MATHRHS: CONCLUSION SYNTHESIS IN THE END-TO-END THEOREM-PROVING PIPELINE

Anonymous authors
Paper under double-blind review

ABSTRACT

Many mathematical problems require synthesising a conclusion before a complete theorem statement can be formalised and proved. Evaluations that provide this conclusion in advance skip an important subtask: determining what must be proved. In this study we investigate *where conclusion synthesis should occur in the end-to-end theorem-proving pipeline*. We introduce MathRHS, a benchmark of 457 mathematical problems with conclusions (right-hand side) that separates conclusions from problem statements, and compare three settings: standalone conclusion synthesis before autoformalisation, joint synthesis of the conclusion and formal statement, and conclusion synthesis during proving. By evaluating conclusion correctness, formalisation faithfulness, and verified proof success separately, we trace how synthesis performance translates into downstream results. Standalone synthesis substantially improves conclusion accuracy over joint synthesis: from 61.27% to 75.22% for GPT-4.1 and from 45.73% to 82.02% for DeepSeek-V3.2. These improvements also benefit subsequent stages: using DeepSeek-V3.2 for both standalone synthesis and autoformalisation increases faithful autoformalisation from 12.28% to 21.49% and verified proof success from 4.38% to 5.48%. These findings establish conclusion synthesis as an important pipeline design choice: separating it from autoformalisation improves conclusion accuracy and can strengthen downstream prover performance. Nevertheless, the strongest evaluated pipeline proves only 6.36% of problems, highlighting the challenge of integrating correct conclusions through faithful formalisation to verified proofs. We argue that more effort should be put into targeted training for conclusion synthesis to generate more successful end-to-end proofs.

1 INTRODUCTION

Formal mathematical reasoning with Large Language Models (LLM) combined with proof assistants such as Isabelle (Wenzel et al., 2008) and Lean (Moura & Ullrich, 2021) has emerged as a powerful paradigm for solving complex mathematical problems with machine-verifiable correctness guarantees. At the 2026 IMO, multiple AI systems solved all six problems, with some systems producing machine-verifiable Lean proofs (Axiom Math, 2026). The theorem proving pipeline employed by these systems is commonly divided into two stages: *autoformalisation*, which translates a natural language problem (or informal problem) into a formal statement (Szegedy, 2020), and *theorem proving*, which uses series of tactics to generate a proof to the formal statement (Polu & Sutskever, 2020). For many mathematical problems, however, the formal statement itself depends on a conclusion that must first be synthesised. Problems may ask for a numerical value, an algebraic expression, an optimal bound, or a set of solutions (Pauli, 2022).

In practice, existing autoformalisation and theorem-proving benchmarks provide the conclusion in the input statement (Zheng et al., 2021; Azerbayev et al., 2023). For example, a problem may specifically provide the conclusion in the informal problem description (e.g. "What are the roots of $x^2 - 4x$? The roots are 0 and 4") or by formulating the problem to assume the conclusion (e.g. "Show that the roots of $x^2 - 4x$ are 0 and 4"). However, providing the conclusion in advance therefore removes a substantive part of solving the original problem.

Alternative settings omit the explicit conclusion during autoformalisation and introduce placeholder variables, deferring conclusion synthesis entirely to the proof search stage, as seen in open-answer

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

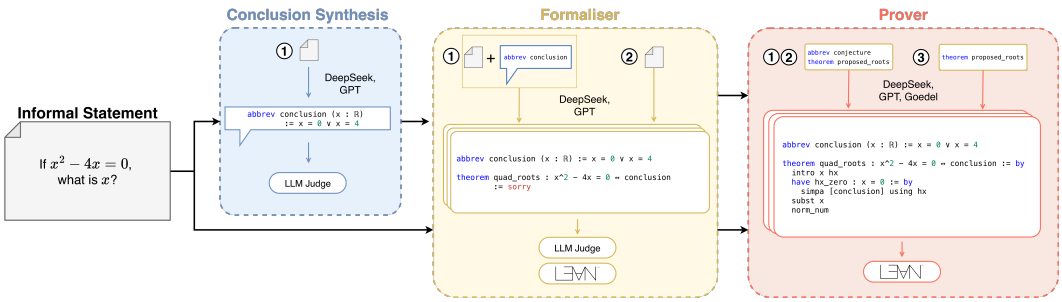


Figure 1: Overview of the end-to-end theorem-proving pipeline evaluated in MathRHS and the three conclusion synthesis settings: ① standalone conclusion synthesis before formalisation, ② conclusion synthesis during autoformalisation, and ③ conclusion synthesis during proving. Conclusion correctness is evaluated with an LLM-as-a-Judge, formalisation faithfulness requires both Lean typechecking and a faithfulness judge, and proof success is determined by Lean verification.

subsets of PutnamBench (Tsoukalas et al., 2024) and CombiBench (Liu et al., 2025). However, relying on the prover to simultaneously synthesise and self-verify the conclusion introduces important caveats. Most notably, formal proof verification alone does not guarantee answer completeness or problem faithfulness. For example, given the problem "If $x^2 - 4x = 0$, what is x?" an in-proof search that verifies $x = 0$ could yield a fully verifiable Lean proof for an incomplete solution set, omitting the valid root, $x = 4$. Evaluating formal reasoning systems therefore requires disentangling conclusion correctness, formalisation faithfulness, and proof verification.

Moreover, although recent theorem-provers (MI Inc., 2026) have achieved impressive milestones such as solving all 672 formal problems in PutnamBench[1] and advancing on CombiBench[2], these evaluations restrict conclusion discovery to in-proof placeholder instantiation starting from gold formal statements. Consequently, they omit comparisons against pipelines where conclusion synthesis is performed prior to or jointly with autoformalisation.

In this study, we investigate where conclusion synthesis should occur in the theorem-proving pipeline, as illustrated in Figure 1. Starting from an informal mathematical problem, the pipeline must produce a conclusion, formalise the problem and conclusion as a formal statement, and construct a proof that Lean can verify. We compare three ways to organise these tasks. In *standalone conclusion synthesis* (setting ①), conclusion synthesis first generates the conclusion, which is then supplied to the formaliser together with the informal problem; the resulting statement is passed to the prover. In *conclusion synthesis during autoformalisation* (setting ②), the formaliser generates the conclusion and formal statement jointly before passing the statement to the prover. In *conclusion synthesis during proving* (setting ③), the formal statement contains a conclusion placeholder that the prover instantiates while constructing a proof. We evaluate this final setting using gold formal statements to isolate conclusion synthesis during proving from autoformalisation errors.

To support this comparison, we introduce MathRHS,[3] a benchmark of 457 mathematical problems that separates conclusions from problem statements, making conclusion synthesis an explicit evaluation target. Our evaluation distinguishes three criteria: *conclusion correctness*, whether the generated conclusion correctly and completely addresses the problem; *formalisation faithfulness*, whether the formal statement preserves the problem formulation such all constraints included in the informal statement; and *proof success*, whether Lean verifies the completed proof. This distinction allows us to identify where successful conclusion synthesis translates into downstream performance and where further performance bottlenecks remain.

Our results show that isolating conclusion synthesis from autoformalisation substantially improves conclusion accuracy across all stages. Standalone conclusion synthesis achieves 75.22% conclusion accuracy with GPT-4.1 and 82.02% with DeepSeek-V3.2, compared with 61.27% and 45.73%, respectively, when conclusion synthesis is performed jointly during autoformalisation. These gains also extend downstream. For GPT-4.1, standalone conclusion synthesis increases faithful formalisa-

[1] https://trishullab.github.io/PutnamBench/leaderboard
[2] https://moonshotai.github.io/CombiBench/leaderboard
[3] Available at: https://anonymous.4open.science/r/ConjectureBench-123E

tion performance from 11.81% to 13.60% and proof success from 4.81% to 6.36%. For DeepSeek-V3.2, formalisation performance increases from 12.28% to 21.49% and proof success from 4.38% to 5.48%. The results show that separating conclusion synthesis from formalisation can improve both formalisation quality and downstream proof success.

Our contributions are threefold. First, we introduce MathRHS, a benchmark that makes conclusion synthesis an explicit evaluation target in formal theorem proving. Second, we compare three placements of conclusion synthesis (before autoformalisation, during autoformalisation, and during proving) and evaluate conclusion correctness, formalisation faithfulness, and proof success separately to locate where each pipeline fails. Third, we show that synthesising the conclusion before autoformalisation improves conclusion accuracy by up to 36 percentage points and leads to more faithful formalisations and more verified proofs.

2 PRELIMINARY

In mathematics, a statement establishes a relation where hypotheses (left-hand side, LHS) lead to a conclusion (right-hand side, RHS). While existing theorem-proving benchmarks often assume a known conclusion or instantiate it during the proof search, open-answer problem solving requires synthesising this conclusion first. Conflating conclusion synthesis and proof construction is misleading as they draw on distinct reasoning mechanisms: synthesising a conclusion relies on intuition, pattern recognition, and heuristic search, whereas proving demands the rigorous application of deductive rules or tactics (Fernández-León et al., 2021). In this work, we focus on *conclusion synthesis*, the automated generation of target answers, whether numerical values, algebraic expressions, or binary decisions, as a separate task in formal mathematical reasoning.

To illustrate the importance of faithfulness verification with conclusion synthesis during proof and to also demonstrate why formal proof verification alone cannot guarantee conclusion correctness, consider the problem: "*If* $x^2 - 4x = 0$*, what is* x*?*" The complete answer set is $\{0, 4\}$. In Lean 4, a faithful formalisation should express not only that 0 and 4 are roots, but that they are the *only* real numbers satisfying the equation. If the answer is synthesised upfront, it can be defined explicitly as:

Lean 4

```
abbrev conclusion (x : ℝ) := x = 0 ∨ x = 4
theorem quad_roots : x^2 - 4*x = 0 → conclusion := sorry
```

Alternatively, if conclusion synthesis is deferred to the proving stage, the explicit definition body is replaced with a typed placeholder: `abbrev conclusion (x : ℝ) := sorry`. In this setting, Lean treats `conclusion` as an uninstantiated placeholder that must be resolved during proof search. Consider the following three candidate conclusions:

Lean 4

```
abbrev conclusion_1 (x : ℝ) := x = 0
abbrev conclusion_2 (x : ℝ) := x = 4
abbrev conclusion_3 (x : ℝ) := x = 0 ∨ x = 4
```

While all three definitions yield mathematically valid directional implications that Lean can verify, `conclusion_1` and `conclusion_2` represent incomplete answers. Only `conclusion_3` provides the complete solution set. This distinction highlights why conclusion correctness and formaliser faithfulness must be evaluated independently of proof verification. Another possible `conclusion` could be `abbrev conclusion (x : ℝ) := x^3 - 4*x^2 = 0` which gives a trivial conclusion without answering the original question "what is x?" that is expected to be real-values.

Furthermore, the optimal modality for conclusion synthesis varies significantly across problem domains:

- **Natural Language Advantage:** Intuitive problems like "*How many people guarantee a shared birth month?*" are straightforward using intuitive logic, whereas their formalisation requires complex domain-size arguments over set injectivity.
- **Formal Statement Advantage:** Puzzle-like logic, such as "Alex and Blake always tell the truth or always lie. Alex says that at least one of them is a liar", contain language ambiguities that evaporate once formalised into symbolic hypotheses (`hAlex : Alex ⟺ (¬Alex ∨ ¬Blake)`).
- **In-Proof Kernel Advantage:** Computationally heavy tasks, such as finding $2^{100} \pmod{13}$, can bypass manual symbolic derivation entirely when left as placeholders for Lean's computational kernel to reduce definitionally via `rfl`.

These domain-dependent trade-offs indicate that conclusion synthesis can be integrated at different stages of an end-to-end reasoning pipeline: as standalone task, during autoformalisation, or during proving. Disentangling these modalities allows us to systematically determine the most effective paradigm for open-answer formal reasoning, motivating the three conjecturing settings evaluated in MathRHS.

3 METHODOLOGY

3.1 MATHRHS BENCHMARK

MathRHS is constructed from PutnamBench (Tsoukalas et al., 2024) and CombiBench (Liu et al., 2025), two formal mathematics benchmarks that contain problems for which an answer can be separated from the problem statement. We select 355 problems from PutnamBench and 102 problems from CombiBench, where multi-part questions are treated as separate problems, resulting in 457 problems in total.

Table 1: Examples of problems reformulated as open-answer problems in MathRHS, along with the overall dataset distribution across conclusion types.

Original with integrated conclusion	Reworded to seek a conclusion	Type of conclusion	Distribution
Show that there are at least 1991 red points in the plane.	What is the minimum number of red points in the plane?	Numerical	39.0% (178)
Prove that there are at most $2n-1$ subsets in the collection.	What is the maximum number of subsets that can be in such a collection?	Algebraic	36.1% (165)
Prove that B = $\{0, 3, 4, 9, 11\}$ is a difference set in Z_{21}.	Prove or disprove that B = $\{0, 3, 4, 9, 11\}$ is a difference set in Z_{21}.	Binary	24.9% (114)

To transform these benchmarks into open-answer tasks, we manually reviewed and rewritten each problem to ensure that the target conclusion is completely omitted from the informal statement. As shown in Table 1, problems are categorised into three distinct conclusion types based on their underlying conclusion structure: numerical, algebraic and binary. Numerical problems require a concrete value, algebraic problems require an expression that may depend on problem parameters, and binary problems require deciding whether a stated property holds or not.

For each problem, we construct a paired representation consisting of an informal statement, a ground-truth formal statement, and a reference conclusion. The reference conclusion and formal statement are decoupled and withheld from the model during evaluation, except under explicit oracle baseline conditions. This design allows the same problem benchmark to be evaluated consistently across all three conclusion synthesis settings independently of formalisation and proving.

3.2 CONCLUSION SYNTHESIS PLACEMENTS

We evaluate three positions of conclusion synthesis within the end-to-end theorem-proving pipeline, corresponding to the settings shown in Figure 1. In all settings operating directly on natural language, the ground-truth target answer is strictly withheld from the input prompt.

Standalone conclusion synthesis. In this stage, conclusion synthesis is completely decoupled from formalisation and proving. The model receives only the informal open-answer problem and synthesises a conclusion formatted as a concise Lean 4 expression. This conclusion is then passed to the autoformaliser alongside the original informal statement to produce a complete formal statement, which is subsequently delivered to the prover.

Conclusion synthesis during autoformalisation. Rather than relying on a standalone conjecture synthesis, in this setting, the formaliser receives the informal problem and generates the full formal statement, including the synthesised conclusion in a single autoformalisation pass. The resulting formal statement is then passed directly to the prover. This setting evaluates whether language models can implicitly synthesise conclusions while formalising problem constraints.

Conclusion synthesis during proving. In this setting, conclusion synthesis is deferred entirely to the prover. A ground-truth formal statement where the explicit conclusion is abstracted into a typed placeholder as described in Section 2 is passed to the prover. The prover must synthesise a valid instantiation for this placeholder while simultaneously constructing its formal proof.

Oracle. To isolate potential bottleneck stages across the end-to-end framework, we establish oracle control conditions. For autoformalisation, the oracle prompt includes the ground-truth answer alongside the informal statement, eliminating the need for upfront conclusion synthesis. For oracle proving, the prover is provided with the ground-truth formal statement with the reference conclusion. This measures prover capability independently of autoformalisation and conclusion synthesis output quality.

3.3 Evaluation

We evaluate three dimensions corresponding to the evaluation measures displayed in Figure 1: conclusion correctness, formalisation faithfulness, and proof verification.

Conclusion correctness. To evaluate whether a synthesised conclusion correctly solves the original informal problem, we use an LLM-based judge. The judge compares the generated answer against the reference conclusion to assess mathematical equivalence. When evaluating conclusions embedded directly inside formal statements or generated during proof search, the judge receives the surrounding formal statement to interpret the candidate answer in its proper functional role (e.g., distinguishing a target root value 2 from an exponent or array index 2). The LLM-based approach enables us to verify the synthesised conclusion at any stage of the pipeline. We validate the LLM judge against human expert annotations on 100 randomly sampled autoformalisation outputs, achieving 83% agreement. Further details are provided in Appendix A.

Formalisation faithfulness. A formal statement is evaluated via a two-stage filter. First, the statement must successfully typecheck in Lean, ensuring syntactic and type-level validity. This is necessary to pass the statement to the prover. Second, typechecked statements are evaluated by an LLM-based semantic grader, similar to Wang et al. (2025), to verify that they accurately capture the domain, hypotheses, conclusion, and completeness requirements of the original problem without over-constraining or trivialising the statement. A formalisation is counted as faithful if and only if it both typechecks in Lean 4 and passes semantic grading.

Proof verification. Proof correctness is verified deterministically by Lean. A proof is considered successful if and only if the Lean kernel accepts the complete theorem without open goals, unassigned metavariables, or incomplete proof terms. While proof verification establishes the internal logical validity of the formal theorem, the preceding faithfulness evaluation ensures that the verified theorem actually solves the intended informal problem.

4 Experimental Setup

Models and configurations. We evaluate GPT-4.1 (Achiam et al., 2023) and DeepSeek-V3.2 (DeepSeek-AI et al., 2024) for conclusion synthesis and autoformalisation, with Goedel-Prover-V2 (Lin et al., 2025) as the prover in every configurations. For standalone conclusion synthesis, we evaluate all four conclusion synthesis–autoformaliser pairings: conclusion synthesis by

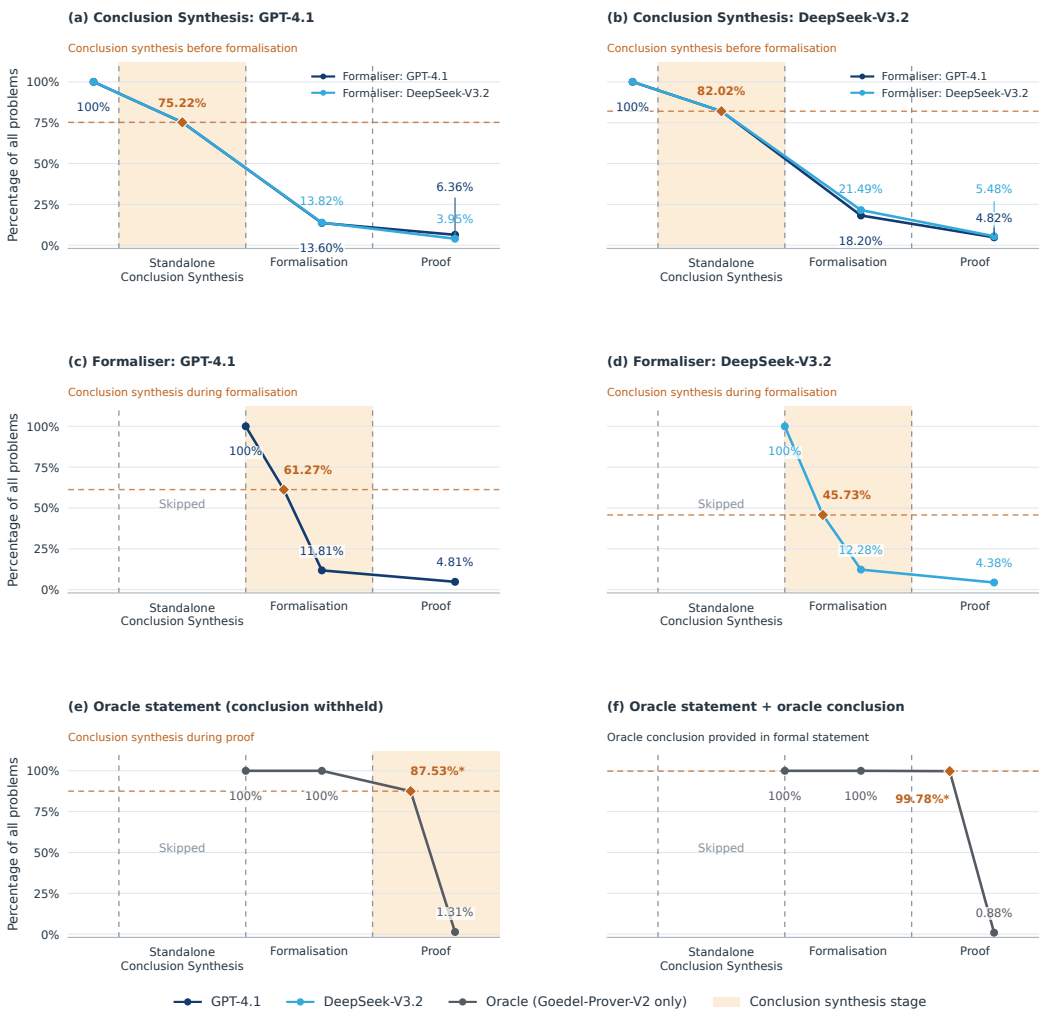


Figure 2: End-to-end conclusion synthesis across the evaluated pipeline configurations. Panels (a)–(b) show all four standalone concluion synthesis–autoformaliser pairings, panels (c)–(d) show joint conclusion synthesis and autoformalisation, and panels (e)–(f) use gold formal statements with the answer withheld or provided, respectively. All configurations use Goedel-Prover-V2 for proving. Shaded region indicates the conjecture synthesis stage and orange diamonds indicate conclusion correctness, evaluated by an LLM-as-a-Judge. Formalisation faithfulness requires passing both typecheck and faithfulness. Proof success requires typecheck of the completed proof. Asterisks indicate conclusion correctness evaluated after proof generation, including outputs without verified proofs. All percentages are calculated over the full evaluation set.

either model are supplied to each autoformaliser together with the informal problem shown in panel (a) and (b) of Figure 2. For conclusion synthesis during autoformalisation, we evaluate each model separately, generating the conclusion and formal statement jointly, shown in panels (c) and (d). For conjecturing during proving, we provide Goedel-Prover-V2 with the gold formal statement, replacing the conclusion with a placeholder. As a control, we also supply the gold statement with the reference conclusion provided. These two oracle conditions, shown in panels (e) and (f), isolate proving from autoformalisation errors, assuming 100% formalisation faithfulness by construction.

Figure 2 reports conclusion correctness, formalisation faithfulness, and verified proof success as percentages of the full evaluation set. At each stage, a problem counts as successful if at least one eligible candidate satisfies the relevant criterion. Conclusion correctness is assessed at the stage where the conclusion synthesis occurs (shaded region in Figure 2); in oracle conditions, conclusions are evaluated post-hoc after proof generation, including outputs whose proofs fail verification.

For standalone conclusion synthesis, we generate 10 candidate conclusions per problem and evaluate each with an LLM-as-a-judge. Only accepted conclusions proceed to autoformalisation, where the autoformalisers generate a further 10 candidate formal statements for each distinct accepted conclusions. For joint conclusion synthesis and autoformalisation, we instead generate 10 candidate statements per problem without prior filtering. In both settings, generated statements proceed to the prover only if they pass Lean typechecking and semantic faithfulness grading. We allocate up to 32 proof attempts per candidate statement, with the same for each gold statement in the oracle conditions.

5 Results and Discussion

Separating conclusion synthesis improves conclusion accuracy. Figure 2 displays the conclusion synthesis accuracy with an orange diamond and compares conclusion synthesis before autoformalisation, during autoformalisation, and during proving. Under the LLM-as-a-Judge evaluation, GPT-4.1 achieves 75.22% accuracy with standalone conclusion synthesis, compared with 61.27% when generating the conclusion and formal statement jointly. For DeepSeek-V3.2, the corresponding scores are 82.02% and 45.73%. Separating conclusion synthesis from autoformalisation therefore improves conclusion accuracy by 13.95 and 36.29 percentage points, respectively.

Conclusion synthesis during proving achieves 87.53% accuracy, but the prover receives a gold formal statement with a conclusion placeholder. This result shows that the prover can often generate an accepted conclusion when the formal representation is already available. It does not establish that deferring conclusion synthesis to proving is more effective in an end-to-end pipeline, because this condition bypasses autoformalisation. Conclusion accuracy is also evaluated independently of proof validity: although 87.53% of problems have an accepted conclusion, only 1.31% have a valid proof.

Standalone conclusion synthesis improves faithful autoformalisation. Separating conclusion synthesis from autoformalisation improves formalisation performance for both models. We measure faithful autoformalisation performance as the percentage of all evaluated problems with at least one generated statement that passes both Lean typecheck and the LLM-as-a-Judge assessment of semantic faithfulness. From the Formalisation segment of panel (a)–(c), we observe that for the GPT-4.1 autoformaliser, performance increases from 11.81% in the joint setting to 13.60% with conclusions generated by GPT-4.1 and 18.20% with conclusions generated by DeepSeek-V3.2. For the DeepSeek-V3.2 autoformaliser, comparing panel (d) with (a) to (b), performance increases from 12.28% to 13.82% and 21.49%, respectively. Conclusions generated by DeepSeek-V3.2 yield higher autoformalisation performance with both models, consistent with its stronger standalone conclusion accuracy. Nevertheless, the gap between accepted conclusions and faithful formalisations remains substantial. In the same-model pipelines, out of the 61.27% problems with accepted GPT-4.1 conclusions only 19.28% progress to faithful formalisation, likewise only 26.85% make the same progression with DeepSeek-V3.2 conclusions. Standalone conclusion synthesis helps, but integrating the conclusion in a faithful formal statement remains challenging.

Upstream improvements do not consistently translate into proof success. Using the same model for standalone conclusion synthesis and autoformalisation improves proof success relative to conclusion synthesis during autoformalisation: from 4.81% to 6.36% for GPT-4.1 as shown from panels (c) to (a), and from 4.38% to 5.48% for DeepSeek-V3.2, shown in panels (d) and (b). In this case, proof success denotes the percentage of all evaluated problems for which the theorem-proving pipeline produces a verified proof after the preceding selection and faithfulness checks. However, the pipeline with the highest autoformalisation performance (i.e. using DeepSeek-V3.2) does not achieve the highest proof success. The DeepSeek-V3.2 $\rightarrow$ DeepSeek-V3.2 setting (seen in panel (b)) produces faithful formalisations for 21.49% of problems, compared with 13.60% for GPT-4.1 $\rightarrow$ GPT-4.1 (see panel (a)), yet the latter achieves higher proof success: 6.36% versus 5.48%.

Analysing conditional proof success provides a complementary view. Among problems reaching faithful formalisation, 46.76% receive a verified proof in the GPT-4.1 $\rightarrow$ GPT-4.1 pipeline, compared with 25.50% in the DeepSeek-V3.2 $\rightarrow$ DeepSeek-V3.2 pipeline. These rates concern different problem sets and formal statements, so they do not isolate formaliser quality or prover ability.

Table 2: Number of problems with at least one valid conclusion, by solution type, evaluated using LLM-as-a-Judge at pass@10. We compare standalone conclusion synthesis in the MODEL $\rightarrow$ MODEL conclusion synthesis–autoformalisation pipeline with joint conclusion synthesis and autoformalisation using the same MODEL. Standalone conclusions are evaluated before autoformalisation.

Model	Setting	Numerical (178)	Algebraic (165)	Binary (114)
GPT-4.1	Standalone	79.8% (142)	66.1% (109)	81.6% (93)
	During autoformalisation	57.9% (103)	52.1% (86)	79.8% (91)
DeepSeek-V3.2	Standalone	88.8% (158)	73.3% (121)	83.3% (95)
	During autoformalisation	39.9% (71)	34.5% (57)	71.1% (81)

Conclusion accuracy improves amongst all conclusion types. Table 2 shows that standalone conclusion synthesis improves GPT-4.1 and DeepSeek-V3.2 conclusion accuracy across all three conclusion types. Numerical, algebraic and binary accuracy respectively increase by 21.35, 13.94, and 1.75 percentage points for GPT-4.1 and 48.9, 38.8, 12.2 points for DeepSeek-V3.2. The benefit of separating conclusion synthesis from autoformalisation therefore extends across all categories, with the largest improvements on numerical and algebraic problems. Binary conclusions are on average the least successful, this is possibly because the task of deciding whether a statement is true or false is not common in standard benchmarks, and therefore the model is not used to the formatting of the problem.

The results support conclusion synthesis as an explicit pipeline stage: separating it from autoformalisation improves conclusion accuracy and faithful autoformalisation performance for both models, and improves proof success in the same-model pipelines. Even the strongest evaluated pipeline produces verified proofs for only 6.36% of problems. The remaining challenge is therefore to connect correct conclusion synthesis with faithful autoformalisation and successful proof generation. This motivates investigating targeted training for conclusion synthesis while assessing progress through all three criteria: conclusion correctness, formalisation faithfulness, and proof success.

6 Related Work

In the literature, synthesising conclusions is usually interleaved with proving by proposing placeholder conclusions that are subsequently validated during tactic search (Dong & Ma, 2025). Sun et al. (2025) extend this concept by iteratively instantiating numerical special cases from an autoformalised statement to form candidate conclusions in a closed loop. Additionally, Zhou et al. (2024) demonstrate that for elementary domains, LLMs can generate solutions for autoformalisation tools to verify. Crucially, however, these works treat conclusion synthesis merely as an unisolated heuristic embedded within broader proof search. Our work systematically isolates conclusion synthesis as an independent capability across distinct operational stages, we also establish dedicated metrics to ensure that the synthesised statements faithfully resolve the original open-ended problem, as well as the correctness of the conclusion.

An adjacent line of research focuses on incorporating external guidance signals to improve autoformalisation performance. Existing pipelines leverage retrieved library premises (Zhang et al., 2026), LLM-as-a-judge filtering (Wang et al., 2025), chain-of-thought scoring (Xin et al., 2024), interactive Lean compiler feedback (Lu et al., 2024; Sun et al., 2025), or iterative LLM critiquing (Peng et al., 2025). While these methodologies demonstrate the value of auxiliary context for autoformalisation, they are fundamentally orthogonal to our focus: they assume the target theorem is fully specified or recoverable from the text, addressing translation fidelity rather than the synthesis of missing conclusions. Nonetheless, they provide some insight of how to integrate external information within the formal reasoning pipeline.

Similarly, research in formal proof search explores augmenting neural provers with auxiliary structural priors. Existing provers steer tactic execution using informal proof sketches (Jiang et al., 2023), modular subgoals (Wang et al., 2024), process-supervised reward signals (Xin et al., 2024), or counterexample search to prune invalid branches (Dong & Ma, 2025). Although these techniques provide

procedural guidance during proof execution, they address a distinct problem setting: they uniformly presuppose a fully specified formal goal. Conclusion synthesis provides a fundamentally different form of prior.

7 Conclusion

In this work, we identify conclusion synthesis as an overlooked step in end-to-end formal mathematical reasoning for open-answer problems with LLMs. We investigate the fundamental question: *where in the formal reasoning pipeline should conclusion synthesis occur?* To answer this, we introduce MathRHS, a benchmark of 457 open-answer problems designed to evaluate conclusion synthesis across three distinct stages: as a standalone task, jointly during autoformalisation, and during formal proof generation. We evaluate each pipeline stages across three decoupled metrics: conclusion correctness, formalisation faithfulness, and proof verification.

Our empirical findings demonstrate that decoupling conclusion synthesis from formalisation and proving yields substantial performance gains. Synthesising answers separately prior to autoformalisation increases conclusion correctness by an average of 25.12% for GPT-4.1 and 36.29% for DeepSeek-V3.2 compared to joint generation. Feeding these explicit conclusions to downstream autoformalisation and theorem-proving components further improves performance across individual pipeline tasks by at least 25%. Nevertheless, even under the optimal pipeline configuration, 93% of benchmark problems remain unproven, underscoring the challenge open-answer formal reasoning poses to LLMs.

For future work, we believe that advancing formal mathematical reasoning on open-answer problems requires treating conclusion synthesis as an independent task. This calls for richer benchmarks that pair informal problems with explicit target answers across all pipeline stages, specialised inference-time search techniques for conclusion synthesis, and training strategies that effectively synthesise and integrate conclusions into downstream formalisation and proof construction.

AI Use Statement

In this work, we used generative AI tools for implementing research methods (specifically using language models as judges to assess candidate conclusion correctness and semantic formalisation faithfulness). We have not used generative AI tools for generating synthetic data sets, developing theoretical models, formulating mathematical claims, providing critical ingredients for proving mathematical claims, assisting in writing proofs, proposing or refining hypotheses, or designing research methodology, and the remaining required disclosure tasks are not applicable to this work. Additionally, we used generative AI tools for creating or modifying scientific figures, creating or editing software code, and editing the manuscript to improve readability.

We have thoroughly reviewed all AI-assisted work: LLM-as-a-judge outputs were validated against human expert annotations (achieving 83% agreement), AI-generated code and figures were verified and tested for correctness by the authors, and all text revisions were manually reviewed and approved. We take full responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. GPT-4 Technical Report. *arXiv preprint arXiv:2303.08774*, 2023. URL https://arxiv.org/pdf/2303.08774.

Axiom Math. AxiomProver at imo 2026. https://github.com/AxiomMath/IMO2026, 2026. [Accessed 22-09-2026].

Zhangir Azerbayev, Bartosz Piotrowski, Hailey Schoelkopf, Edward W. Ayers, Dragomir Radev, and Jeremy Avigad. ProofNet: Autoformalizing and Formally Proving Undergraduate-Level Mathematics. 2023. URL https://arxiv.org/abs/2302.12433.

DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Haowei Zhang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Li, Hui Qu, J. L. Cai, Jian Liang, Jianzhong Guo, Jiaqi Ni, Jiashi Li, Jiawei Wang, Jin Chen, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, Junxiao Song, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Xu, Leyi Xia, Liang Zhao, Litong Wang, Liyue Zhang, Meng Li, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Ning Tian, Panpan Huang, Peiyi Wang, Peng Zhang, Qiancheng Wang, Qihao Zhu, Qinyu Chen, Qiushi Du, R. J. Chen, R. L. Jin, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runxin Xu, Ruoyu Zhang, Ruyi Chen, S. S. Li, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaoqing Wu, Shengfeng Ye, Shengfeng Ye, Shirong Ma, Shiyu Wang, Shuang Zhou, Shuiping Yu, Shunfeng Zhou, Shuting Pan, T. Wang, Tao Yun, Tian Pei, Tianyu Sun, W. L. Xiao, Wangding Zeng, Wanjia Zhao, Wei An, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, X. Q. Li, Xiangyue Jin, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaojin Shen, Xiaokang Chen, Xiaokang Zhang, Xiaosha Chen, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xingkai Yu, Xinnan Song, Xinxia Shan, Xinyi Zhou, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, Y. K. Li, Y. Q. Wang, Y. X. Wei, Y. X. Zhu, Yang Zhang, Yanhong Xu, Yanhong Xu, Yanping Huang, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Li, Yaohui Wang, Yi Yu, Yi Zheng, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Ying Tang, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yu Wu, Yuan Ou, Yuchen Zhu, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yukun Zha, Yunfan Xiong, Yunxian Ma, Yuting Yan, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Z. F. Wu, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhen Huang, Zhen Zhang, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhibin Gou, Zhicheng Ma, Zhigang Yan, Zhihong Shao, Zhipeng Xu, Zhiyu Wu, Zhongyu Zhang, Zhuoshu Li, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Ziyi Gao, and Zizheng Pan. Deepseek- v3 Technical Report. *arXiv preprint arXiv:2412.19437*, 2024.

Kefan Dong and Tengyu Ma. STP: Self-play LLM theorem provers with iterative conjecturing and proving. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=zWArMedNuW.

Aurora Fernández-León, José María Gavilán-Izquierdo, and Rocío Toscano. A case study of the practices of conjecturing and proving of research mathematicians. *International Journal of Mathematical Education in Science and Technology*, 52(5):767–781, 2021. doi: 10.1080/0020739X.2020.1717658. URL https://doi.org/10.1080/0020739X.2020.1717658.

Albert Qiaochu Jiang, Sean Welleck, Jin Peng Zhou, Timothee Lacroix, Jiacheng Liu, Wenda Li, Mateja Jamnik, Guillaume Lample, and Yuhuai Wu. Draft, Sketch, and Prove: Guiding Formal Theorem Provers with Informal Proofs. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=SMa9EAovKMC.

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

J Liu, X Lin, J Bayer, Y Dillies, W Jiang, X Liang, R Soletskyi, H Wang, Y Xie, B Xiong, et al. Combibench: Benchmarking llm capability for combinatorial mathematics. *arXiv preprint arXiv:2505.03171*, 2025. URL https://arxiv.org/pdf/2505.03171.

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

MI Inc. Midas prover solves PutnamBench. https://mi.inc/blog/midas-prover-solves-putnambench, 2026. Accessed: 2026-09-26.

Leonardo de Moura and Sebastian Ullrich. The Lean 4 theorem prover and programming language. In *Automated Deduction–CADE 28: 28th International Conference on Automated Deduction, Virtual Event, July 12–15, 2021, Proceedings 28*, pp. 625–635. Springer, 2021.

Sebastian Pauli. Mat 112 integers and modern applications for the uninitiated. *Caesar Ciphers*, 2022. URL https://mat112.uncg.edu/HTML/root-1-2.html.

Zhongyuan Peng, Yifan Yao, Kaijing Ma, Shuyue Guo, Yizhe Li, Yichi Zhang, Chenchen Zhang, Yifan Zhang, Zhouliang Yu, Luming Li, et al. Criticlean: Critic-guided reinforcement learning for mathematical formalization. *arXiv preprint arXiv:2507.06181*, 2025. URL https://arxiv.org/pdf/2507.06181.

Stanislas Polu and Ilya Sutskever. Generative language modeling for automated theorem proving. 2020. URL https://arxiv.org/abs/2009.03393.

Jialiang Sun, Yuzhi Tang, Ao Li, Chris J Maddison, and Kuldeep S Meel. Enumerate-Conjecture-Prove: Formally Solving Answer-Construction Problems in Math Competitions. *arXiv preprint arXiv:2505.18492*, 2025. URL https://arxiv.org/abs/2505.18492.

Christian Szegedy. A Promising Path Towards Autoformalization and General Artificial Intelligence. In *Intelligent Computer Mathematics: 13th International Conference, CICM 2020, Bertinoro, Italy, July 26–31, 2020, Proceedings*, pp. 3–20, Berlin, Heidelberg, 2020. Springer-Verlag. ISBN 978-3-030-53517-9. doi: 10.1007/978-3-030-53518-6_1. URL https://doi.org/10.1007/978-3-030-53518-6_1.

George Tsoukalas, Jasper Lee, John Jennings, Jimmy Xin, Michelle Ding, Michael Jennings, Amitayush Thakur, and Swarat Chaudhuri. PutnamBench: Evaluating Neural Theorem-Provers on the Putnam Mathematical Competition. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 11545–11569. Curran Associates, Inc., 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/1582eaf9e0cf349e1e5a6ee453100aa1-Paper-Datasets_and_Benchmarks_Track.pdf.

Haiming Wang, Huajian Xin, Chuanyang Zheng, Zhengying Liu, Qingxing Cao, Yinya Huang, Jing Xiong, Han Shi, Enze Xie, Jian Yin, Zhenguo Li, and Xiaodan Liang. LEGO-prover: Neural theorem proving with growing libraries. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=3f5PALef5B.

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, Jiawei Liu, Jonas Bayer, Julien Michel, Longhui Yu, Léo Dreyfus-Schmidt, Lewis Tunstall, Luigi Pagani, Moreira Machado, Pauline Bourigault, Ran Wang, Stanislas Polu, Thibaut Barroyer, Wen-Ding Li, Yazhe Niu, Yann Fleureau, Yangyang Hu, Zhouliang Yu, Zihan Wang, Zhilin Yang, Zhengying Liu, and Jia Li. Kimina-prover preview: Towards large formal reasoning models with reinforcement learning. *arXiv preprint arXiv:2504.11354*, 2025. URL https://arxiv.org/abs/2504.11354.

Makarius Wenzel, Lawrence C. Paulson, and Tobias Nipkow. The isabelle framework. In Otmane Ait Mohamed, César Muñoz, and Sofiène Tahar (eds.), *Theorem Proving in Higher Order Logics*, pp. 33–38, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. ISBN 978-3-540-71067-7.

Huajian Xin, Daya Guo, Zhihong Shao, Z.Z. Ren, Qihao Zhu, Bo Liu, Chong Ruan, Wenda Li, and Xiaodan Liang. Advancing theorem proving in LLMs through large-scale synthetic data. In *The 4th Workshop on Mathematical Reasoning and AI at NeurIPS'24*, 2024. URL https://openreview.net/forum?id=TPtXLihkny.

Meiru Zhang, Philipp Borchert, Milan Gritta, and Gerasimos Lampouras. DRIFT: Decompose, retrieve, illustrate, then formalize theorems. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=IGEs577RFz.

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

Jin Peng Zhou, Charles Staats, Wenda Li, Christian Szegedy, Kilian Q. Weinberger, and Yuhuai Wu. Don't Trust: Verify - Grounding LLM Quantitative Reasoning with Autoformalization. In *ICLR*, 2024. URL https://openreview.net/forum?id=V5tdi14ple.

APPENDIX

A METRICS

We use the following three metrics to verify the outputs of the end-to-end theorem proving pipeline:

- **Conclusion Correctness** evaluates whether a conclusion appears in the output statement. Model selection was based on human alignment, reaching up to 83% agreement with humans. Table 3 shows the distribution of True and False presences of conclusions over 100 samples. Table 4 shows the human-model agreement for LLM-based judge.
- **Faithfulness Grader**: A pipeline evaluating semantic alignment. Formalisations are back-translated using InternLM2-Math-Plus-20B. A separate Qwen3-14B judge, calibrated to 67.5% against human annotators, evaluates the natural language statements.
- **Typecheck**: This binary measure of syntactic correctness indicates whether the generated Lean code compiles without error within a standardised environment (`v4.24.0`). This is used to verify whether the formalisation statement is valid in Lean, and also as **Proof Success** for checking the correctness of proofs.

	TRUE	**FALSE**	**Total**
Provided	35	11	46
Absent	21	33	54
Total	56	44	100

Table 3: Contingency table for provided and absent instances.

Model	**Percentage**
internlm2-math-plus-20b	60
qwen3-14b	79
gpt-oss-20b	70
qwen3-30b-a3b-instruct	**83**

Table 4: Model alignment to human annotators.

A.1 Conclusion Correctness Prompt

The prompt used to generate the conclusion correctness' LLM-based judge is given in Figure 3.

Conclusion Correctness Evaluation Prompt

System Prompt

```
You are an expert in the Lean 4 theorem proving language and formal
mathematics. Your task is to determine if a given formal statement in
Lean 4 contains a specific conjectured value, algebraic formula, or bound.
You will be given three inputs:
1. **Conjecture**: The value, formula, or bound to look for.
2. **Ground Truth Formal Statement**: An example of a Lean 4 statement that
correctly formalizes the conjecture. Use this as a reference for a valid
implementation.
3. **Formal Statement**: The Lean 4 code you need to evaluate.
Your goal is to determine if the **Formal Statement** contains the core
assertion of the **Conjecture**. The **Ground Truth Formal Statement** is
provided to help you understand how the conjecture can be formally expressed.
The statement you are evaluating might not have the exact same syntax as the
ground truth. You must carefully check for **semantically equivalent
variations** of the conjecture's core idea. This includes, but is not limited
to, permutations of terms, different but equivalent algebraic expressions, or
reordered hypotheses. Additionally, a conjecture can be expressed either by
defining a proposition (e.g., `abbrev conjecture : Prop := ...`) or by
asserting it within a theorem, which implicitly states the conjecture holds.
You should consider these forms equivalent.
Your output must follow this structure exactly:
1. First, provide a brief explanation of your reasoning.
2. Second, conclude with the final answer in the format: 'The formal
statement contains the conjecture: **True**' or 'The formal statement
contains the conjecture: **False**'.
```

User Prompt

```
**Conjecture:**
```lean
{{ conjecture }}
```
Ground Truth Formal Statement:
```lean
{{ statement1 }}
```
Formal Statement:
```lean
{{ statement2 }}
```
```

Figure 3: Jinja templates for the system and user prompts used by LLM-based judge for conclusion correctness.

B Details on Experimental Setup

Experiments were conducted in Lean `v4.24.0`. Each instance was run for 10 passes using ten fixed random seeds to ensure reproducibility. We utilised a generation temperature of 0.7, with all other parameters kept at default for standalone conclusion synthesis and during autoformalisation. For proving, we use Goedel-Prover-V2 with pass@32. Below are all prompts used for different LLMs:

B.1 Standalone Conclusion Synthesis Prompt

The prompt used to generate the standalone conclusion is given in Figure 4.

Conjecturing Prompt

System Prompt

```
You are an advanced assistant specializing in formal mathematics and Lean 4
theorem proving. You have extensive expertise in translating mathematical
concepts from natural language into precise Lean 4 code.
You do not provide proofs or full theorem statements, only the mathematical
expression representing the solution, proposition, or the value being asserted.
You should first analyze the informal problem statement, then provide the final
expression as valid Lean 4 code.
```

User Prompt

```
Your task is to take a natural language mathematical statement and extract the
mathematical expression, proposition, or value, representing it as a Lean 4
expression.
**Instructions:**
1. Analyze the informal problem statement to deconstruct its mathematical components.
2. Provide the final solution as a single Lean 4 expression.
3. Present the final output inside a Lean code block, using:
```lean
abbrev solution {solution code}
```
Informal statement
{{ example.informal_statement }}
```

Figure 4: Jinja template for the system and user prompt used in to generate a conjecture in Lean 4.

B.2 AUTOFORMALISATION PROMPT

The prompt used to generate the autoformalisation is given in Figure 5.

Autoformalisation Prompt

System Prompt

```
You are an advanced assistant specializing in formal mathematics and Lean 4
theorem proving. You have extensive expertise in translating mathematical
concepts from natural language into precise Lean 4 code.
```

User Prompt

```
Translate the following natural language statement, provided under
**Informal statement** into a formal Lean 4 theorem. Use the theorem name
specified under **Name** as the Lean identifier for the theorem. Your
response must:
- Write only valid Lean 4 code, with clear and idiomatic use of Lean
syntax and conventions.
- Only include the formalization, and do not include any proof or imports.
- Define the theorem using the provided name.
- Faithfully capture the meaning of the informal statement in your
formalization.
- Enclose all Lean code within triple backticks
Output:
```lean
theorem [NAME] : [Lean formalization of the statement] := sorry
```
{%- for example in examples %}
EXAMPLE {{ example.id }}:
Name
{{ example.name }}
Informal statement
{{ example.informal_statement }}
The code below presents a solution implementation written in Lean 4.
This solution has already been incorporated into the current Lean
environment and is available for use in the formalization.
import Mathlib
{%- if conjecture_is_seen %}
{{ example.conjecture }}
{%- endif %}
Output:
```lean
{{ example.formal_statement }}
```
Above are examples for you to model the translation of the below natural
language statement into a Lean 4 formal theorem:
{%- endfor }
Name
{{ query.name }}
Informal statement
{{ query.informal_statement }}
The code below presents a solution implementation written in Lean 4.
This solution has already been incorporated into the current Lean
environment and is available for use in the formalization.
import Mathlib
{%- if conjecture_is_seen %}
{{ example.conjecture }}
{%- endif %}
Combined Hints
{{ query.combined_cot_lot }}
Output:
```lean
```

Figure 5: Jinja templates for the system and user prompt for autoformalisation.
```