ModelWisdom: An Integrated Toolkit for TLA^{+} Model Visualization, Digest and Repair (short tool paper)

Zhiyong Chen[⋆1,2][0009-0005-2555-5728], Jialun Cao[*3,4][0000-0003-4892-6294], Chang Xu[⋆⋆1,2][0000-0002-6299-4704], and Shing-Chi Cheung[3,4][0000-0002-3508-7172]

[1] State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China
[2] School of Computer Science, Nanjing University, Nanjing, China
zhiyongchen@smail.nju.edu.cn, changxu@nju.edu.cn
[3] Department of Computer Science and Engineering, The Hong Kong University of Science and Technology, Hong Kong, China
[4] Guangzhou HKUST Fok Ying Tung Research Institute, Guangzhou, China
jialuncao@ust.hk, scc@cse.ust.hk

Abstract. Model checking in TLA+ provides strong correctness guarantees, yet practitioners continue to face significant challenges in interpreting counterexamples, understanding large state-transition graphs, and repairing faulty models. These difficulties stem from the limited explainability of raw model-checker output and the substantial manual effort required to trace violations back to source specifications. Although the TLA+ Toolbox includes a state diagram viewer, it offers only a static, fully expanded graph without folding, color highlighting, or semantic explanations, which limits its scalability and interpretability. We present ModelWisdom, an interactive environment that uses visualization and large language models to make TLA+ model checking more interpretable and actionable. ModelWisdom offers: (i) Model Visualization, with colorized violation highlighting, click-through links from transitions to TLA+ code, and mapping between violating states and broken properties; (ii) Graph Optimization, including tree-based structuring and node/edge folding to manage large models; (iii) Model Digest, which summarizes and explains subgraphs via large language models (LLMs) and performs preprocessing and partial explanations; and (iv) Model Repair, which extracts error information and supports iterative debugging. Together, these capabilities turn raw model-checker output into an interactive, explainable workflow, improving understanding and reducing debugging effort for nontrivial TLA+ specifications. This tool is available: `https://github.com/ModelWisdom/ModelWisdom`. A demonstrative video can be found at `https://www.youtube.com/watch?v=plyZo30VShA`.

⋆ These authors contributed equally to this work.
⋆⋆ Corresponding author.

Keywords: Model Checking · Large Language Model.

1 Introduction

Model checking [3,10] in TLA+ [18,21,17] provides strong correctness guarantees, yet practitioners often struggle to interpret the large state-transition graphs and counterexamples produced by the model checker. While the TLA+ Toolbox [16] includes a built-in state diagram viewer, it displays graphs in a static, fully expanded form without node or edge folding, colorized violation cues, or semantic explanations. As a result, users need to sift through dense graphs manually, trace transitions back to the code, and connect violation states to the temporal properties they violate – tasks that become increasingly difficult as models grow in size and complexity.

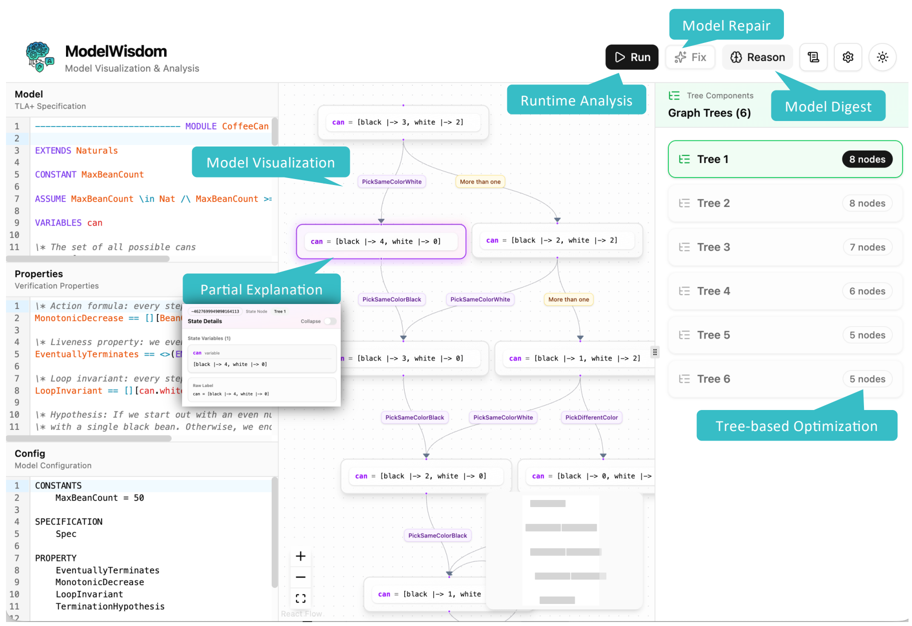


Fig. 1: Interface of ModelWisdomwith Highlight Features

These challenges echo long-standing usability problems in model checking tools [11,12], where understanding diagnostic output often becomes harder than writing the specification itself. Meanwhile, recent advances in large language models (LLMs) open new opportunities to turn these verification artifacts into interactive, interpretable representations. While prior work has used LLMs for specification synthesis [24,20,14,26,1,7] or proof assistance [25,23,22], little attention has been given to enhancing the visibility, navigability, and explainability of model checking. Complementing algorithmic improvements, improving the human–tool interface is increasingly recognized as a critical step toward broader industry adoption of formal verification.

To address these gaps, we present ModelWisdom, an LLM-assisted interactive environment that enhances the interpretability, explorability, and repairability of TLA+ model-checking results. ModelWisdom integrates visualization, structural abstraction, automated debugging support, and model summarization into a unified workflow. Its contributions are fourfold:

✔ **Model Visualization & Explainability**: Colorized violation nodes, click-to-navigate edges linking to source code, and direct mapping from violating states to broken properties.

✔ **Performance-Oriented Optimization**: Scalable exploration through tree-structured graph layouts, node and edge folding, and automated graph compaction, as well as support for identifying homogeneous node clusters that can guide parameterized verification [27,13,2,8,9].

✔ **Model Digest**: Graph summarization, preprocessing, and selective subgraph explanation using LLM-guided analysis.

✔ **Model Repair Assistance**: Structured extraction of error information, support for both single-pass and multi-pass repair strategies, and an interactive history panel that records intermediate fixes and enables systematic, traceable debugging of faulty specifications.

2 ModelWisdom

ModelWisdom is open source on Github (`https://github.com/ModelWisdom/ModelWisdom`), comprising approximately 7,000 lines of fully typed TypeScript and Python code. It consists of two main components: the primary web application and a server responsible for executing the TLC (i.e., TLA+ Checker). The supporting LLMs include GPT and Claude families (e.g., gpt-4 and Claude 3.7 Sonnet), and can be set on the website.

2.1 Model Visualizer

In this paper, we introduce ModelVisualizer, a tool designed to enhance both the visualization and performance of state transition graph analysis in the context of TLA+ verification. ModelVisualizer enables users to input a TLA+ model, specify properties, and configure parameters for verification via TLC. Upon completion, ModelVisualizer automatically collects the resulting state transition graph and tightly integrates it with the underlying TLA+ code.

One of our insights is to bridge model code and interactive exploration: ModelVisualizer highlights violation nodes with distinct colors, provides click-through navigation from transitions directly to the corresponding lines of TLA+ code, and offers a direct mapping between violating states and the specific properties they break. This approach allows users to quickly trace errors from state graph nodes to concrete model definitions. Similar visualization features are also

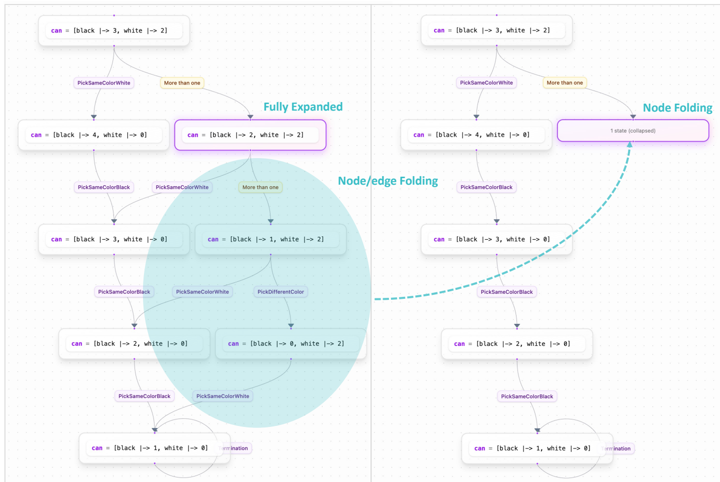


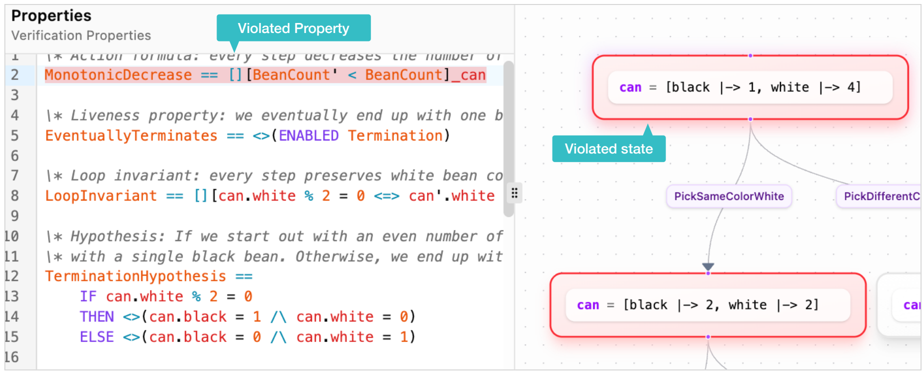
Fig. 2: Interface of ModelWisdomwith node folding

supported by the TLA+ Toolkit [16] and UPPAAL [19,4,6,5], while ModelWisdomwas empowered by LLMs, offering explaining and repairing features.

To address the challenge of visualizing large and complex state graphs without sacrificing performance, ModelVisualizer incorporates several graph optimization techniques. These include tree-based structuring for hierarchical clarity, node and edge folding to reduce visual clutter, and graph compaction for efficient rendering. The core idea behind these optimization strategies is lazy rendering, where only the currently interacted portions of the graph are generated and displayed on demand.

Demonstration 1. Figure 1 demonstrates the interface of ModelWisdom under the TLA+ official example of `CoffeeCan`[5]. The interactive state graph is visualized in the middle of the application after successful runtime analysis. Meanwhile, the ModelVisualizer only renders one tree at a time for performance, and the users can toggle the trees on the right-handle panel. Figure 2 demonstrates the node-folding technique, which can be utilized to track state transitions incrementally, or to collapse irrelevant paths for enhanced rendering efficiency. Figure 3a demonstrates the highlight of violated states and properties, where we deliberately modify the postcondition statement of the `PickSameColorWhite` action from `can' = [can EXCEPT !.black = @ + 1, !.white = @ - 2]` to `can' = [can EXCEPT !.black = @ + 1, !.white = @ - 1]`.

[5] `https://github.com/tlaplus/Examples/blob/master/specifications/CoffeeCan/CoffeeCan.tla`



(a) Violated Property Visualization

(b) ModelRepair Dialog

(c) ModelDigest Dialog

Fig. 3: Interface of ModelWisdomwith highlighting violated properties, ModelRepair, and ModelDigest

2.2 Model Digest

In this paper, we offer ModelDigest to explain TLA+ models as a whole or in part. ModelDigest enables users to highlight specific code segments of interest directly within the editor, facilitating targeted explanations. Prior to engaging LLMs, ModelDigest automatically extracts key structural information from the state graph, including initial and terminal states, cycles, and the most frequent transitions—to provide essential context. Leveraging this information, LLMs generate comprehensive explanations of the TLA+ model, covering aspects including the overall model structure, state variables, constants, actions, transitions, and invariants.

Demonstration 2. Figure 3c demonstrates the explanation of ModelDigest for the repaired model in Figure 3a. ModelDigest provides LLMs with the complete TLA+ specification as well as a summary of the state graph, including the initial state (`can = [black |-> 0, white |-> 5]`) and the terminal state (`can = [black |-> 0, white |-> 1]`). Finally, ModelDigest's explanation includes the semantics of each action in natural language. For example, for the action `PickSameColorBlack`, the explanation is as follows: (1) Removes two black beans and adds one black bean back; (2) Preconditions: more than one bean in total and at least two black beans; (3) Postcondition: one black bean is removed.

2.3 Model Repair

In this paper, we propose ModelRepair. It is designed to facilitate the debugging and iterative repair of TLA+ models, inspired by the TLA+ debugger [15]. ModelRepair automatically extracts error messages generated during TLC checking and leverages LLMs to repair the TLA+ model when syntax errors or violations of properties are detected. The tool supports both single-pass and multi-pass repair workflows. In the single-pass repair mode, ModelRepair provides an interactive, one-shot repair experience, where the user can review and accept the suggested fix for a given error. In contrast, the multi-pass repair workflow enables iterative debugging by automatically engaging in multiple rounds of interaction between LLMs and TLC. During this process, ModelRepair maintains a history panel that records each repair attempt, allowing users to track changes and understand the evolution of the model. The iterative process continues until the model is free of errors or a predefined repair limit is reached.

Demonstration 3. Figure 3b illustrates the repair and execution history of ModelRepair in the case of the invalid models in Figure 3a. ModelRepair provides the LLMs with comprehensive information regarding the incorrect specification as well as the extracted error message. Subsequently, ModelRepair successfully corrects the erroneous statement `!.white = @ - 1`, reverting it to the correct one `!.white = @ - 2`. On the right side, detailed logs are provided, including the specific content of the LLM-driven dialog as well as the repaired model.

3 Future Work

In future work, we plan to extend ModelWisdom beyond TLA+ to support a broader class of model checking languages and verification frameworks. Many of the challenges addressed by ModelWisdom, such as interpreting counterexamples, navigating large state-transition graphs, summarizing behaviors, and assisting with repair, are shared across specification languages like Promela, Alloy, NuSMV, and UPPAAL and other state-transition models. By generalizing our visualization pipeline, LLM-assisted summarization modules, and repair interfaces, we aim to create a unified, language-agnostic environment that brings explainability and interactive assistance to diverse verification ecosystems. This expansion will not only increase the tool's applicability but also deepen our understanding of how LLMs and visualization techniques can systematically enhance the broader practice of formal verification.

4 Conclusion

ModelWisdom enhances the usability and interpretability of TLA+ model checking by integrating scalable graph visualization, semantic summarization, and automated repair assistance. By combining interactive state-space exploration with LLM-driven explanations, the tool lowers the cognitive barrier to understanding counterexamples and accelerates the process of correcting faulty specifications. These capabilities go beyond the existing TLA+ Toolbox, offering colorized violation highlighting, collapsible graph structures, and property-aware diagnostics. Together, these features make ModelWisdom a practical and powerful companion for engineers and researchers working with complex TLA+ specifications.

5 Acknowledgments

We would like to thank the reviewers for their insightful comments. This work is supported by the National Natural Science Foundation of China (Grant No. 92582201), the Hong Kong SAR Research Grant Council/General Research Fund (Ref No. 16206524), and the Hong Kong SAR Research Grant Council/Theme-based Research Scheme (Ref No. T41-517/25-N). The authors would like to thank the support from the Collaborative Innovation Center of Novel Software Technology and Industrialization, Jiangsu, China.

References

1. Alhanahnah, M., Rashedul Hasan, M., Xu, L., Bagheri, H.: An empirical evaluation of pre-trained large language models for repairing declarative formal specifications. Empirical Software Engineering **30**(5), 149 (2025)
2. Arons, T., Pnueli, A., Ruah, S., Xu, Y., Zuck, L.: Parameterized verification with automatically computed inductive assertions? In: International Conference on Computer Aided Verification. pp. 221–234. Springer (2001)
3. Baier, C., Katoen, J.P.: Principles of model checking. MIT press (2008)
4. Behrmann, G., David, A., Larsen, K.G.: A tutorial on uppaal. Formal methods for the design of real-time systems pp. 200–236 (2004)
5. Behrmann, G., David, A., Larsen, K.G.: A tutorial on uppaal 4.0. Department of computer science, Aalborg university **1**(1), 1–48 (2006)
6. Behrmann, G., David, A., Larsen, K.G., Håkansson, J., Pettersson, P., Yi, W., Hendriks, M.: Uppaal 4.0. In: QEST. vol. 6, pp. 125–126 (2006)
7. Cao, J., Lu, Y., Li, M., Ma, H., Li, H., He, M., Wen, C., Sun, L., Zhang, H., Qin, S., Cheung, S.C., Tian, C.: From informal to formal – incorporating and evaluating LLMs on natural language requirements to verifiable formal proofs. In: Che, W., Nabende, J., Shutova, E., Pilehvar, M.T. (eds.) Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). pp. 26984–27003. Association for Computational Linguistics, Vienna, Austria (Jul 2025). `https://doi.org/10.18653/v1/2025.acl-long.1310`, `https://aclanthology.org/2025.acl-long.1310/`
8. Chen, X., Gopalakrishnan, G.: A general compositional approach to verifying hierarchical cache coherence protocols. Tech. rep., Technical Report. UUCS-06-014, School of Computing, University of Utah (2006)
9. Chou, C.T., Mannava, P.K., Park, S.: A simple method for parameterized verification of cache coherence protocols. In: International Conference on Formal Methods in Computer-Aided Design. pp. 382–398. Springer (2004)

10. Clarke, E.M.: Model checking. In: International conference on foundations of software technology and theoretical computer science. pp. 54–56. Springer (1997)
11. Havelund, K., Rosu, G.: Monitoring programs using rewriting. In: Proceedings 16th Annual International Conference on Automated Software Engineering (ASE 2001). pp. 135–143. IEEE (2001)
12. Holzmann, G.J.: The model checker spin. IEEE Transactions on software engineering **23**(5), 279–295 (1997)
13. John, A., Konnov, I., Schmid, U., Veith, H., Widder, J.: Parameterized model checking of fault-tolerant distributed algorithms by abstraction. In: 2013 Formal Methods in Computer-Aided Design. pp. 201–209. IEEE (2013)
14. Kogler, P., Falkner, A., Sperl, S.: Reliable generation of formal specifications using large language models. In: SE 2024-Companion. pp. 141–153. Gesellschaft für Informatik eV (2024)
15. Kuppe, M.A.: The tla debugger. In: International Conference on Software Engineering and Formal Methods. pp. 174–180. Springer (2022)
16. Kuppe, M.A., Lamport, L., Ricketts, D.: The tla+ toolbox. arXiv preprint arXiv:1912.10633 (2019)
17. Lamport, L.: Specifying concurrent systems with tla+. Calculational System Design pp. 183–247 (1999)
18. Lamport, L.: Specifying systems (2002)
19. Larsen, K.G., Pettersson, P., Yi, W.: Uppaal in a nutshell. International journal on software tools for technology transfer **1**(1), 134–152 (1997)
20. Ma, L., Liu, S., Li, Y., Xie, X., Bu, L.: Specgen: Automated generation of formal program specifications via large language models. In: 2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE). pp. 16–28. IEEE (2025)
21. Merz, S.: The specification language tla+. In: Logics of specification languages, pp. 401–451. Springer (2007)
22. Qu, Y., Zhang, T., Garg, N., Kumar, A.: Recursive introspection: Teaching language model agents how to self-improve. Advances in Neural Information Processing Systems **37**, 55249–55285 (2024)
23. Song, P., Yang, K., Anandkumar, A.: Towards large language models as copilots for theorem proving in lean. In: The 3rd Workshop on Mathematical Reasoning and AI at NeurIPS'23 (2023)
24. Wen, C., Cao, J., Su, J., Xu, Z., Qin, S., He, M., Li, H., Cheung, S.C., Tian, C.: Enchanting program specification synthesis by large language models using static analysis and program verification. In: International Conference on Computer Aided Verification. pp. 302–328. Springer (2024)
25. Yang, K., Swope, A., Gu, A., Chalamala, R., Song, P., Yu, S., Godil, S., Prenger, R.J., Anandkumar, A.: Leandojo: Theorem proving with retrieval-augmented language models. Advances in Neural Information Processing Systems **36**, 21573–21612 (2023)
26. Zhao, M., Tao, R., Huang, Y., Shi, J., Qin, S., Yang, Y.: Nl2ctl: automatic generation of formal requirements specifications via large language models. In: International Conference on Formal Engineering Methods. pp. 1–17. Springer (2024)
27. Zuck, L., Pnueli, A.: Model checking and abstraction to the aid of parameterized systems (a survey). Computer Languages, Systems & Structures **30**(3-4), 139–169 (2004)