry case, the same triangle configuration and midpoint relation are preserved after removing the operational dissection and reassembly requirements. In the PDE case, the target finite-difference expression $m(f)$ and the minimization objective are retained after replacing derivative-based assumptions with discrete functional identities.

These examples suggest that counterfactual deletion produces diagnostically meaningful subproblems rather than arbitrary perturbations.

A.3 Full Prompts

We provide the full prompts used in different stages of our framework. The prompts are placed after the counterfactual case studies because they serve as implementation details. Each prompt is grouped by its role, input variables, system message, and user message.

A.4 Runtime Environment Configuration

To ensure consistency and reproducibility of experimental results, all experiments were conducted under the following hardware and software environment. The software environment is shown in Table 6.

Hardware configuration.

- **GPU:** 4×NVIDIA A800 80GB PCIe
- **GPU memory:** 80GB per card (total 320GB)
- **Operating mode:** PCIe Gen4, NVLink not enabled
- **CPU:** AMD EPYC 7R32 (128 cores / 256 threads)
- **System memory:** 1TB DDR4 ECC RAM

Component	Content	Verification / Result
Original Problem	Find the intersection point between the line and curve in the graph of $x^y = y^x$ in the first quadrant.	Direct formalization: Fail
Retrieved Concepts	`exponential equation`; `curve`; `intersection point`; `first quadrant`; `graph`	Relevance-based RAG: Semantic alignment unstable
Counterfactual Deletion	Delete concept cluster: `graph`; `first quadrant`; `coordinates`; `intersection point`	Counterfactual subproblem: Compile = Pass Semantic = Entailment
Utility-Selected Concepts	`equation`; `straight line`; `curve`	Utility-aware retrieval: Compile = Pass Semantic = Entailment

Table 4: Case study illustrating counterfactual utility filtering. The deleted concept cluster is semantically related to the original problem but weakly constrains the verifier-required formal structures. After removing these concepts, the remaining concepts more directly support Lean-level formal anchor construction.

Original Problem	Removed Concept Cluster	Counterfactual Subproblem
Determine the smallest constant C_N such that $\left\lvert \sum_{n=0}^{N} \frac{N+\frac{1}{2}-n}{(N+1)(2n+1)} \sin((2n+1)x) \right\rvert \le C_N$ holds for all real x.	`{inequality, constant}`	For a positive integer N, define $f_N(x) = \sum_{n=0}^{N} \frac{N+\frac{1}{2}-n}{(N+1)(2n+1)} \sin((2n+1)x).$ Prove that f_N is odd and 2π-periodic.
Show that one triangle can be dissected into finitely many polygonal pieces and reassembled into another configuration of equal area.	`{dissection, reassembly}`	Let M be the midpoint of side BC of triangle ABC. Prove that triangles AMB and AMC have equal area.
Let $\mathcal{F}$ be the family of twice continuously differentiable functions satisfying a partial differential equation on $[1,\infty) \times [1,\infty)$. For each $f \in \mathcal{F}$, define a quantity $m(f)$ and determine its minimum value. Prove that the minimum is independent of the choice of f.	`{partial derivative, twice continuously differentiable, partial differential equation}`	Let $\mathcal{F}$ be the set of functions satisfying $f(x+1,y+1) - f(x+1,y) - f(x,y+1) + f(x,y) = 1.$ For each $f \in \mathcal{F}$, define $m(f) = \min_{s \ge 1}\big(f(s+1,s+1) - f(s+1,s) - f(s,s+1) + f(s,s)\big).$ Determine $m(f)$.

Table 5: Examples of high-quality counterfactual subproblems. Each counterfactual problem removes the targeted concept cluster while preserving the mathematical core of the original problem.

Category	Version / Configuration
Operating System	Ubuntu 23.10
GPU Driver	535.171.04
CUDA	12.2
Python	3.13.5
PyTorch	2.11.0
vLLM	0.21.0
Transformers	5.8.1
Lean Compiler	v4.20.0-rc5
Elan	4.2.1
Mathlib4	leanprover/lean4:v4.20.0-rc5

Table 6: Software versions and runtime configuration used in the experiments.

Direct Formalization Prompt

Purpose: Directly generate Lean formalization without external knowledge.
Variables: `problem`
System Prompt:

```
You are an expert in mathematics and Lean 4.
```

User Prompt:

```
Please autoformalize the following problem in Lean 4 with a header. Use the following theorem names: my_favorite_theorem.

{problem}
```

Knowledge-Assisted Formalization Prompt

Purpose: Generate Lean formalization with retrieved knowledge context.
Variables: `problem`, `context_str`
System Prompt:

```
You are an expert mathematician specializing in Lean 4 formalization. You must always produce complete, compilable Lean 4
     code starting with the required imports.
```

User Prompt:

```
Task: Autoformalize the following mathematical problem into Lean 4.

REQUIREMENTS:
1. The theorem name must be exactly: my_favorite_theorem
2. Do not include any proof, just the theorem statement with "sorry"
3. Your entire response should be only valid Lean 4 code
4. No explanations, no markdown markers, just pure Lean code

Use the mathematical definitions provided below for reference, but DO NOT translate the definitions themselves.

Problem to translate:
{problem}

Reference definitions (for context only):
{context_str}
```

Counterfactual Subproblem Synthesis Prompt

Purpose: Generate new mathematical proof problems by removing concept clusters.
Variables: `original_problem`, `concepts`
System Prompt:

```
You are an expert mathematician.
```

User Prompt:

```
Role:
You are a rigorous mathematical problem designer with strong expertise in logical consistency, concept control, proof-based
     problem formulation, and concept dependency analysis.

Task Objective:
I will provide you with:
1. an Original Mathematical Problem, and
2. a list of mathematical concepts involved in the original problem.

Your task is to generate a sequence of new mathematical proof problems using the following two-stage process:

Stage 1: Concept Clustering
- First, analyze the given list of mathematical concepts and group them into clusters based on strong mathematical
     dependency, equivalence, or mutual recoverability.
- Concepts should be placed in the same cluster if removing one while keeping the others would be mathematically unstable (i.
     e., the removed concept would automatically reappear through standard equivalences).
- Clearly determine the final number of concept clusters K.

Stage 2: Cluster-Based Concept Removal
- You must generate K new mathematical problems.
- For the i-th new problem, assume that the entire i-th concept cluster has been removed and must NOT be used in any form.
```

```
Definition of "Removed Concept Cluster" (CRITICAL):
For the i-th new problem, all concepts in the removed cluster must NOT:
1. Appear explicitly in wording, notation, or assumptions
2. Appear implicitly through equivalent formulations, standard characterizations, or logical consequences
3. Serve as:
   - the main criterion of the statement,
   - an equivalence condition,
   - a conclusion-defining property,
   - or an indispensable intermediate step in the intended proof

If removing a concept cluster makes the original type of claim invalid, you MUST change the nature of the statement rather
      than force an incorrect or trivial result.

General Requirements (apply to all generated problems):
1. Each new problem must be a proof problem, formulated using expressions such as "Prove that ..." or "Prove the following
      statement."
2. Each problem must be mathematically correct, logically coherent, and well-posed. Mathematical validity has the highest
      priority.
3. The removed concept cluster must not play the same logical role as in the original problem, even in weakened or
      background form.
4. Avoid introducing new mathematical concepts beyond those appearing in the remaining concept clusters as much as possible.
   - Basic logical language (equality, quantifiers, elementary set notation) may be used only as neutral syntax and must not
       become a conceptual substitute for the removed cluster.
5. The new problems do not need to preserve the original mathematical claim, but they may be inspired by the original
      problem's style, level of abstraction, or proof structure.
6. For each new problem, explicitly ensure that:
   - a reader with no prior knowledge of the removed concept cluster can still fully understand the problem statement, and
   - the problem remains meaningful and solvable without that cluster.

Output Format:
- Output a single JSON array and NOTHING ELSE.
- Do NOT include markdown, explanations, commentary, or extra text.
- The JSON array length MUST be exactly K.

Each element MUST have the following structure:
{
  "index": <integer starting from 1>,
  "removed_concept_cluster": [<string>, ...],
  "problem_statement": "<string>"
}

Rules:
1. "problem_statement" MUST start with "Prove that ..." or "Prove the following statement."
2. "removed_concept_cluster" MUST list all concepts in the removed cluster.
3. Do NOT include proofs, solutions, explanations, or clustering analysis.
4. Do NOT include any additional fields.
5. The output MUST be valid JSON.

Original Mathematical Problem:
{original_problem}

Mathematical Concepts:
{concepts}
```