Hypothesis Decomposition and Retrieval for Autoformalization of Discrete Mathematics

Agatha Duzan
EPFL
agatha.duzan@epfl.ch

Abstract

Formalizing natural language mathematics into formal proof assistant syntax, known as *autoformalization*, remains a challenge in automated theorem proving. Although recent large language models have shown some progress, achieving correct formalizations has proved difficult. In this work, we propose two complementary methods to enhance autoformalization of discrete mathematics in Lean 4. First, we add a step of explicitly decomposing each informal statement into its premises and goals, aiming to clarify the underlying logical structure. Second, we incorporate retrieval from extensive formal libraries—using both a proof-state-based retriever and a semantic search engine—to surface relevant definitions or structures that reduce hallucinations and type-check failures.

We evaluate our approaches on a curated dataset of 234 informal–formal statement pairs, measuring correctness with type-checking, text similarity (BLEU), and the bidirectional definitional equivalence metric (BEq). Although single-method gains over the direct translation baseline are modest, we find that these approaches provide complementary successes: combining them yields a higher coverage of correct formal statements. Our results indicate that carefully integrated decomposition and retrieval steps can resolve certain failures, suggesting that multi-strategy pipelines with automated output selection are a promising path forward for large-scale autoformalization efforts in mathematics.

1 Introduction

Automated theorem proving has long promised to transform mathematical practice by providing rigorous, machine-verified proofs at scale. Achieving this vision, however, requires statements and concepts to be expressed in a formal language like Lean or Isabelle, where proof assistants can guarantee the correctness of a proof. That first step of translating natural language, unconstrained form, mathematical statements into a formal language is referred to as *autoformalization*, and remains a bottleneck.

Recent advances in LLMs have shown some progress in autoformalization, yet the process remains far from satisfactory, leaving substantial room for improvement. Even the most advanced pipelines fail to produce consistently valid formal statements (Wu et al., 2022; Poiroux et al., 2024), and error rates remain quite high. Furthermore, the scarcity of gold-standard informal-to-formal datasets poses a significant challenge. Unlike large general-domain NLP tasks, which benefit from extensive training data, autoformalization efforts are often constrained to a few thousand paired statements at most—a notable limitation. In this work, we explore two complementary strategies to improve autoformalization of discrete mathematics. First, we incorporate an *informal hypothesis decomposition* step, prompting LLMs to explicitly list premises and goals before synthesizing the final Lean statement. Second, we integrate *retrieval* from large formal libraries, on the hypothesis that surfacing the relevant definitions or structures can guide the model away from hallucinations and type-check failures. By testing these methods independently and in combination, we aim to establish which approachs can meaningfully improve correctness over a direct translation baseline.

Our findings indicate that while it is difficult to substancially increase the overall translation accuracy, certain types of decomposition and retrieval indeed offer complementary benefits—yielding correct formalizations on instances where a single approach fails. This suggests a promising direction of combining multiple, specialized autoformalization strategies and then selecting among candidate outputs via automated checks and heuristics. In short, although we are still far from a fully reliable pipeline, the experiments here underscore the value

of structured reasoning steps and targeted library retrieval for autoformalization.

2 Related Work

2.1 Autoformalization and Automated Theorem Proving

Recent work has explored various ways to improve accuracy in autoformalization and automated theorem proving, but no single approach stands out as being significantly better than the rest, and breakthrough results in this area remain elusive.

Wu et al. (2022) propose an iterative strategy, starting with few-shot translation from informal to formal statements and refining the model using the newly generated proofs. Despite relative gains in accuracy , the approach suffers from high computational cost and remains limited to more accessible math problems. Notably, their reported best accuracy for formalization is only around 30%, suggesting room for improvement.

Another approach is DeepSeek-Prover, by Xin et al. (2024). It focuses on synthetic data generation to improve automated theorem proving (but raises questions on synthetic data for autoformalization as well). Although filtering heuristics remove many incorrect or trivial statements, about 20% of the final statements are false, underscoring the difficulty of generating large-scale and high-quality synthetic data for formal mathematics.

Jiang et al. (2022) propose another line of research with Draft, Sketch, and Prove. They introduce a multi-step pipeline for formal proofs: (i) drafting an informal proof, (ii) sketching a high-level formal proof outline, and (iii) calling on automated provers (like Sledgehammer in Isabelle) to fill in the details. Their results suggest that structured decomposition and partial automation can improve formalization outcomes.

2.2 Retrieval

A promising step for autoformalization is retrieval from large formal mathematics libraries.

Yang et al. (2024) introduce LeanDojo, which uses a retrieval-augmented system (ReProver) to fetch relevant premises from the Lean Mathlib library. The retriever is trained with example pairs of relevant/irrelevant premises in context, enabling more precise identification of relevant definitions and theorems. Once retrieved, these items are combined with the current proof state and fed into an encoder-decoder model to generate the next proof step. This approach surpasses GPT-4 on theorem-proving (on the benchmark introduced in the same paper), highlighting the effectiveness of good retrieval mechanisms in formal theorem proving.

Similarly, Anonymous (2024) incorporate a retrieval module into their RAutoformalizer framework (Retrieval-Augmented Autoformalizer) to reduce the risk of hallucinations, a common issue in autoformalization where the model invents non-existent or incorrect definitions. By analyzing dependencies within formal libraries, they present a topologically ordered dataset of informal–formal pairs. When a new statement arrives, the system retrieves only the relevant objects (definitions, structures) to guide formalization, which significantly reduces hallucinations and improves overall correctness.

Focusing on natural language queries, Gao et al. (2024) introduce LeanSearch, a semantic search engine for Mathlib4. Rather than searching by exact theorem names or Lean-specific documentation, users can input an informal query to retrieve formal theorems or definitions with semantic similarity. Although primarily designed for theorem retrieval, this tool can also locate definitions or structures, making it valuable for retrieval in autoformalization pipelines.

2.3 Benchmarks and Datasets

Multiple benchmarks target the evaluation of autoformalization and theorem proving. MiniF2F (Zheng et al., 2021) compiles high-school to early-university math problems (e.g., from the International Mathematical Olympiad) and pairs their informal and formal statements. PutnamBench (Tsoukalas et al., 2024) also pairs informal/formal statements, focusing on challenging undergraduate-level problems from the Putnam Mathematical Competition. ProofNet (Azerbayev et al., 2023) offers an even broader range of formal reasoning tasks—including theorem proving, proof completion, proof verification, and more—evaluating a system's ability to manipulate multiple aspects of formal logic and mathematics.

2.4 Automated Metrics for Evaluation

Although human evaluation of formal statements is typically the gold standard, it is slow, expensive, and requires specific expert knowledge, which makes it unpractical for large-scale assessments. Developing automated metrics that closely align with human judgment on the equivalence of for-

mal mathematical statements remains a significant challenge.

Poiroux et al. (2024) 's work highlight that type checking is as a necessary condition for the correctness of a generated formal statement. Combining type checking with the BLEU score, provides a proxy metric (TC-BLEU) for the equivalence of two formal statements.

The BEq metric (Anonymous, 2024) offers another perspective by explicitly checking equivalence between two formal statements. While this approach is highly precise, yielding no false positives, it tends to be overly strict, leading to frequent false negatives. Consequently, BEq can serve as a conservative lower bound for human evaluation.

3 Method

Our goal is to improve the accuracy of autoformalization by introducing intermediate steps in the translation process. We begin with a baseline of direct translation, and then explore three more advanced approaches that build upon this baseline.

Specifically, we investigate how (i) adding a hypothesis-decomposition step, (ii) incorporating retrieval from formal libraries, or (iii) combining both decomposition and retrieval can enhance autoformalization.

3.1 Baseline

Our baseline is the direct translation: given the informal statement of a theorem, we prompt a LLM to directly produce its formal statement in Lean 4. This direct method uses a carefully designed prompt and few-shot examples of informal-formal statement pairs (see example in A.1).

3.2 Informal Hypothesis Decomposition

The key idea behind our first approach is to make the underlying logical structure of a theorem more explicit. As illustrated in Figure 1, we start by taking the informal statement and prompting the LLM to extract the relevant premises (hypotheses and assumptions) and goal (what has to be proven).

Then, we form the full prompt containing the informal statement, hypothesis decomposition, and few-shot examples. The final output is the formalization of the statement.

In other words, hypothesis decomposition acts as a structured chain-of-thought, prompting the model to specify each assumption before attempting a formalization of the entire statement.

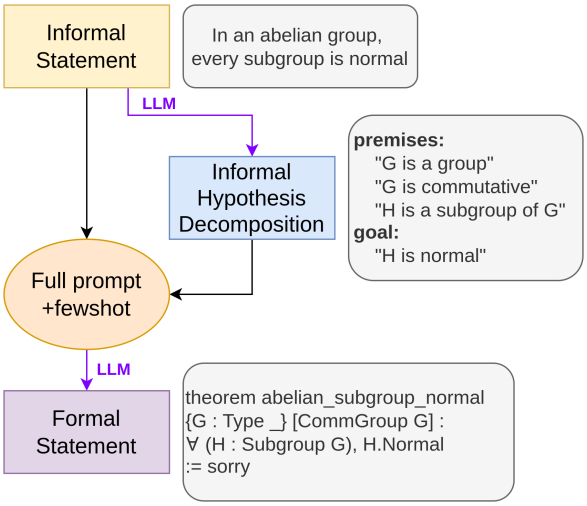

Figure 1: Informal Hypothesis Decomposition

3.3 State–based Retrieval

Our second approach augments the baseline with a retrieval step: it leverages the proof state of an initial (and potentially imperfect) formalization to pull in relevant context from the Mathlib library.

Figure 2 depicts this method: we first prompt the LLM to produce a tentative Lean 4 statement directly from the informal statement, similar to our baseline. Then, we convert this preliminary statement into proof-state format, i.e., the abstract context that Lean would display if one tried to prove the statement interactively. We feed this to the LeanDojo retriever, which is trained to encode proof states and retrieve relevant snippets from Mathlib (type definitions, structures, etc.). Finally, we form the full prompt containing the informal statement, retrieved snippets, and few-shot examples. The output is the formalization of the statement.

If the retrieval can provide relevant parts of the Mathlib library, we expect it can help the model generate an accurate and type-correct formal statement.

3.4 Hypothesis-guided LeanSearch Retrieval

Our last approach combines hypothesis decomposition with a targeted retrieval strategy, as shown in Figure 3.

We first prompt the model to extract the hypotheses and convert them into brief natural-language queries (we use few-shot with specific examples of decomposition and queries for this step). Then, we feed each query into LeanSearch (Gao et al., 2024), a semantic search engine for Mathlib4, which is capable of returning definitions, structures, or theo-

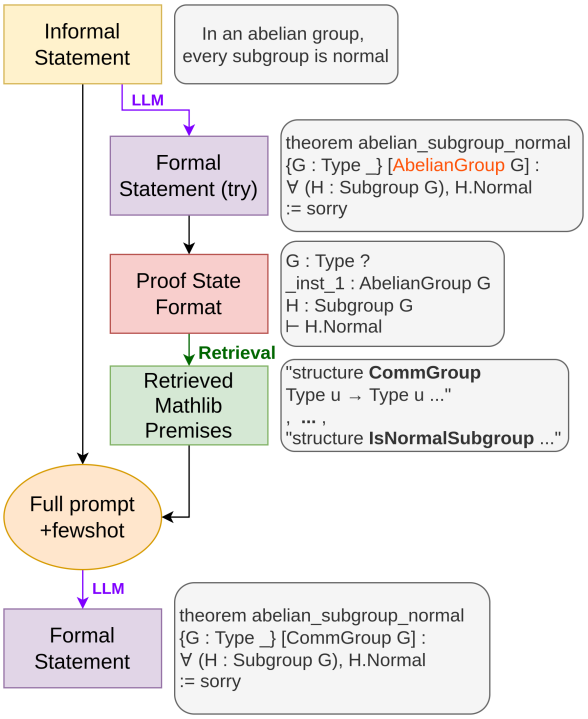

Figure 2: State–based Retrieval with LeanDojo. The retrieval can be performed even when the first formal statement contains errors.

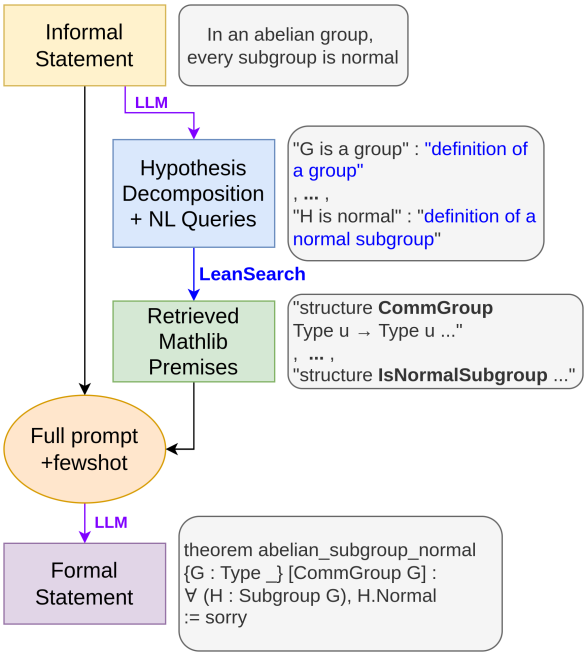

Figure 3: Hypothesis Decomposition and Retrieval

rems given a query in natural language (we discard theorems and lemmas from the retrieval since they are less useful for autoformalization). Finally, we form the full prompt containing the informal statement, retrieved snippets, and few-shot examples. The output is the formalization of the statement.

By focusing retrieval on each individual hypothesis, we aim to retrieve more targeted and relevant definitions for the statement at hand. However, to assess whether this fine-grained approach truly outperforms a single-query baseline, we also test an alternative configuration where the entire informal statement is passed to LeanSearch. This comparison should elicit whether retrieval with hypothesis decomposition offers a significant advantage over simply retrieving using the entire statement.

3.5 Evaluation

To assess the correctness and quality of our autoformalization methods, we need to evaluate the generated formal statements against the golden formal statements.

For a broad comparison, we use the BLEU text similarity metric: it measures n-gram overlap and is commonly used to evaluate machine translation tasks. While it offers a rough estimation of lexical and semantic similarity, it is insufficient for capturing the precise logical equivalence required by formal mathematics.

That's why we also use two formal verification metrics that leverage Lean: type-checking and BEq.

3.5.1 Type checking

Our first line of correctness verification is type-checking. Concretely, we attempt to compile the generated formal statement on a Lean server.

We prepend necessary imports ('header' attribute, or by default 'import Mathlib') and strip out extraneous comments or proof bodies, leaving only the statement. If Lean's REPL (Read-Eval-Print Loop) raises an error, we mark the type check as failed. Otherwise, if Lean parses and accepts the statement as well-typed, we record a success.

A statement that fails to type-check cannot be correct in the formal sense, making this a strict binary metric. It's important to note that passing type checking is only a necessary (but not sufficient) condition for correctness.

3.5.2 Bidirectional Extended Definitional Equivalence

To more precisely capture logical equivalence between two formal statements (golden vs generated), we use Bidirectional Extended Definitional Equivalence (BEq). This metric, inspired by prior work on autoformalization, checks whether each statement can be used to prove the other.

Specifically: we interpret both formal statements in Lean (with appropriate environment and header). For each direction, we attempt to show one statement is derivable from the other using Lean tactics. If both directions succeed, we conclude the statements are equivalent in a definitional sense; if either direction fails, we cannot confirm equivalence.

Because BEq tests each direction, it is more robust than purely textual metrics—if a statement is textually close but mathematically stronger than the other one, the equivalences fail. Although BEq is very conservative (and can over-reject valid variants), it offers a valuable lower bound on equivalence.

4 Experimental Setup

4.1 Datasets

For this project, we focus primarily on discrete mathematics, where the step to formalizing statements in Lean is often more straightforward. This choice reflects both personal familiarity with the field and the practical necessity of restricting the scope of a semester-long project.

We also created a smaller abstract algebra dataset for exploratory purposes (see Appendix A.3), curated similarly to the number theory dataset.

4.1.1 Number Theory Dataset

To build our number theory dataset, we aggregated formal/informal pairs from two main sources:

- **MiniF2F** (Zheng et al., 2021): We extracted all theorems whose name contains 'numbertheory', yielding 136 entries. Of these, 120 came from the MATH dataset (Hendrycks et al., 2021) and 16 were custom theorems.

- **PutnamBench** (Tsoukalas et al., 2024): We retained theorems tagged as 'number_theory', 'combinatorics', 'set_theory', or 'probability', producing 110 entries.

We then performed a minimal preprocessing on these statements: we split out imports and open declarations (these headers are necessary for type checking but extraneous for other text-based evaluations), and clean up the statements by removing LaTeX-style formatting.

The end result is a dataset of 234 formal/informal pairs, available on Hugging Face.

4.1.2 Mathlib Corpus for Retrieval

To support the state-based retrieval approach , we built a corpus by scraping all dependencies in Mathlib version 4.14.0. This process gave us roughly 200k items, which we filtered to exclude theorems and lemmas (less relevant for statement translation), leaving 59k items. We then encoded each item using the trained model from the LeanDojo repository, giving us a corpus of Mathlib snippets and their corresponding encodings that we can leverage for retrieval.

4.2 Implementation Details

We use GPT-4o (exact model: gpt-4o-2024-11-20) as our primary generator, given its strong performance on mathematical and reasoning tasks. All experiments set the model's temperature to 0 for deterministic decoding and a maximum output length of 1000 tokens.

To guide the system, we use 8 fewshot examples that are intentionally not from discrete mathematics, ensuring the model has a general sense of how to formalize a statement in Lean without relying on any "built-in" solutions for the same domain.

For the state-based retrieval, we rely on the model provided by LeanDojo (kaiyuy/leandojo-lean4-retriever-byt5-small). Given prior findings in the literature, we test two settings for the number of retrieved premises (k=3 and k=5).

For the LeanSearch-based retrieval, we also test with k=3 and k=5 retrieved premises for the baseline that uses the entire informal statement as the search query. In the hypothesis-decomposition variant, each hypothesis is turned into its own LeanSearch query, from which we retrieve k=1 premise per hypothesis.

5 Results

Table 1 summarizes our main results, focusing on type-check rate, TC-BLEU, and BEq metrics. We compare the baseline methods against our proposed approaches, and look at the impact of adding different intermediate steps.

Our first observation is that overall performance metrics fall within a relatively narrow range ($21\% \pm 2\%$ on BEq), suggesting that GPT-4o may not always fully capitalize on additional cues such as decomposed premises or retrieved library snippets. Nevertheless, we see some interesting trends.

The results show that adding the informal hypothesis decomposition step degrades the overall

Method	BEq (%)	Type-Check (%)	TC-BLEU
Base	20.51	47.01	0.195
Base - no fewshot	19.66	43.16	0.177
Informal hypothesis decomposition	19.23	38.03	0.165
State-based retrieval (top-k = 3)	20.09	52.99	0.201
State-based retrieval (top-k = 5)	21.79	53.42	0.200
LeanSearch retrieval baseline (top-k = 3)	20.51	52.14	0.204
LeanSearch retrieval baseline (top-k = 5)	20.94	**54.27**	0.202
Hypothesis-guided LeanSearch retrieval	**22.22**	52.56	**0.204**

Table 1: Main results table

performance: this suggests that in most cases, this step introduces more noise than useful information for the model. On the other hand, none of the retrieval methods worsens performance compared to the baseline: even small improvements suggest that the retrieved examples are relevant (otherwise it would be unrelevant 'noise' again and worsen performance).

Our method of hypothesis-guided LeanSearch retrieval yields the highest BEq (22.22%) , while the highest type check rate (54.27%) is attained with the LeanSearch retrieval baseline (top-k = 5).

5.1 Complementarity of Methods

We saw that our metrics don't vary much across different methods, but we also want to examine the overlap of successes: do methods with similar BEq succeed on the same informal statements?

Given a binary metric, we define a complementarity score $S = \frac{U_1+U_2}{U_1+U_2+B}$, where U_1 and U_2 are the unique successes of Methods 1 and 2, respectively, and B is the number of successes shared by both. A higher S (closer to 1) indicates that the methods succeed in more disjoint sets of statements and are therefore more complimentary (using them together would boost coverage).

As shown in Figure 4, different variants of the same approach have low complimentarity. Notably, the hypothesis-guided LeanSearch retrieval is very redundant with the LeanSearch baselines.

More interestingly, we see that informal hypothesis decomposition has a high complimentarity with all other methods: even though it individually performs worse, it occasionally formalizes statements that the other methods fail, making it an interesting candidate to be used jointly with other methods.

5.2 Combining Multiple Methods

These findings motivate us to experiment with combining methods. Specifically, for each informal statement, we generate multiple formal statements (one from each method) and select the one yielding the highest BEq score (or highest type-check if all fail BEq).

The results are shown in Table 2: we see that combining complimentary methods (baseline, LeanSearch top5 retrieval, state-based top5 retrieval and informal hypothesis decomposition) produces the best results overall: 28.21% BEq and 61.97% type-checking.

This shows that weaker-performing approaches like informal hypothesis decomposition can carry unique strengths for certain statements, making them valuable in a combined pipeline.

6 Discussion

One of the clearest insights from this project is that it remains challenging for LLMs—even high performance models like GPT-4—to reliably leverage additional information to significantly improve formalization outcomes. While our retrieval-based methods led to modest gains, these improvements were smaller than anticipated. This suggests that naïve injection of contextual snippets or a decomposition of hypotheses does not consistently help unless the model can fully integrate the supplied information into the reasoning process.

The informal hypothesis decomposition approach showed lower raw performance, yet it exhibited strong complementarity: it succeeded in formalizing certain statements that the other methods failed. This implies that structural prompts (like explicit hypothesis decomposition) may be relevant in specific scenarios or statement types. Although such an approach can introduce noise or redundancy in general, it can still add value when combined with other methods, when the final formal statement is chosen among multiple candidate formalizations.

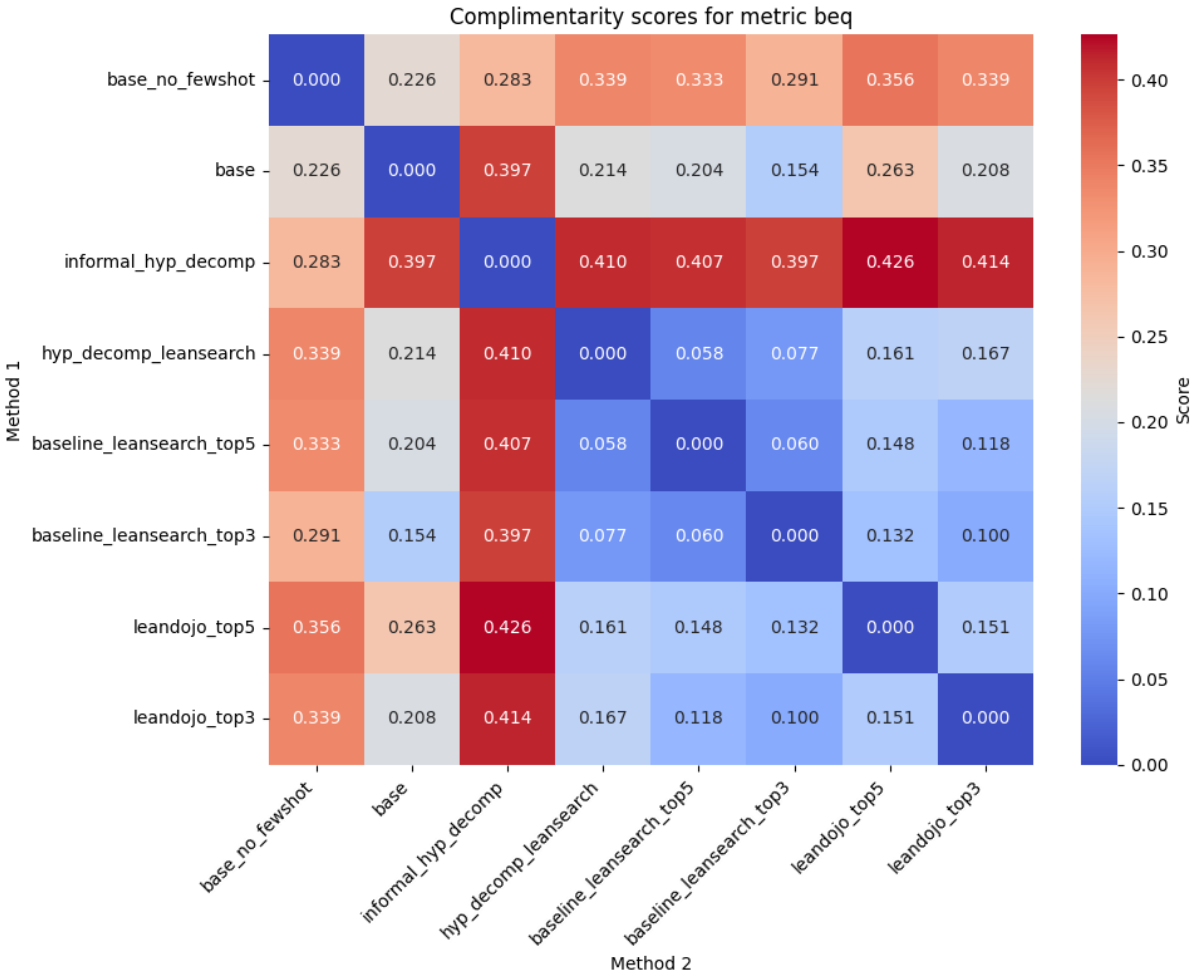

Figure 4: Complimentarity of our methods, according to uniqueness of BEq successes

N	Best combination	BEq (%)	Type-Check (%)
2	Base + LeanSearch top5	23.08	56.84
3	Base + LeanSearch top5 + Informal hypothesis decomposition	26.50	59.40
4	Base + LeanSearch top5 + State-based top5 + Informal hypothesis decomposition	**28.21**	**61.97**

Table 2: Best combinations of N methods

Our results also underscore the importance of using multiple evaluation metrics. Type checking alone cannot guarantee logical equivalence, while the stricter BEq metric leads to an over-rejection of statements, particularly for more complex or lengthy ones. The observed discrepancy highlights the continuing need for better automated metrics that balance precision with tolerance for semantically valid variations in formal language.

Overall, the project reveals that (1) retrieval can be integrated without harm, but its best application likely requires models trained to make full use of retrieved snippets, and (2) complementary strategies—like combining decomposition and retrieval—can capture a broader set of successes than any single method.

7 Future Work

Several directions stand out as promising ways to build on this work. First, scaling up both the model and sampling could improve the rate of correct formalization. For instance, using OpenAI's o1 model (better mathematical reasoning capabilities) and then sampling multiple candidate formalizations for each statement could increase coverage. Subsequent filtering—through type checking, BEq, or other heuristics—would keep only the valid or likely-correct formalizations.

Second, although our study tested how hypothesis decomposition and retrieval can help, it did not

explore systematic ways of fine-tuning the model to specifically incorporate this additional information. Developing training strategies that enable effective use of retrieved snippets or decomposed hypotheses may enable models to extract greater value from intermediate steps.

A third avenue for future work is to improve automated metrics for autoformalization. Our findings underscore that metrics like type checking and BEq, although useful, have limitations in coverage, precision, or leniency. Designing more sophisticated equivalence checkers—or augmenting existing ones with approximate semantic matching—would facilitate quicker and more accurate evaluation, especially for large-scale experiments.

Finally, building larger and higher-quality datasets remains crucial. The scarcity of extensive, reliable informal-to-formal pairs limits both supervised learning and evaluation. A promising direction is to generate synthetic data that is mathematically sound and includes diverse examples covering various domains and difficulty levels. Curating or improving such synthetic resources could significantly advance the community's ability to develop and test new autoformalization methods.

References

Anonymous. 2024. Rethinking and improving autoformalization: towards a faithful metric and a dependency retrieval-based approach. In *Submitted to The Thirteenth International Conference on Learning Representations*. Under review.

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*.

Guoxiong Gao, Haocheng Ju, Jiedong Jiang, Zihan Qin, and Bin Dong. 2024. A semantic search engine for mathlib4. *Preprint*, arXiv:2403.13310.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. *Preprint*, arXiv:2103.03874.

Albert Q Jiang, Sean Welleck, Jin Peng Zhou, Wenda Li, Jiacheng Liu, Mateja Jamnik, Timothée Lacroix, Yuhuai Wu, and Guillaume Lample. 2022. Draft, sketch, and prove: Guiding formal theorem provers with informal proofs. *arXiv preprint arXiv:2210.12283*.

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

George Tsoukalas, Jasper Lee, John Jennings, Jimmy Xin, Michelle Ding, Michael Jennings, Amitayush Thakur, and Swarat Chaudhuri. 2024. Putnambench: Evaluating neural theorem-provers on the putnam mathematical competition. *Preprint*, arXiv:2407.11214.

Yuhuai Wu, Albert Q. Jiang, Wenda Li, Markus N. Rabe, Charles Staats, Mateja Jamnik, and Christian Szegedy. 2022. Autoformalization with large language models. *Preprint*, arXiv:2205.12615.

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. *arXiv preprint arXiv:2405.14333*.

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

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

A Appendix

A.1 Prompt Example

We experimented with various prompting techniques to optimize the model performance. The prompt below proved to be the most effective, it is the one we use in the baseline. Other methods use very similar prompting, adding the hypothesis decomposition or retrieved snippets.

```
"You are an expert in formalizing mathematical
statements in Lean 4. Given the following informal
mathematical statement, write the corresponding formal
statement in Lean 4 syntax.
Output format: The translated LEAN 4 theorem should be
provided as a single cohesive code block, displaying
the correct syntax and formatting expected for a LEAN
4 theorem statement. Do not enclose the code block in
backticks. Write sorry as the proof.

Some examples:
Informal statement:
<FEWSHOT_1_INFORMAL>
Formal statement in Lean 4:
<FEWSHOT_1_FORMAL>
...
```

```
Now it's your turn:
Informal statement:
<INFORMAL>
Formal statement in Lean 4:"
```

A.2 Complimentarity Analysis on Type Checking

We also examined the complementarity of methods using the type check metric.

As seen in Figure 5, the trends observed largely align with those identified for BEq, reinforcing the idea that complementarity scores capture meaningful patterns in method overlap. For this reason, we opted to present only the BEq results in the main body of the report to avoid redundancy.

A.3 Abstract Algebra Experiment

To further test our approaches, we created a smaller dataset focusing on abstract algebra statements. This dataset was intentionally curated to include more complex problems, involving intricate types and definitions, with the goal of evaluating the performance of our methods on harder problems.

Unfortunately, our experiments yielded an average BEq score of 0 across all methods, making it difficult to assess the relative impact of different approaches. We attribute this result to the increased complexity of the statements, which often involve longer formal expressions and more sophisticated type dependencies. These factors exacerbate the strictness of the BEq metric and its tendency to over-reject.

The poor performance highlights a key limitation of BEq as an evaluation metric for statements of greater length or complexity.

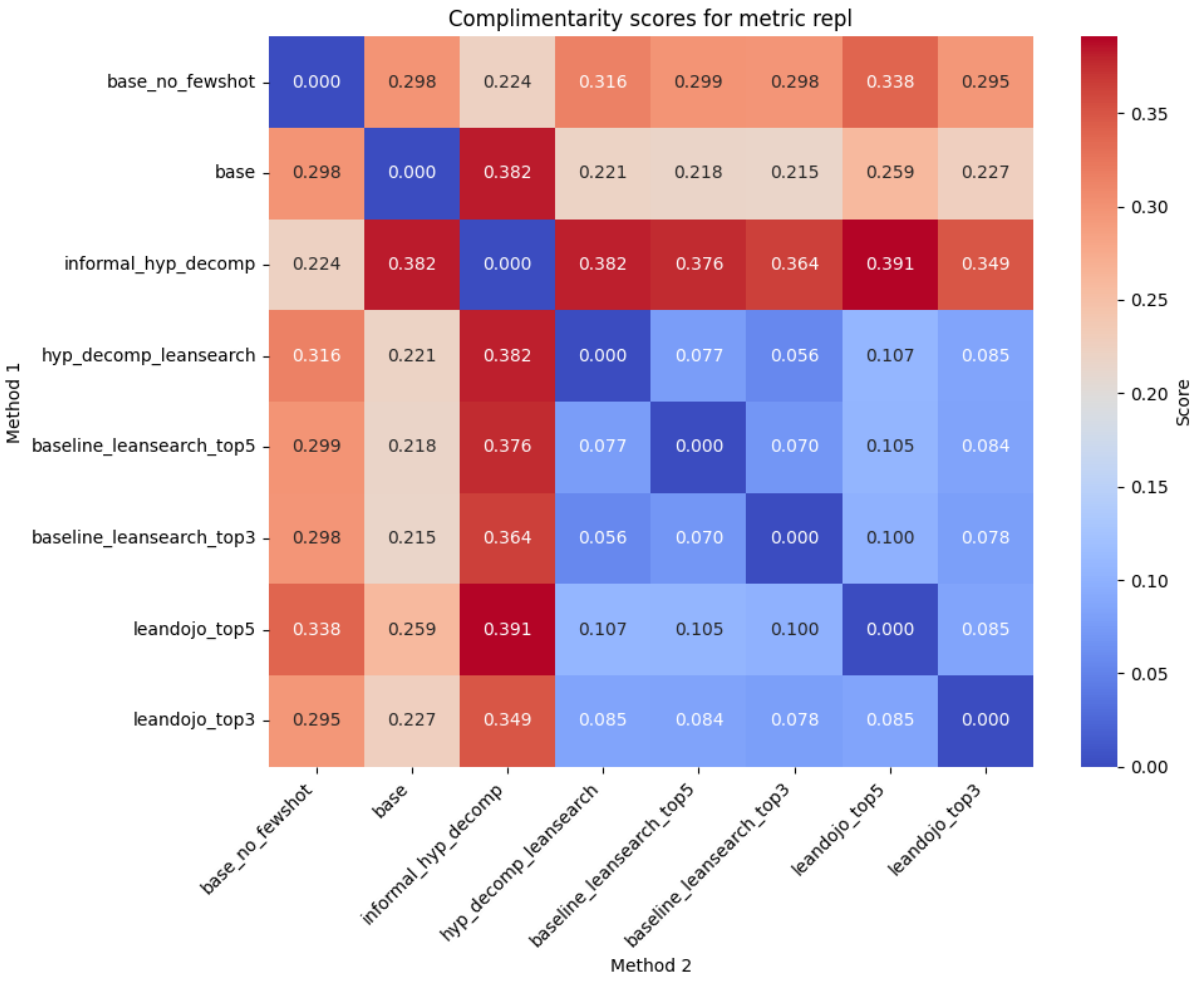

Figure 5: Complimentarity of our methods, according to uniqueness of type checking successes