# AI Impact on Human Proof Formalization Workflows

**Katherine M. Collins**[†,1,2], **Simon Frieder**[†,3], **Jonas Bayer**[2], **Jacob Loader**[2], **Jeck Lim**[4], **Peiyang Song**[4], **Fabian Zaiser**[1], **Lexin Zhou**[5], **Shanda Li**[6], **Shi-Zhuo Looi**[4], **Jose Hernandez-Orallo**[7,2], **Joshua B. Tenenbaum**[1], **Cameron E. Freer**[1], **Umang Bhatt**[2], **Adrian Weller**[2], **Valerie Chen**[‡,6], **Ilia Sucholutsky**[‡,8]
Massachusetts Institute of Technology[1], University of Cambridge[2], University of Oxford[3], Caltech[4], Princeton University[5], Carnegie Mellon University[6], Universitat Politècnica de València[7], New York University[8]
[†]Contributed equally. Corresponding authors: katiemc@mit.edu, simon.frieder@wolfson.ox.ac.uk
[‡]Co-senior authors

## Abstract

Human mathematicians have written proofs for centuries to substantiate their mathematical arguments. The ability to automatically check the validity of proofs has long been a dream. The development of tools for such checking of "formalized" proofs has been made possible by new languages for formal mathematics, like Lean. Advances in AI systems' ability to generate code promises to transform the ability to formalize proofs and to assist humans in this task. Recent studies have examined how human programmers engage with AI tools for code generation, but the role of AI in humans' formalization process is comparatively understudied. Here, we conduct an initial exploration into people's formalization process with and without AI. We collect more than 80 hours of video from seven participants formalizing informal proofs with and without AI on a range of mathematical problems covering different levels of difficulty and domains. We offer a first characterization of people's formalization process, noting places where AI assistance helps — and a few instances where it may hurt.

## 1 Introduction

Systems to formalize mathematics (e.g., Lean, Isabelle, Rocq) are surging in popularity as means of rigorously and verifiably writing proofs [Brasca et al., 2025, Loeffler and Stoll, 2025, Asgeirsson, 2024]. Each such system encodes mathematics in a formal language — in contrast to the way mathematics has been written for centuries: "informally", through natural language, where checks are done by other humans using their knowledge of the respective mathematical domain. This process can become increasingly error-prone as mathematical proofs become more difficult. It is not inconceivable, then, that in the coming decades, mathematics journals may require new submissions to be submitted with a formal counterpart (or fragments thereof, e.g., "fabstracts", short for formal abstracts, as proposed by Hales [Hales and FormalAbstracts Project, 2017]), which can be automatically verified.

However, it is not always natural to write mathematical proofs formally in a language like Lean [Bayer et al., 2024]. This would require knowledge of the wide library of existing formalized mathematics in that system, its "tactics" for executing proof steps, as well as understanding the tool. Since systems like Lean are still maturing, their libraries, tactics, and tools are frequently changing. The low-level rote coding work of formalizing, which must be precise and detailed, often obscures the elegance and beauty of a high-level mathematical argument, especially when performed unassisted.[1]

[1]For example, Loeffler and Stoll [2025] state: "*We found it was not easily possible to set up a coercion from* `ArithmeticFunction` $R$ *to* `ArithmeticFunction` $R'$ *whenever there is a coercion from* $R$ *to* $R'$*, because the*

The 5th Workshop on Mathematical Reasoning and Artificial Intelligence (MATH-AI), 2025.

Although recent studies have aimed to understand how code assistance helps programmers, the majority of these studies have focused on traditional programming languages [Yan et al., 2025, Etsenake and Nagappan, 2024]. In this work, we focus exclusively on Lean,[2] as it has the largest user base, is dynamic, and has become the standard in terms of formalization. What kinds of tools can help mathematicians formalize their work, faster and less encumbered, while maintaining the joy of their proof-writing process? What kinds of assistance might mathematicians seek when formalizing their work, and when might they seek it? In what circumstances may AI-based assistance hamper the formalization process?

Recent advances in AI have promised to automate more of mathematics and facilitate the synthesis of formal proofs from informal arguments. Each of these tools accomplishes or excels at different subtasks around formalizing a piece of mathematics; for instance, Lean Copilot [Song et al., 2024] provides a way for researchers to inquire about tactics, complete proofs, and perform premise selection; moogle[3] and leansearch[4] help to search Mathlib; and LLMs like GPT or Claude provide formalization snippets, as well as general info about Mathlib, if foundational knowledge is missing (e.g., whether a particular mathematical definition has been formalized). However, effective use of such tools requires a base ability with Lean, mathematical experience, and knowledge about the particular strengths and weaknesses of each tool, as well as about which tool combinations are effective—and not all AI use may be helpful in all scenarios.

We perform the first (to our knowledge) user study of mathematicians formalizing mathematics with AI-based tools. Our goal is to assess whether access to such tools impacts the accuracy and efficiency of the formalization process that a user takes — and to begin to characterize the kinds of patterns users show in how they interact with AI systems as part of formalization. We recruited $8$ participants[5] to engage in a systematic evaluation of formalizing natural language proofs across $6$ mathematics problems – spanning a range of mathematical domains – that explore different aspects of formalization, totaling $80$ hours of participants' recorded formalization processes. In the remainder of this paper, we first conduct a quantitative analysis into participants' formalization process with and without AI-based tools, at an aggregate level (e.g., proof accuracy and time-to-formalize) and then qualitative analysis from individual videos and post-survey responses. Our work offers a snapshot view into how AI tools may be used (or may impair) participants' formalization process. We intend for our work to be an initial glimpse into usage patterns, from which future scaled human-AI interaction studies can be conducted to inform the design of more human-compatible AI thought partners for mathematics [Collins et al., 2024, Frieder et al., 2024].

## 2 User Study

### 2.1 Problems

We curate a selection of six problems (identified by the letters *C*, *N*, *A*, *V*, *G*, *T*, which loosely correspond to their mathematical domain, e.g., *N* represents *number theory*; see Appendix A2.1). This relatively low number of problems is necessary to gather sufficient data for each problem and keep formalization time within reasonable bounds. Hence, we carefully source problems so that they cover a range of domains, mathematical difficulty, and formalization difficulty (since a problem being easy mathematically does not imply easy formalization). More details on problem selection are included in Appendix A2.1.

### 2.2 Participants

Eight participants completed our study; however, we excluded one participant who failed to record their videos separately. Participants were recruited to have at least a baseline level of Lean experience (e.g., engaged with at least a tutorial like the Natural Numbers Game [Bayer et al., 2025]). All participants had taken at least one undergraduate mathematics level course; most had post-graduate

*necessary coercion instance would have too many free parameters;*" and refer to an extended Zulip chat which highlights difficulties that can arise.

[2] `https://lean-lang.org/`

[3] `https://www.moogle.ai/`

[4] `https://leansearch.net/`

[5] One participant was excluded due to failing to follow the study methodology in their video recording.

level mathematics experience (see Appendix A2). In their post-survey, participants indicated their prior level of Lean experience; participants happened to be split roughly evenly along those who categorized themselves as "Beginner" ($N = 3$) versus "Advanced" or "Expert" ($N = 4$) formalizers. Participants were given six informal mathematical problems, together with their proofs, to formalize. Problems were split into two groups of three, matched by formalization difficulty across groups. Participants were given two weeks to formalize the problems and proofs; participants are instructed to formalize a group of three problems and proofs in week one, and the other group of three in week two. Participants' tool access differed across weeks. In one week, participants were not allowed to access any tool (the "human alone" condition). In the other week, participants were allowed access to any AI-based online resource. This included all tools such as leansearch, Lean Copilot, or ChatGPT. Participants were randomly assigned whether to use AI-assistance in week one or two. We asked that participants record their screens while working and send us the recordings. The resulting study includes over 80 hours of participants' screen-recorded formalization processes. Additional details are in Appendix A2.2.

### 2.3 Analysis Methodology

Our primary measures are formalization accuracy and time-to-formalize. Three experienced Lean users from our author team manually grade the accuracy of participants' statements and proofs; one grader marked each problem. Responses are scored with a binary correct or not based on whether both the statement and proof are correct. This is inherently subjective in our current definition of accuracy; we are actively working on expanding our accuracy evaluation for future work. We include an initial more granular analysis of correctness in Appendix A4.2. Time is measured directly from the videos; participants were asked to record their entire problem solving session, which often unfolded over one or more sessions per problem. Time-to-formalize is summed over all sessions per problem. We also qualitatively assess which AI-based formalization tools people use, the number of different tools they use, and the intensity of use for each tool. We coupled these investigations with an analysis of a sampling of the live recorded videos. All our formalizations will be made publicly available upon full publication.

## 3 Results

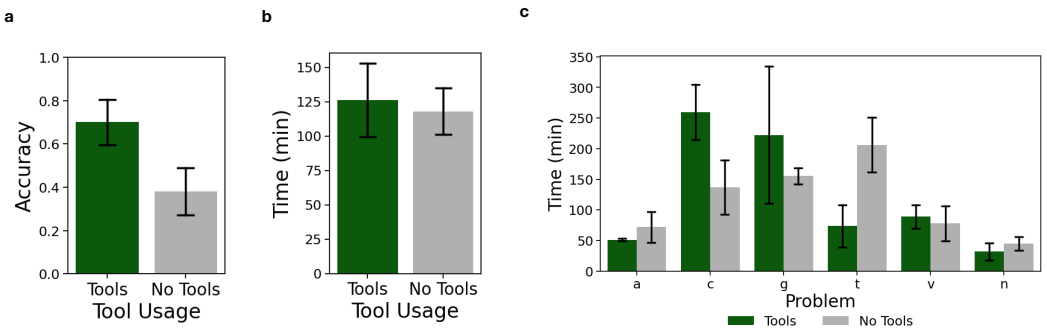


Figure 1: **Aggregate problem solving statistics, with and without tool access. a,** Average accuracy (both statement and proof formalized correctly, see Appendix) across problems, for groups with and without tool use; **b,** Average time taken (in minutes) across problems; **c,** Average time taken (in minutes) broken down by problem. Error bars depict standard error.

**AI use tends to increase formalization accuracy, with a mixed effect on time.** Participants' formalizations are generally more accurate when allowed tool use is higher than when not allowed tool use (Figure 1a). Participants do not *always* formalize proofs correctly with AI assistance; for instance, no participant solved "Problem G" (see Appendix Figure 5). In particular, in exploratory analyses, we observe that one effect of AI assistance may be to boost participants with less formalization or mathematics experience closer to the level of formalization quality of more experienced participants (see Figures 3 and 4 in Appendix A4).

We observe, however, that the impact of AI assistance on formalization time is mixed (Figure 1b) and dependent on the problem (Figure 1c). For instance, we observe that in some problems (e.g., "Problem C") participants with AI access took substantially longer (over an hour, on average) in their formalization.

**Participants' AI usage is varied and frequently tailored to the formalization task.** Participants are varied in their number (Figure 2a) and choice of AI-based tools (Figure 2b). Most participants used more than one (in fact, at least 3) different kinds of tools. While almost all (6 of 7) participants used in-line GitHub Copilot in their VSCode environment, participants were more selective in whether they used other "chatbots" (e.g., ChatGPT or Gemini) or language models (e.g., Kimina Prover [Wang et al., 2025]). Participants seemed to be selecting their choice of tool deliberately, for the task at hand (see their free-responses in Appendix A3). Participants' responses therefore shed light on the kinds of workflows people may use for formalization. We caveat these results, however, by noting that most participants in our study (6 of 7) had baseline familiarity with AI tools in their own work.

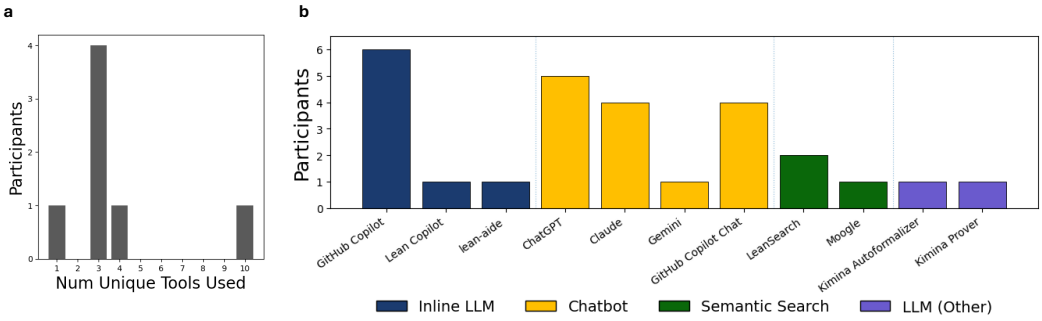


Figure 2: **Tools used by participants. a,** Number of unique tools used by each participant; **b,** Types of tools used across participants.

**Participants differ in their reliance on AI tools.** While participants indicated their kind of AI-based tool use and descriptions of what led to their use, this does not reveal how they used the tools over the course of the (many) hours that they were formalizing. To begin to understand the kind of live behavior participants show interacting with AI-based tools, we conduct a preliminary exploratory analysis into some of the recorded videos. From our initial observations, however, we observed several "kinds" of participant engagement with AI: several participants engaged with light tool use and could be called "human formalizers with AI assistance"; another group heavily used AI, only occasionally editing and the proof manually ("AI formalization with human help"); a third set is more nuanced and intricate, involving heavy and varied AI use yet with extensive human engagement. We describe example usage patterns in Appendix A4.4. Participants also provided rich qualitative reflection on their own AI usage, current frustrations, and dreams for new kinds of AI-empowered workflows, which we included in Appendix A5.

## 4 Discussion and Limitations

We observe that while AI-based tools generally seem to improve people's accuracy when formalizing proofs, the effectiveness of AI use is highly dependent on the manner in which people integrate tools into their workflows. Different AI-based tools may serve to help with different parts of the formalization process, and vary in their accuracy at formalization — warranting critical use on the part of the human participant. However, we note that we have just scratched the surface of our analyses. Our dataset of recorded human formalization processes is sure to elucidate further insights into different peoples' formalization processes. Critically we see already — even in a population that may otherwise seem homogeneous (people with experience in Lean) — that the particular style of AI use (and willingness to accept AI suggestions) is highly varied.

We caveat however that while we have substantial data from each participant formalizing each problem, our current study is nonetheless small in number of participants and number of problems each participant engaged with. We also conduct only a preliminary analysis of tool use (e.g., model

class like Claude, but not specific version). We are actively expanding our video and tool use analyses. There is also a question of how general the AI-assisted behavior participants show extend to more naturalistic formalization settings. Moreover, in our study, we always provided participants with the informal proof. In practice, some mathematics students and researchers may jointly formalize their proof alongside the process of discovering the informal proof itself.

## Acknowledgments

We thank Ced Zhang, Albert Jiang, Tim Gowers, Mateja Jamnik, Kaiyu Yang, and Hussein Mozzanar for valuable conversations that informed this work. KMC acknowledges support from the Cambridge Trust and King's College Cambridge. AW acknowledges support from a Turing AI Fellowship under grant EP/V025279/1, The Alan Turing Institute, and the Leverhulme Trust via CFI.

## Participant Contributors

We thank all participants of our study. The following participants opted to be listed as contributors: Xavier Lien, Cayden Codel, Fabian Zaiser, Mauricio Barba, and Anand Tadipatri; the others elected to remain anonymous.

## References

J. Aitken, P. Gray, T. Melham, and M. Thomas. Interactive theorem proving: An empirical study of user activity. *Journal of Symbolic Computation*, 25(2):263–284, 1998. ISSN 0747-7171. doi: https://doi.org/10.1006/jsco.1997.0175. URL https://www.sciencedirect.com/science/article/pii/S0747717197901759.

Anthropic. Claude code overview, 2025a. URL https://docs.anthropic.com/en/docs/claude-code/overview.

Anthropic. Anthropic Economic Index: AI's impact on software development, 2025b. URL https://www.anthropic.com/research/impact-software-development.

D. Asgeirsson. Towards solid abelian groups: A formal proof of Nöbeling's theorem. In Y. Bertot, T. Kutsia, and M. Norrish, editors, *15th International Conference on Interactive Theorem Proving (ITP 2024)*, volume 309 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 6:1–6:17, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi: 10.4230/LIPIcs.ITP.2024.6. URL https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.ITP.2024.6.

S. Barke, M. B. James, and N. Polikarpova. Grounded Copilot: How programmers interact with code-generating models. *Proceedings of the ACM on Programming Languages*, 7(OOPSLA1): 85–111, 2023.

J. Bayer, C. Benzmüller, K. Buzzard, M. David, L. Lampert, Y. Matiyasevich, L. Paulsen, D. Schleicher, B. Stock, and E. Zelmanov. Mathematical Proof Between Generations. *Notices of the American Mathematical Society*, 71(01):1, Jan. 2024. doi: 10.1090/noti2860.

J. Bayer, J. Loader, K. M. Collins, S. Frieder, A. Weller, J. B. Tenenbaum, and T. Gowers. Studying mathematical reasoning through the Gadget Game. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, volume 47, 2025.

J. Becker, N. Rush, E. Barnes, and D. Rein. Measuring the impact of early-2025 AI on experienced open-source developer productivity. *arXiv preprint arXiv:2507.09089*, 2025.

R. Brasca, C. Birkbeck, E. Rodriguez Boidi, A. Best, R. van De Velde, and A. Yang. A complete formalization of Fermat's Last Theorem for regular primes in Lean. *Annals of Formalized Mathematics*, 1, 2025. doi: 10.46298/afm.14586. URL https://afm.episciences.org/16046.

L. Chen, J. Gu, L. Huang, W. Huang, Z. Jiang, A. Jie, X. Jin, X. Jin, C. Li, K. Ma, C. Ren, J. Shen, W. Shi, T. Sun, H. Sun, J. Wang, S. Wang, Z. Wang, C. Wei, S. Wei, Y. Wu, Y. Wu, Y. Xia, H. Xin, F. Yang, H. Ying, H. Yuan, Z. Yuan, T. Zhan, C. Zhang, Y. Zhang, G. Zhang, T. Zhao, J. Zhao, Y. Zhou, and T. H. Zhu. Seed-Prover: Deep and broad reasoning for automated theorem proving, 2025a. URL https://arxiv.org/abs/2507.23726.

V. Chen, A. Talwalkar, R. Brennan, and G. Neubig. Code with me or for me? How increasing AI automation transforms developer workflows. *arXiv preprint arXiv:2507.08149*, 2025b.

B. Chopra, A. Singha, A. Fariha, S. Gulwani, C. Parnin, A. Tiwari, and A. Z. Henley. Challenges in using conversational AI for data science. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*, HILDA '25, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400719592. doi: 10.1145/3736733.3736748. URL https://doi.org/10.1145/3736733.3736748.

Cognition. Introducing Devin, the first AI software engineer, 2024. URL https://cognition.ai/blog/introducing-devin.

K. M. Collins, I. Sucholutsky, U. Bhatt, K. Chandra, L. Wong, M. Lee, C. E. Zhang, T. Zhi-Xuan, M. Ho, V. Mansinghka, A. Weller, J. B. Tenenbaum, and T. L. Griffiths. Building machines that learn and think with people. *Nature Human Behaviour*, 8(10):1851–1863, 2024.

Z. K. Cui, M. Demirer, S. Jaffe, L. Musolff, S. Peng, and T. Salz. The effects of generative AI on high-skilled work: Evidence from three field experiments with software developers. *Available at SSRN 4945566*, 2025.

A. de Almeida Borges, A. Casanueva Artís, J.-R. Falleri, E. J. Gallego Arias, É. Martin-Dorel, K. Palmskog, A. Serebrenik, and T. Zimmermann. Lessons for interactive theorem proving researchers from a survey of Coq users. *Journal of Automated Reasoning*, 69(1):8, 2025.

D. Etsenake and M. Nagappan. Understanding the human-LLM dynamic: A literature survey of LLM use in programming tasks. *arXiv preprint arXiv:2410.01026*, 2024.

S. Frieder, J. Bayer, K. M. Collins, J. Berner, J. Loader, A. Juhász, F. Ruehle, S. Welleck, G. Poesia, R.-R. Griffiths, A. Weller, A. Goyal, T. Lukasiewicz, and T. Gowers. Data for mathematical copilots: Better ways of presenting proofs for machine learning. *arXiv preprint arXiv:2412.15184*, 2024.

K. Gu, R. Shang, T. Althoff, C. Wang, and S. M. Drucker. How do analysts understand and verify AI-assisted data analyses? In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, pages 1–22, 2024.

T. Hales and FormalAbstracts Project. Formal abstracts. https://formalabstracts.github.io/, 2017. Accessed: 2025-09-19.

M. Jorgensen, K. Brogle, K. M. Collins, L. Ibrahim, A. Shah, P. Ivanovic, N. Broestl, G. Piles, P. Dongha, H. Abdulhussein, et al. Documenting deployment with fabric: A repository of real-world AI governance. *AAAI/ACM Conference on Artificial Intelligence, Ethics, and Society*, 2025.

G. F. Kadoda. A cognitive dimensions view of the differences between designers and users of theorem proving assistants. In *12th Workshop of the Psychology of Programming Interest Group (PPIG)*, page 9, 2000.

M. Kazemitabaar, J. Chow, C. K. T. Ma, B. J. Ericson, D. Weintrop, and T. Grossman. Studying the effect of AI code generators on supporting novice learners in introductory programming. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, pages 1–23, 2023.

D. Loeffler and M. Stoll. Formalizing zeta and L-functions in Lean. *Annals of Formalized Mathematics*, 1, 2025. doi: 10.46298/afm.15328. URL https://afm.episciences.org/15954.

J. Lu, Y. Wan, Z. Liu, Y. Huang, J. Xiong, C. Liu, J. Shen, H. Jin, J. Zhang, H. Wang, Z. Yang, J. Tang, and Z. Guo. Process-driven autoformalization in Lean 4. *arXiv preprint arXiv:2406.01940*, 2024. doi: 10.48550/arXiv.2406.01940. URL https://arxiv.org/abs/2406.01940.

N. A. Merriam. Two modelling approaches applied to user interfaces to theorem proving assistants. In *Proceedings of the 2nd International Workshop on User Interface Design for Theorem Proving Systems*, pages 75–82, 1996.

N. A. Merriam and M. D. Harrison. Evaluating the interfaces of three theorem proving assistants. In *Design, Specification and Verification of Interactive Systems' 96: Proceedings of the Eurographics Workshop in Namur, Belgium, June 5–7, 1996*, pages 330–346. Springer, 1996.

H. Mozannar, V. Chen, M. Alsobay, S. Das, S. Zhao, D. Wei, M. Nagireddy, P. Sattigeri, A. Talwalkar, and D. Sontag. The RealHumanEval: Evaluating large language models' abilities to support programmers. *Transactions on Machine Learning Research*.

H. Mozannar, G. Bansal, A. Fourney, and E. Horvitz. Reading between the lines: Modeling user behavior and costs in AI-assisted programming. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, pages 1–16, 2024.

D. Nam, A. Macvean, V. Hellendoorn, B. Vasilescu, and B. Myers. Using an LLM to help with code understanding. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, pages 1–13, 2024.

Numina. NuminaMath-CoT: A collection of ~860k competition math problems with chain-of-thought solutions. Hugging Face dataset, 2024. URL https://huggingface.co/datasets/AI-MO/NuminaMath-CoT. Approximately 860k problems with CoT solutions.

Numina. NuminaMath-LEAN: 100,000 Lean 4 formal statements and proofs from competition math. Hugging Face dataset, 2025. URL https://huggingface.co/datasets/AI-MO/NuminaMath-LEAN. Largest human-annotated Lean 4 corpus; used to train Kimina-Prover 72B.

S. Peng, E. Kalliamvakou, P. Cihon, and M. Demirer. The impact of AI on developer productivity: Evidence from github copilot. *arXiv preprint arXiv:2302.06590*, 2023.

J. Prather, B. N. Reeves, P. Denny, B. A. Becker, J. Leinonen, A. Luxton-Reilly, G. Powell, J. Finnie-Ansley, and E. A. Santos. "It's weird that it knows what I want": Usability and interactions with Copilot for novice programmers. *ACM Trans. Comput.-Hum. Interact.*, 31(1), nov 2023. ISSN 1073-0516. doi: 10.1145/3617367. URL https://doi.org/10.1145/3617367.

Z. Z. Ren, Z. Shao, J. Song, H. Xin, H. Wang, W. Zhao, L. Zhang, Z. Fu, Q. Zhu, D. Yang, Z. F. Wu, Z. Gou, S. Ma, H. Tang, Y. Liu, W. Gao, D. Guo, and C. Ruan. DeepSeek-Prover-V2: Advancing formal mathematical reasoning via reinforcement learning for subgoal decomposition, 2025. URL https://arxiv.org/abs/2504.21801.

S. I. Ross, F. Martinez, S. Houde, M. Muller, and J. D. Weisz. The programmer's assistant: Conversational interaction with a large language model for software development. In *Proceedings of the 28th International Conference on Intelligent User Interfaces*, pages 491–514, 2023.

J. Shi, C. Torczon, H. Goldstein, B. C. Pierce, and A. Head. QED in context: An observation study of proof assistant users. *Proceedings of the ACM on Programming Languages*, 9(OOPSLA1): 337–363, 2025.

P. Song, K. Yang, and A. Anandkumar. Lean Copilot: Large language models as copilots for theorem proving in Lean. *arXiv preprint arXiv:2404.12534*, 2024.

P. Vaithilingam, T. Zhang, and E. L. Glassman. Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models. In *CHI Conference on Human Factors in Computing Systems Extended Abstracts*, pages 1–7, 2022.

H. Vasconcelos, G. Bansal, A. Fourney, Q. V. Liao, and J. Wortman Vaughan. Generation probabilities are not enough: Uncertainty highlighting in AI code completions. *ACM Trans. Comput.-Hum. Interact.*, 32(1), Apr. 2025. doi: 10.1145/3702320. URL https://doi.org/10.1145/3702320.

H. Wang, M. Unsal, X. Lin, M. Baksys, J. Liu, M. D. Santos, F. Sung, M. Vinyes, Z. Ying, Z. Zhu, et al. Kimina-Prover Preview: Towards large formal reasoning models with reinforcement learning. *arXiv preprint arXiv:2504.11354*, 2025.

X. Wang, B. Li, Y. Song, F. F. Xu, X. Tang, M. Zhuge, J. Pan, Y. Song, B. Li, J. Singh, et al. OpenHands: An open platform for AI software developers as generalist agents. In *The Thirteenth International Conference on Learning Representations*, 2024.

J. D. Weisz, S. V. Kumar, M. Muller, K.-E. Browne, A. Goldberg, K. E. Heintze, and S. Bajpai. Examining the use and impact of an ai code assistant on developer productivity and experience in the enterprise. In *Proceedings of the Extended Abstracts of the CHI Conference on Human Factors in Computing Systems*, CHI EA '25, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400713958. doi: 10.1145/3706599.3706670. URL https://doi.org/10.1145/3706599.3706670.

Y.-M. Yan, C.-Q. Chen, Y.-B. Hu, and X.-D. Ye. LLM-based collaborative programming: Impact on students' computational thinking and self-efficacy. *Humanities and Social Sciences Communications*, 12(1):149, 2025. doi: 10.1057/s41599-025-04471-1. URL https://www.nature.com/articles/s41599-025-04471-1.

K. Yang, A. Swope, A. Gu, R. Chalamala, P. Song, S. Yu, S. Godil, R. J. Prenger, and A. Anandkumar. LeanDojo: Theorem proving with retrieval-augmented language models. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 21573–21612. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/4441469427094f8873d0fecb0c4e1cee-Paper-Datasets_and_Benchmarks.pdf.

H. Ying, Z. Wu, Y. Geng, J. Wang, D. Lin, and K. Chen. Lean workbook: A large-scale Lean problem set formalized from natural language math problems. *arXiv preprint arXiv:2406.03847*, 2024. doi: 10.48550/arXiv.2406.03847. URL https://arxiv.org/abs/2406.03847.

A. Ziegler, E. Kalliamvakou, X. A. Li, A. Rice, D. Rifkin, S. Simister, G. Sittampalam, and E. Aftandilian. Productivity assessment of neural code completion. In *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming*, pages 21–29, 2022.

# Appendix



## A1 Additional related work

**AI for formalization** Many recent formalization systems use LLMs to provide varying degrees of automation to users. Interactive tools range from retrieval-augmented generation (RAG) pipelines such as ReProver [Yang et al., 2023] that suggest relevant Mathlib premises to the user, to assistants such as Lean Copilot [Song et al., 2024] that provide tactic suggestion, proof search, and premise selection in the editor. A greater degree of automation is provided by systems such as DeepSeek-Prover-v2 [Ren et al., 2025], Kimina Prover [Wang et al., 2025], or Seed-Prover [Chen et al., 2025a] that are trained via reinforcement learning that aim to one-shot formalizations more accurately or make use of a range of tools to improve formalization quality. Large upstream corpora such as NuminaMath (approximately 860k competition-style problems with chain-of-thought solutions) and the Lean-specific NuminaMath-LEAN (approximately 100k human-annotated Lean 4 statements and proofs) increasingly train math LLMs and Lean provers, including Kimina-Prover, so they help contextualize the capabilities available to participants in our study [Numina, 2024, 2025, Wang et al., 2025]. Complementing prover systems, process-driven autoformalization in Lean 4 introduces the FormL4 benchmark and a process-supervised verifier that leverages Lean 4 compiler feedback, and Lean Workbook contributes about 57k natural language-Lean pairs via an iterative translation-and-filtering pipeline checked by the Lean compiler [Lu et al., 2024, Ying et al., 2024].

**Interactive theorem proving user studies** A body of work from the 1990s and early 2000s studied how mathematicians engage with interactive theorem proving workflows for formalization [Aitken et al., 1998, Merriam and Harrison, 1996, Merriam, 1996, Kadoda, 2000]. For instance, Merriam and Harrison characterized four key activities that people engage in when formalizing: planning

(designing the high-level sketch of the proof), reusing (leveraging previously written code and writing code strategically with reuse in mind), reflecting (evaluating what has been written and further understanding the proof goal), and articulating (actually specifying commands to the prover assistant). As we explore in our work, these four strategies can each be modularly engaged with AI-based tools for formalization (effectively or not). Additional user studies into tools for formalization have been conducted since, e.g., in de Almeida Borges et al., the authors analyze a wide range of Coq users' survey responses about their use behavior, finding marked differences in styles and preferences around formalization depending on the level of experience of the user. Most closely related to our work, Shi et al. conduct an observation study formalizing proofs in Lean and Rocq; this work offers a rich, in-depth look at modern formalization practice. The authors observe varied tool use, similar to the kind engaged in our study, and also notice that several users care about features beyond just correct formalization—a pattern we noticed in our study as well (e.g., some participants tried to write their proofs to use particular libraries, or commented on how "ugly" they found their code). Most differentially from us, they do not substantively investigate peoples' interaction with AI systems as part of their formalization, in contrast to the focus of our work.

**Human-AI interaction for coding** A fairly extensive set of user studies has been conducted to understand how developers code with various AI copilots. These studies have focused on two forms of assistance that AI copilots provide: autocomplete suggestions [Vaithilingam et al., 2022, Peng et al., 2023, Barke et al., 2023, Prather et al., 2023, Mozannar et al., 2024, Vasconcelos et al., 2025, Cui et al., 2025, Mozannar et al.] and chat dialogue [Ross et al., 2023, Chopra et al., 2025, Kazemitabaar et al., 2023, Gu et al., 2024, Nam et al., 2024, Mozannar et al.]. These studies show how copilots have generally had a positive impact on software development, e.g., leading to an increase in perceived productivity [Ziegler et al., 2022, Weisz et al., 2025, Becker et al., 2025] and rate of task completion in controlled studies [Vaithilingam et al., 2022, Peng et al., 2023] compared to developers writing code on their own. More recently, moving beyond copilots for software development, we have seen the introduction of coding *agents* (e.g., Devin [Cognition, 2024], OpenHands [Wang et al., 2024], Claude Code [Anthropic, 2025a]). As such, there is growing work that compares prior copilot tools to agents [Anthropic, 2025b, Chen et al., 2025b], showing how more autonomous tools change developer workflows.

## A2 Additional details on study design

We next provide additional details on study design, namely problem construction, participant recruitment, and problem grading.

### A2.1 Additional details on problem construction

Problems were selected and constructed by our author team. All problems have sufficiently short formal proofs that a team of pilots was able to formalize all six of them in approximately twelve hours. This team was made up from the authors of this study, and formalizers all had at least a first degree in mathematics, or were at most at the level of a Post-Doc; all had significant prior experience in formalizing.

Formalization difficulty comes in two parts: the intrinsic difficulty of formalizing a problem; and whether Mathlib provides pre-existing definitions and support to carry out formalization. In the latter case, simply adapting the key parts of the known formal proof to transform it to the sought proof is much easier than formalizing from scratch. We also account for this possibility and include both problems where a similar formalization exists, as well as problems where a similar formalization does not exist.

We provide a brief overview of what each problem involved:

- *Problem N:* This is a simple number theory problem about solving a congruence equation. The expected format of the solution is deliberately open-ended to encourage participants to present their answers in diverse ways.
- *Problem A:* This is an analysis problem involving continuous and differentiable functions and the use of the mean value theorem.

- *Problem C:* This is a counting problem about integer sequences defined by a self-referential rule involving divisibility. After classifying the solutions, one must apply foundational results about finite cardinalities to complete the proof.
- *Problem G:* This is a geometry problem, solved primarily using angle chasing. Part of the solution involves constructing an auxiliary point along an interval that creates a specified angle with another point, which seems "obviously" valid to a human mathematician. Much of the difficulty in formalizing comes from showing that such a point exists, which requires careful use of the intermediate value theorem, noting that the angle subtended varies continuously as a point moves along a line.
- *Problem T:* This is a topology problem involving Baire spaces, which requires unraveling definitions, and careful understanding of subspace topologies.
- *Problem V:* This is a simple visual counting problem involving tiling a 2x3 grid with dominoes. The mathematical proof is simple; the difficulty comes from having to formally state this problem, which is cumbersome to do in Lean.

### A2.2 Additional participant details

We recruited participants through our networks through university mailing lists and personal contacts. Participants were told the study would take approximately 12 hours and were paid at a fixed rate of $240 (for an estimated $20/hr). In case of questions regarding what consists of AI tool use, we worked with the participants to make sure our criteria were uniform across participants. Participants provided self-reports at the end of the study, indicating their experience with mathematics and formalization, as well as providing qualitative responses into their AI use (see Section A5. Participants also provided screenshots of any scratchwork used during their formalization process. The study received prior ethics approval by our university departmental ethics review, and all participants provided informed consent.

All participants noted that they used Lean at least once a month (they were given the option to indicate less frequent use). All participants indicated that they had "moderate comfort" with at least 1 formal proof language (Four participants were comfortable with one language; two with two languages; and one participant with three languages). Four participants indicated that they "always" used AI-based tools when formalizing in their day-to-day workflows; two indicated they "sometimes" did; and one indicated "rarely" using AI-based tools.

### A2.3 Additional analysis details

Three experienced Lean users from our author team marked the accuracy of participants' formalized proofs.

We code tool use based on a mixture of participants' self-reported tool use (see Section A3) and notes from watching videos, as participants did not always report all tools they used. While we have watched several hours of participants' videos, and notice that most participants are consistent in their tool use across problem sessions, until we have completed a full coding of all 80 hours of video, it is there possible that we missed some tool use.

## A3 Participant self-reported tool usage

After the study, participants wrote in how they used certain tools. We include participant free-form responses verbatim. Note, upon video inspection, some participants did not list all tools they used (e.g., many used GitHub Copilot in-line, even if not written).

**Participant 1**

- *"LeanSearch/Moogle: I use these to locate relevant results in the library. My typical workflow is to use these tools via the browser, but I've gradually started using the* `#search` *command from Mathlib that makes it possible to access these tools directly from the Lean editor.*
- *LeanAide/Kimina autoformalizer: I use these tools to suggest ways to formalize statements in Lean. Even if the formalizations are wrong, they're still helpful because they point me to definitions in the library that I may not have previously considered.*

- *Kimina Prover model: If a lemma seems sufficiently easy, I try to offload it to the Kimina prover model to see if it can discover a proof automatically.*
- *GitHub Copilot chat: Even though most LLMs available in the GitHub Copilot chat interface are not specifically trained on Lean, I find it useful to add relevant Lean files to the chat context and ask various questions or get suggestions on planning out and structuring a proof."*

**Participant 2**

- *"I used ChatGPT by inputting the pdf file and asking it to formalize the proof for me*
- *I used ChatGPT by asking it to do deep research to find similar problems that have been formalized*
- *I used Claude by inputting the pdf file and asking it to formalize the proof for me*
- *I used Claude by asking it to do deep research to find similar problems that have been formalized*
- *I used Copilot in vscode to do quick generation to small chunks of the proofs"*

**Participant 3**

- *"leansearch.org – helps me find theorems that I suspect ought to already exist. This is the main thing that I was sad not to have in the NO-TOOLS segment. I describe the theorem I want in natural language or semi-formal language, and leansearch very often points me in a good direction.*
- *Gemini or Chatgpt – if I'm really stuck, sometimes I'll copy the whole problem to one of these tools. Sometimes they make helpful suggestions."*

**Participant 4**

- *"GitHub Copilot for VSCode. I use this as intelligent autocomplete. It rarely produces correct proofs, but it can guess names and patterns very well, especially when typing out repetitive code.*
- *GitHub Copilot Chat for VSCode. I did not use during the study, but sometimes I use it in agent mode to state and prove repetitive lemmas, e.g. many trivial consequences of some theorem. I also use it for code generation, e.g. when writing Lean tactics.*
- *Language models (e.g. Claude Sonnet 4). They are helpful for finding useful Unix commands. They are not helpful for understanding mathematics because they are often confidently incorrect."*

**Participant 5**

- *"I exclusively use GitHub Copilot (it comes for free with my [...] account). I use Copilot to help me write normal code (e.g. C and Python) and to formalize in Lean. I usually don't use the chat feature, preferring to use the tool when I'm writing code and theorems. I tend to start function names and definitions myself, rather than writing a comment and letting the tool fill it in, which gives me greater control over the shape and feel of a project."*

**Participant 6**

- *"Github Copilot (all the time, for code suggestions; sometimes for questions in chat), ChatGPT (sometimes, for questions)"*

**Participant 7**

- *"I use ChatGPT to come up with general ideas about how to formalize a problem, and I use the starter code if it is any good."*

Interestingly, Participant 7 is the participant who immediately pasted their problem into both ChatGPT and Claude windows and proceeded to heavily rely on AI tools for the entire proof. Their self-report dramatically understates their AI usage.

## A4 Additional analyses into participants' formalization

We include additional exploratory analyses into participants' formalization results.

### A4.1 Formalization decomposed by experience

We also decompose participants' formalization accuracy based on their self-reported Lean experience (Figure 3) and mathematics experience (Figure 4. We caution over interpretation due to small sample sizes.

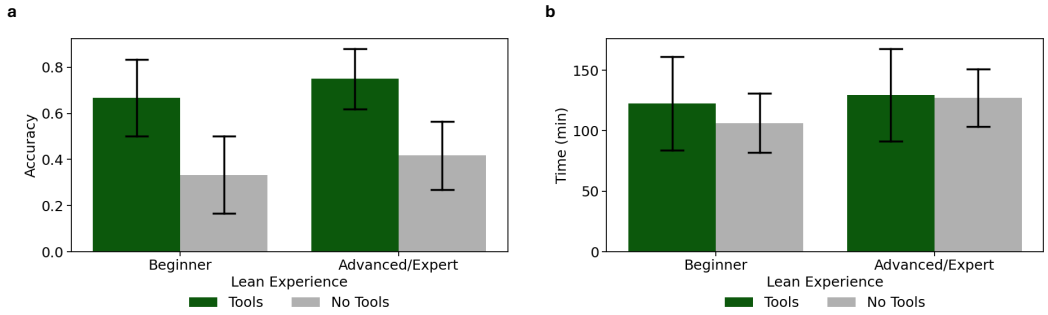


Figure 3: **Formalization performance by Lean experience. a,** Average accuracy based on whether participants self-reported as being Beginners ($N = 3$) or Experts ($N = 4$) with Lean; **b,** Average time (min) across problems. Error bars show standard error.

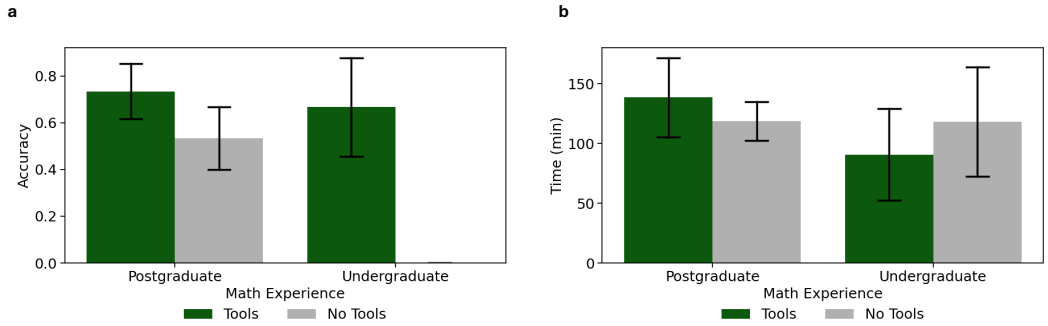


Figure 4: **Formalization performance by mathematics experience. a,** Average accuracy based on whether participants self-reported as being having studied mathematics up to or through the postgraduate level ($N = 5$) or undergraduate level ($N = 2$). Accuracy is decomposed into the problems where participants did or did not have access to tools (three problems per participant per category).; **b,** Average time (min) across problems. Error bars show standard error. These analyses are descriptive and exploratory; we urge caution in any extrapolation due to the small sample size.

### A4.2 Finer-grained accuracy evaluations

Throughout the paper, we have focused on accuracy as measured by having both the statement and proof correct. We report the average accuracy per problem in Figure 5. However, a few participants were correct in their statement formalization but incorrect in the proof (or vice versa — formalized the statement incorrectly, but proved the statement correctly under that interpretation), or formalized only part of the proof. E.g., one proof that was marked as incorrect did get some of the way there according to the grader on Problem N:

"Only one direction $(12x \equiv 9 \pmod{15} \implies x = 5n + 2)$ is formalized. That direction is correctly formalized and proved. ZMOD is used for modulus in the assumption, $5n + 2$ is explicitly written as the answer."

Whereas another participant (for Problem G) made a start at the problem but did not formalize the entire proof correctly according to the grader:

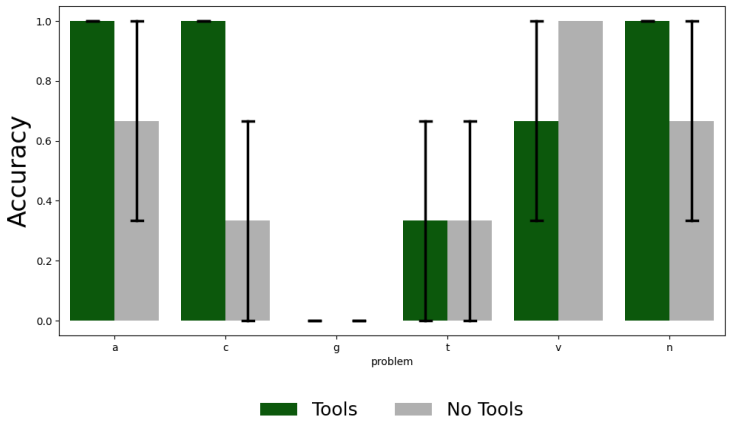


Figure 5: **Accuracy per problem, for participants with and without tool use.** Accuracy is averaged per problem. Error bars show standard error.

"Formalised a lot, but with some sorrys (beyond existence of D). Noted that showing the existence of D "seems hard" "

We are actively expanding further further exploratory analysis of a finer-grained human evaluation of the proofs. As before, three formal mathematics researchers from our team assessed each proof (one per proof). Future work can conduct a more expansive human audit of proofs.

### A4.3 Analyzing code structure

We also conduct exploratory analyses into the style of code participants provided, depending on whether they had access to tools or not. We observe that there may be some effect of AI-based tool use on overall proof length, with participants potentially producing — on average — longer formal proofs with AI-assistance (Figure 6a). However, this effect is variable at a per-problem level (Figure 6b) and moderated by whether participants even finished the proof.

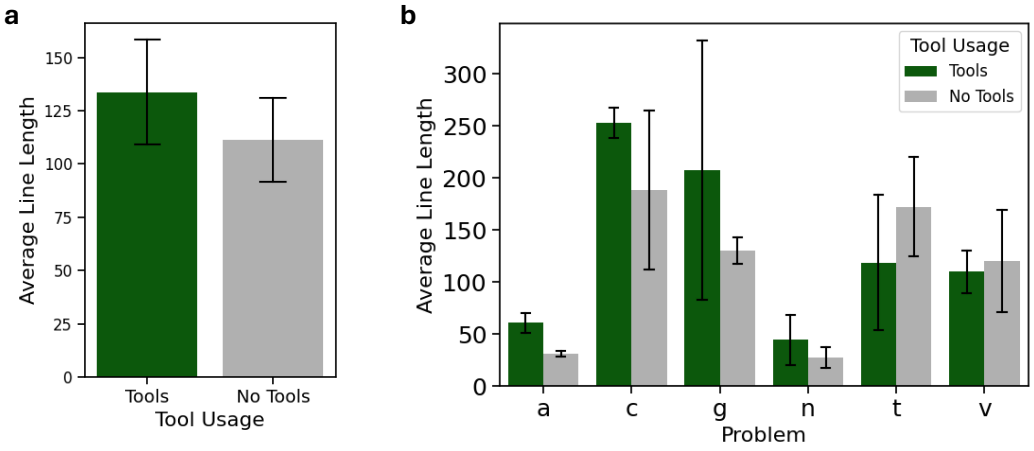


Figure 6: **Formalized proof length. a,** Average length of code (not including comments).; **b,** Average code length (not including comments) per problem.

### A4.4 Analyses into formalization processes from participant videos

Participants recorded themselves formalizing proofs, producing over 80 hours of videos. Two authors from our team manually watched and annotated a selection of videos to understand when and how

participants were using AI. We are actively extending our analyses for more rigorous coding over our full library of formalization videos.

Here, we offer a glimpse into the kinds of behavior different participant showed, with brief summaries of a few of the formalization videos. These behaviors can be characterized in varying reliance strategies in a user's AI workflow (a la Jorgensen et al. [2025]). As described in the main text, one group of participants predominantly formalized the proofs themselves, only lightly engaging AI tools (often, GitHub Copilot in-line). For instance, two participants who could be classified as "human formalizers with AI assistance" each sparingly used GitHub Copilot as part of solving Problem V. One participant listed themselves as an expert formalizer in Lean; the other, as a Beginner Lean user. Both participants used comments to "prompt" GitHub Copilot and regularly overrided Copilot suggestions. One participant (with Lean experience) used Copilot in more of a coarse-to-fine fashion, working on some functions, templating them and using Copilot to fill in details, particularly with substantive and repetitive pattern matching. Relatedly, the participant with less Lean experience also regularly rejected Copilot suggestions; they can be seen frequently pausing and deliberately reading the Copilot suggestions, before occasionally accepting them — suggestive of the intentional nature of their proof process. Both participants solved Problem V correctly.

In contrast, another participant could be categorized as behaving in a way of "AI formalization, with human help." This participant started one problem by opening up two chat windows: one ChatGPT and one Claude. The participant then pasted in the problem PDF into each window, and proceeded to engage in a copy-output/copy-error message cycle into the models over the course of an hour. While the participant shifted to using GitHub Copilot in the latter half of the proof, even there, the participant did not seem to engage in as much critical thinking. There was a span of five minutes where the participant went back and forth with GitHub Copilot saying "finish this," having the model generate a series of local code snippets (the participant queried $8$ times, accepting $7$ of them outright). This led to a substantial amount of the code not being generated by the participant; the participant seemed to recognize that the code did not match their style and noted in a comment at the top near the end of the video ("`i'm sorry this is so ugly`"). While this participant formalized the problem correctly, they took about $3\times$ as long as the average participant for this problem ("Problem N").

Lastly, one participant (who used ten different tools over the course of the study) was a highly varied user, engaging many different kinds of AI tools for different parts of their formalization process—yet demonstrated regular and substantial agency in how they used the tools (e.g., regularly declined in use of AI-based tools over the course of the problem). For instance, on one problem ("Problem N"), this participant started by pasting the informal statement into Kimina to formalize, pasted into VSCode and manually edited. They then went to look at the solution, opened GitHub Copilot chat to get a tailored response using a particular tool, querying "`Define the set of all numbers of the form x = 5n + 2 for some integer n, in Lean, using ZMOD.`" They then further clarified their interest in ZMOD: "`I'd like to use the ZMOD notation. An example is: [...]`" and copied over the solution. They then asked for "`suggest tactics`" from Lean Copilot, looked through and chose the second suggestion, and proceeded to again search for tactics (this time choosing the third suggestion). They then accepted a major block of GitHub-generated code, before moving to Claude Sonnet to rewrite part of their code ("`I'd like to prove this just using the definition of ZMOD. Could you rewrite this proof?`"). They then proceeded to delete a large block of AI-generated code in their proof, went back to Kimina and had the model generate the full formal proof, given the informal, and made manual tweaks. However, $30$ seconds later, the participant proceeded with a series of major deletions, essentially restarting the entire formalization process and wrote the rest of the proof primarily unassisted (ending up with a much simpler proof that used the `omega` tactic). This is a case where the participant heavily engaged AI, but ended up, in their final proof, relying very little on the actual AI-generated code.

## A5 Additional participant self-reported thoughts on AI-based assistance for Lean

Participants were asked the following questions after their study (in addition to the question of what tools they used, as presented in Section A3). We include participants' responses verbatim.

### A5.1 "What do you like most about AI-based tools for formalizing?"

- *"I like that AI tools can help me formalize "obvious" results that I have trouble formalizing on my own. I believe that AI can also help familiarize me with new topics and good ways to formalize those topics."*
- *" The can (often) take care of tedious low-level details. They also offer better "library search" than Mathlib's search function."*
- *"When I'm certain of an approach, I like how AI-based tools fill in the unimportant/rote details. For example, if I need to open a file in Python, rather than looking up the syntax, I can just leave a comment that I want to open a file, and Copilot will fill in the code for me. When it comes to Lean formalization, Copilot copies parts of proofs from elsewhere in the file, fetch names of lemmas from Mathlib, and fill in assumptions for lemmas, which saves me time. I rely less on its proof completion, especially for nontrivial proofs. But it works pretty well on straightforward induction proofs or simp-filled proofs."*
- *" Discovering definitions and lemmas in the library is significantly easier thanks to AI-based search and autoformalization tools."*
- *" I also find it helpful to load relevant Mathlib files into the GitHub Copilot chat context to ask questions that would otherwise take me a significant amount of time to find answers to."*
- *" I like that it helps to get me started to have a framework of understanding the problem and giving me ideas to approach the formalization, even if it is unable to give me the full thing. It is also helpful in pointing me to other resources that I can reference"*
- *" leansearch gives me fast lookup with low cognitive overhead. Chatgpt/Gemini sometimes are very good at pointing out mistakes that I've made in formulating the problem."*
- *" Using Copilot can save time."*

### A5.2 "What frustrates you most about the current state of AI-based tools for formalizing?"

- *" Sometimes the code is wrong or doesn't do what I want it to do (this is sometimes fixed by giving it more prompts)"*
- *"Miscalibrated confidence in the answers. Even if the AI "doesn't know" it will output something plausible but wrong, which is more frustrating than not getting an answer at all. Also AI-based tools seem to be much worse for combinatorics and geometry than for algebra or analysis."*
- *"When it's wrong, it's really wrong. Sometimes Copilot suggests a (broken) "proof" of a theorem that will in reality turn out to be much longer or more complex. This requires me to cancel the suggestion and fill in the proof myself (which is what I would have done anyway, but the extra keystrokes to dismiss it can be distracting)."*
- *"I find that there aren't enough tools specifically tailored to the Lean environment. There's been a lot of exciting progress in the AI for mathematics space in the recent years, but not all of that progress has translated into better tooling for the Lean community. Even when tools exist, they can be difficult to get working, requiring a complicated set-up involving external dependencies or an API key configuration (although it's completely understandable why this is the case). An ideal situation would be to have a VS Code extension that can interact with the current Lean session and assist the user with various formalization tasks. Another option would be to have AI-based tools and tactics shipped by default with Mathlib or Lean, but this has the disadvantage of being "opt-out" rather than "opt-in"."*
- *"There's still quite a bit of hallucination, calling tactics that don't exist for example. Even giving it reference to the Mathlib documentation (by copying and pasting) didn't really help"*
- *"1. Mathlib seems to evolve faster than models can keep up. 2. Chat models are often very confident about wrong things – even stuff like "here's a much shorter simpler proof!" and it's actually longer and has error. 3. Models are expensive to use. 4. Stuff like AlphaProof is not available to the public."*
- *"The practice of using such tools does not match strong results reported in research benchmarks; they are mostly unable to prove even very simple things, including when given access*

*to the Lean goal state via MCP. Also, most of the tools are challenging to set up for a local development environment."*

### A5.3 "What do you envision any future workflow pattern for how you may go about discovery new proofs / formalizing them? Where would you want tools that help you? Where would you NOT want AI-based tools to be used?"

- *"Better code generation, maybe also suggest more advanced techniques (like using tactics)"*
- *"Reduction of "bullshitting" (giving plausible but wrong answers) when they "don't know". Also, expansion of application domains to be more general than algebra and analysis."*
- *"Automatically fill in lemmas/definitions as I work on a larger theorem! For example, it would be great if I could leave a lemma with "sorry," and then while I use the lemma in the main theorem, Copilot (or whatever tool) could copy my file up to the lemma and work with it in the background until it finds a proof. Right now, all interactions are human-driven. (Of course, this will introduce more problems - what if the definition/proof is too verbose, or not quite what I want. Then I need to manually clean up, which is still faster than writing the proof myself, but can be more frustrating/tiring. But if it's tuned right, this would save time!)"*
- *"Good autoformalization and proof completion models that integrate seamlessly into the Lean editor would be useful to have."*
- *"A generally Lean-aware chatbot that can alert the user to Lean formalization conventions (like writing 'a < b' instead of 'b > a') explain how a mathematical concept is defined in Mathlib, and how it is meant to be used generate outlines for proof formalization suggest a convenient way to define a mathematical concept in Lean (for example, by combining existing definitions in the library or by modelling it based on an existing definition of a similar concept) suggest new lemmas to formalize suggest attributes to tag lemmas with (like* `[simp]`, `[aesop]` *or* `[grind]`*) mention relevant lemmas, tactics or domain-specific proof formalization strategies improve and optimize proofs give high-level feedback on a Lean document (like a Mathlib reviewer) would make formalization significantly easier and more accessible, in my opinion."*
- *"Being able to generate tactics that actually exist. Of course ideally if it can do end to end formalization of proofs that would be amazing, but in the meantime being able to more reliably tackle smaller subproblems that contribute towards the larger goal"*
- *"I want tools that can simplify my existing proofs."*
- *"Autoformalization from natural language, and automatic theorem proving."*

### A5.4 "What would you want to see most in future AI-based tools for formalizing proofs?"

- *"If I already have an idea of the solution, I would consult a large language model for ideas on how to go about formalizing parts of the problem. The AI tool could suggest tactics and theorems to use, and maybe give a working formalization. I didn't try asking it for suggestions on how to solve a problem (because we already had solutions), but I think that would be cool."*
- *"I mostly want AI to take care of tedious low-level details of a proof; I'm happy to do the main structure."*
- *"It would be great if I could quickly outline the shape of the proof, such as ":= by – induction on n, using core lemmas [X, Y, Z]" without explaining my thought process too deeply, and having the skeleton filled in. The closer a tool can allow me to type in only the key pieces, and having the rest of the proof/syntax get filled in, the better."*
- *"I don't like AI-based tools writing too much code for me - it makes me lose my mental model of how the program functions. But if an AI-based tool can fill in an entire proof for me, and the proof is succinct and compiles quickly, then my mental model stays intact. The problem is that AI-based proofs are often not succinct or efficient, and so I have to trade-off writing the proof correctly from the start, or editing a bad proof into a good one."*
- *"I think a nice workflow for formalization would be one where you could describe a mathematical idea or proof gradually to a chatbot and get suggestions for ways to formalize it*

*in Lean. In this interaction, the human only needs to have a passing familiarity with the Lean syntax, and a bulk of the work will be done by the machine. To keep the response time reasonable, ideally the chatbot should not spend an enormous amount of time autonomously formalizing or discovering proofs, but instead should develop these in conversation with the human."*

- *"I think I would use it quite regularly for generating small tactics that help me do the mundane stuff, but overall I probably would still have to guide the overall structure of the proof and think of next steps"*
- *"More real-time feedback, in an unobstrusive way. Interactive theorem proving strongly relies on using goal states and other feedback from the theorem prover. AI-based workflows should integrate better with this information and use the information rather than trying to one-shot the entire proof."*

## A6 Participant instructions

Participants were given the following instructions and told which order to do their TOOLS or NO-TOOLS weeks, as outlined above. The instructions included links to the consent form and post-survey. Participants were sent instructions via email. Following a few questions from participants, we sent a clarifying email on tools, as outlined below.

**Instructions**

1. Please formalize the following three problems over the course of this week. This includes both the formalization of the problem statement and the solution.
2. When formalizing the problem statement, aim to stay as close as possible to the original statement.
3. For the solution, you may either follow the provided approach or develop your own, as long as it leads to a correct and complete formal proof. We have formalized the solutions ourselves, so you can rest assured that the problems are formalizable without undue effort.
4. Don't worry if you cannot finish any problem. You can work on the problems in any order, including starting one problem, stopping, and starting another before coming back to a problem. We care about your process! We what do ask you, is to make sure that you submit *a single screen recording per problem* and not mix work on multiple problems within a single screen recording.
5. If this is assigned for your **NO-TOOLS week**, please DO NOT use any external tools for assistance in formalizing the proof. If you use an IDE, such as Visual Studio Code, make sure to also turn off any GitHub Copilot or related tools within that environment.
6. If this is your **TOOLS-ALLOWED week**, you are allowed to use any AI-based tool(s) for assistance (this includes LLMs, tools specific to the Lean ecosystem, as well as any other tools you might find useful).
7. We kindly ask that whenever you are formalizing the problems, at any time, please record your ENTIRE screen (e.g., using QuickTime video, vokoscreenNG or any other tool you might find useful). We ask that you record *all* interactions related to your problem solving; this include web browsing (if you use Google search, or LLMs), as well as any other software that assists you in formalizing. If you ever forget to record your screen, please note this in the material that you submit back as part of the study. While it is possible that you may forget, if you do this more than twice, you may not be paid. Do not worry if you accidentally screen-record personal matters - there is an option to black these out at the end, to exclude any frames that reveal personal information.

**Submission Details**

At the end of the study, please **submit a zip** with the following information:

- Your formalized proofs for each problem – in a file labeled "{your-last-name}{your-first-name}{problem-id}.txt"
- Any screen recording for that problem labeled "{your-last-name}{your-first-name}-{problem-id}-{record-idx}.mp4" for each record-idx (between 0 to however many clips you took for that problem)
- A Google form with your post-survey responses, including a general questionnaire about your prior mathematics and Lean experience.
- If you use scratch paper at any time during the study, we ask that you screenshot or upload the scratchpaper with the label "'{your-last-name}{your-first-name}{problem-id}-{page-idx}.png/pdf"

Any / all of the information may be released as part of the data collected in this work. However, we will scrub any names so the data is anonymized. We will then stitch these videos together. We will make the screen recordings public (and anonymized) as part of the study contributions; however, you will be given a chance to "mask" any part of the recording before publication of your accidentally screen-recorded material that should not be made public.

By participating, you agree to having data shared and affirming that you are at least 18 years old. We will communicate payment details after the study. **Please make sure you have filled out the consent form (also emailed) before you begin.** You only need to fill out the consent form once.

**Clarification email excerpts**

We wanted to provide some clarification on what counts as a "tool" for the study, following a great question by a participant. In our instructions we said: "If this is assigned for your NO-TOOLS week, please DO NOT use any external tools for assistance in formalizing the proof." but we realized it might not be clear to everyone where exactly to draw the line what a tool is and what not - we received some questions from som of you about this, so we wanted to write an email to everyone to make sure we are all aligned: If you use Google search, or any other "tool" that is neither specifically designed to aid math and formalization, this counts as NO TOOL, the only exception being LLMs, which do count al TOOL (not all LLMs were are designed to aid math and formalization, but still provide significant support, which is why we categorized them as tools). We don't want write a list of potential tools here, since we do not want to bias you. But we want to encourage you nonetheless to ask us beforehand, if you are unsure, whether something counts as tool or not - PLEASE ASK US FIRST.

You are not expected to spend more than 12 hrs on the problems over the two weeks. We recognize some problems are more difficult than others, and may take longer. You may not be able to finish all problems in 12 hrs! We encourage you to make sure you at least attempt all problems (rather than spend all your time on a subset and miss out on trying one or more problems). Of course if you want to spend more than 12 hours, awesome!, but that is not expected at all.