Bridging LLM Planning Agents and Formal Methods: A Case Study in Plan Verification

Keshav Ramani
J.P. Morgan AI Research
New York, USA
keshav.ramani@jpmchase.com

Vali Tawosi
J.P. Morgan AI Research
London, UK
vali.tawosi@jpmorgan.com

Salwa Alamir
J.P. Morgan AI Research
London, UK
salwa.alamir@jpmchase.com

Daniel Borrajo
J.P. Morgan AI Research
Madrid, Spain
daniel.borrajo@jpmchase.com

***Abstract*—We introduce a novel framework for evaluating the alignment between natural language plans and their expected behavior by converting them into Kripke structures and Linear Temporal Logic (LTL) using Large Language Models (LLMs) and performing model checking. We systematically evaluate this framework on a simplified version of the PlanBench plan verification dataset and report on metrics like Accuracy, Precision, Recall and F1 scores. Our experiments demonstrate that GPT-5 achieves excellent classification performance (F1 score of 96.3%) while almost always producing syntactically perfect formal representations that can act as guarantees. However, the synthesis of semantically perfect formal models remains an area for future exploration.**

***Index Terms*—Plan Verification, Formal Methods, LLM for Plan Verification**

I. Introduction

Large Language Models (LLMs) excel at a wide range of tasks but often lack formal assurances of output correctness. Most evaluations rely on empirical results, without rigorous checks for reliability. We propose a safer design approach: using LLMs to convert natural language into structured formats, then applying classical, deterministic AI methods for reasoning. This leverages LLMs' strengths in language processing and the reliability of deterministic techniques, especially formal verification, which is widely used in safety-critical systems to provide guarantees.

To integrate these methods, we focus on plan verification. While LLMs are used for planning in areas like agent systems and travel, their outputs lack formal guarantees. Our framework aims to add such guarantees, improving reliability and extending to software verification. Prior work has used symbolic reasoning tools, such as SMT solvers, to verify code [1], and model checking to ensure distributed system correctness [2]. Our approach advocates translating any language input into a formal representation, then applying verification techniques for robust validation.

In our current work, we explore how translating plans to a formal model with LTL [3] specifications can unlock the potential of formal verification. Given the absence of datasets that directly evaluate this setting, we adapt the PlanBench [4] plan verification task to align with our objectives. While the original task necessitates identifying specific reasons for plan failures, our approach requires only discerning between valid and invalid plans. Parsing errors encountered during translation are classified as `unknown`. This work would particularly be of interest to Planning agents, which is an emerging area in Agentic software engineering. Future endeavors may choose to translate to alternative representations, embodying the philosophy that translating to different structured representations can facilitate various tasks, such as using PDDL [5] for planning, among others. The following are our contributions:

- **A Framework for LLM-Driven Formal Verification of plans** that leverages LLMs to translate natural language plans into formal models (Kripke Structures [6]) and specifications (LTL), automating model checking and formal verification of plan validity.
- **Empirical Evaluation of LLMs for Plan Verification** through a comprehensive experimental study comparing GPT-4o and GPT-5 on the simplified PlanBench task, reporting accuracy, precision, recall, and F1 for both formal verification and baseline LLM judgment.

II. Related Work

Multi-agent LLM frameworks like ALMAS [7] have advanced end-to-end software engineering, but concerns remain about the reliability of LLM-generated code, as highlighted by CodeMirage [8]. Automated code assessment using LLMs has also been studied [9], emphasizing the need for strong verification methods.

Recent studies have combined LLMs with formal verification and automated planning. VeCoGen [10] and Lemur [11] show LLMs' potential for automating code generation and verification. Jha et al. [12] and Hassan et al. [13] explore co-synthesis and mutation testing, demonstrating LLMs' role in guiding formal verification. Our work aligns with these efforts, focusing on plan verification but with broader implications for software verification, while leaving the combined synthesis area for future work.

FVEL [14] and VeriPlan [15] integrate LLMs with formal tools for interactive planning. Härer [16] and Cemri et al. [17] address specification and evaluation challenges in multi-agent LLM systems, while Crouse et al. [18] and Tihanyi et al. [19] discuss specification and vulnerability detection. Unlike VeriPlan, our method is simpler, avoids templates, and is evaluated on the PlanBench dataset rather than user studies.

GenPlanX [20] and "LLMs Can't Plan" [21] critique LLMs' planning abilities and highlight its challenges. Taxonomies

2151-0849/25/$31.00 ©2025 IEEE
DOI 10.1109/ASEW67777.2025.00018

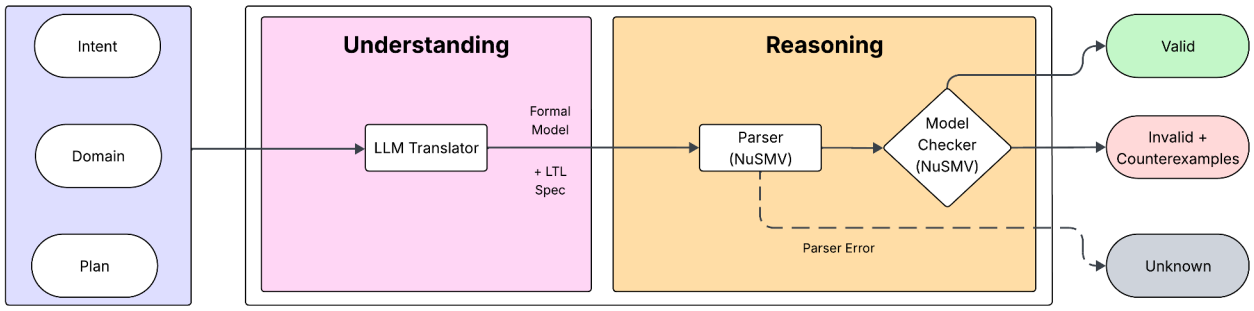

Fig. 1. Overview of the LLM-driven plan-specification alignment framework. The Planbench plan verification task contains natural language descriptions of goals, intents, and the environment along with the plan. The LLM addresses the task of understanding such natural language inputs and converting them to a Kripke structure (represented in the NuSMV format) and an LTL specification. For reasoning, we parse this output and provide it to the NuSMV model checker.

for coordination errors in multi-agent LLMs have also been developed [17]. Our approach aims to enhance LLM-generated plans with formal guarantees and could extend to automated software verification and agentic software engineering.

III. Methodology

Figure 1 shows an overview of the LLM-driven plan-specification alignment framework. This consists of two key components: Understanding and Reasoning which are both covered in this section in detail.

A. Understanding: Natural Language Plans to Formal Model

Seminal works have reframed the planning problem as a model checking problem [22]. We derive inspiration from these works, specifically utilizing Kripke structures [6] to represent plans, similar to how Souri et al. [2] used it to model the behavior of distributed software systems. By representing plans as Kripke structures, we effectively convert them into state transition systems, where each state corresponds to a unique system configuration, and transitions are modeled as actions that facilitate permissible changes between these configurations. In the context of model checking, a Kripke structure is defined as $K = (S, S_0, R, L)$, where S is a finite set of states, each representing a unique system configuration. The set $S_0 \subseteq S$ contains the initial states from which system executions begin. Transitions R are defined as actions that facilitate permissible changes between states. The labeling function $L : S \to 2^{AP}$ maps each state to a set of atomic propositions from AP, representing properties that hold in those states.

In order to verify plans, they are represented as sequences of states and actions, where each sequence $\pi = s_0, s_1, \ldots$ satisfies $s_0 \in S_0$ and $(s_i, s_{i+1}) \in R$ for all $i \geq 0$. This approach allows us to analyze and verify the correctness and effectiveness of plans in dynamic environments using model checking techniques. The capability of NuSMV[23] to model these Kripke structures, enables the verification of several formal properties (including goal reachability, safety and liveness), which are specified using Linear Temporal Logic (LTL) with a rich vocabulary of atomic propositions. The syntax of LTL is defined as

$$\varphi ::= p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid X\varphi \mid F\varphi \mid G\varphi \mid \varphi_1 U \varphi_2$$

where $p \in AP$. LTL semantics are path-based; for instance, $\pi \models G\varphi$ if φ holds at all positions along π. We leverage a large language model (LLM) to generate accurate translations of natural language plans to Kripke Structures in NuSMV and formal properties in LTL. This formulation allows our approach to be applied in a wide variety of dynamic environments, eliminating the traditional dependency on experts to craft specifications on models, as commonly observed in the field of formal verification.

Each aspect of the natural language plan is translated into a formal state-transition system in four key steps. *(1) Variables as Representations of Facts:* In the natural language description, objects and their properties (or facts) are described using terms. The state for each object is captured as a boolean variable. The relationships between objects are recorded via variables (e.g., object `a` inherits object `b`, etc). *(2) Initial Conditions*: The initial conditions described in the natural language plan are encoded into the initial state of the Kripke structure. These assignments are captured by the `init` command. *(3) Actions as State Transitions:* Each action in the natural language plan (e.g., Debug, Refactor, Compile) is described with preconditions and effects. Preconditions are conditions that need to hold prior to actions are included as guards in the conditional updates. Effects are facts becoming true or false after an action are specified by updating the related state variables. *(4) Sequencing Using the Stage Variable* A dedicated variable `stage` is used to sequence the actions. The natural language plan enumerates a series of actions, and the corresponding stages (e.g., `s0, s1, ... s10`) ensure that actions are processed in order. Each stage, representing a particular action, triggers its corresponding state transitions and effects.

Finally, the desired outcome is encoded using a LTL specification. In the natural language plan, the goal could be to achieve a set of variables being set to True. In LTL, this is expressed as $F(goal)$ (F symbolizes eventual satisfaction).

TABLE I
PERFORMANCE METRICS FOR THE SIMPLIFIED PLAN VERIFICATION TASK FROM THE PLANBENCH DATASET. ONE-SHOT REPRESENTS THE SCENARIO WHEN FORMAL MODELS ARE GENERATED AND CHECKED, W/O FV IS WHEN THE LLM DIRECTLY DETERMINES PLAN VALIDITY

LLM	Approach	Valid	Invalid	Unk. ↓	Accuracy ↑	Precision ↑	Recall ↑	F1 ↑	Time
GPT-4o	One-shot	28.08	37.31	34.61	52.06	59.19	45.54	51.48	15.8
GPT-5	One-shot	50.64	42.50	**6.87**	**95.89**	**99.44**	**93.34**	**96.30**	47.08
GPT-4o	w/o FV	40.40	59.60	0.00	80.37	97.00	67.99	79.95	7.86
GPT-5	w/o FV	57.64	42.36	0.00	99.59	99.65	99.65	99.65	15.27

B. Reasoning: Formal Foundations of Model Checking

One of the variants of model checking that NuSMV implements is known as Bounded Model Checking (BMC) [24]. BMC is a verification technique for finite-state systems that aims to find counterexamples to temporal logic properties within a specified bound k on the execution length. Given a Kripke structure K and a Linear Temporal Logic (LTL) property φ, BMC translates the search for a counterexample of length k into a propositional satisfiability (SAT) problem. Specifically, BMC constructs a formula ψ_k such that ψ_k is satisfiable if and only if there exists a path $\pi = s_0, s_1, \ldots, s_k$ satisfying the following conditions: $s_0 \in S_0$, $(s_i, s_{i+1}) \in R$ for $0 \leq i < k$, and $\pi \not\models \varphi$. The SAT solver is then employed to determine the satisfiability of ψ_k:

$$\psi_k := \text{Init}(s_0) \wedge \bigwedge_{i=0}^{k-1} \text{Trans}(s_i, s_{i+1}) \wedge \neg \text{Prop}_{\varphi}(s_0, \ldots, s_k)$$

In this formulation, $\text{Init}(s_0)$ encodes the initial states, $\text{Trans}(s_i, s_{i+1})$ encodes the transitions between states, and $\text{Prop}_{\varphi}(s_0, \ldots, s_k)$ encodes the negation of the property φ over the path. If ψ_k is satisfiable, the resulting assignment provides a counterexample of length k.

IV. RESULTS

We evaluate our framework using the simplified PlanBench verification task. For each planning problem, the LLM generates a NuSMV model and LTL property, which are validated using the NuSMV model checker. The output is categorized as `valid` (SAT), `invalid` (UNSAT), or `unknown`. We compare the formally verified output with the baseline LLM judgment to assess the performance trade-offs involved in approximating a formal model of the plan.

In this study, we explore a single approach to implementing this framework: providing a one-shot example to the LLM to facilitate translation into NuSMV and LTL. We conduct experiments with two LLMs, GPT-4o and GPT-5. GPT-4o is configured with a temperature of 0 to ensure deterministic outputs. For GPT-5 it is to be noted that the temperature parameter is no longer supported, and the reasoning effort parameter is set to 'low'.

Outputs marked as `unknown` are excluded from the metric calculations, but are separately reported. The treatment of `unknown` outputs in evaluation can significantly affect reported metrics and the interpretation of verification results. Counting unknowns as valid can inflate overall accuracy and recall, potentially overstating the system's reliability. Treating unknowns as invalid penalizes the system for each non-adjudicated case, leading to lower accuracy and recall. Excluding unknowns from metric calculations offers a clearer assessment of performance on adjudicated cases.

We compare these results against the ground truth labels and report Accuracy, Precision, Recall, and F1-score for each LLM, as presented in Table I. The decision to prioritize precision or recall in verification should be guided by the specific application domain and its associated risk profile. High precision guarantees reliability, though it may result in reduced coverage (lower recall). Prioritizing recall is appropriate for exploratory, creative, or research-oriented domains where the cost of missing a valid input outweighs the occasional acceptance of an invalid one. High recall enhances coverage but may compromise reliability.

V. DISCUSSION

Our experiments confirm the effectiveness of our approach. As shown in Table I, GPT-5 achieves high accuracy (95.89%) and F1 score (96.30), maintaining high performance even while generating formal representations that align with the ground truth regarding the plan's validity. In contrast, GPT-4o's performance drops sharply, with accuracy and F1 scores around 52%, mainly due to difficulties in producing correct formal outputs. Both models occasionally fail to generate syntactically perfect NuSMV models, but GPT-5's error rate is much lower (6.87% unknowns in the few-shot setting) than GPT-4o's (34.61%). This demonstrates GPT-5's stronger ability in formal model generation. Our findings highlight the value of formal verification over baseline LLM judgments, which lack formal guarantees. The higher error rate for GPT-4o also suggests that prompt engineering and post-processing are important for improving results.

Initial qualitative analysis shows GPT-5 usually produces syntactically correct NuSMV models and LTL specifications, but further work is needed to ensure semantic accuracy and handle edge cases. In some cases, GPT-5's models passed verification but did not fully reflect the original plan's intent, indicating a need for better translation and counterexample analysis.

Overall, our results demonstrate that integrating LLMs with formal verification unlocks new possibilities for reliable, scalable plan validation, while also revealing open challenges in the generation of perfect formal representations that capture planning nuances.

VI. Limitations, Conclusion, and Future Directions

Our framework improves the reliability and transparency of LLM-generated plans, but several limitations remain. The study reduces PlanBench verification to binary classification and does not yet address the reasons behind plan invalidity or the semantic quality of generated formal models. While we provide empirical and basic qualitative analysis, further work is needed to assess whether LLMs truly capture the intended formal representations. Improving GPT-5's reasoning and achieving near-perfect accuracy remain open challenges, as does developing a taxonomy of common errors. Our focus is on a simplified verification task, leaving broader applications—such as software and circuit verification—unexplored. There are also concerns about potential misuse, costly errors in critical domains, and bias in generated specifications, underscoring the need for strong safeguards as highlighted in recent studies [16], [17], [18].

In conclusion, our experiments demonstrate that GPT-5 can generate near-perfect NuSMV code and come up with a candidate formal model without significant loss in performance. GPT-4o is inferior to GPT-5 in such tasks, and GPT-5's ability to unlock the area of formal model synthesis can increase the impact of formal verification. By leveraging the strengths of language processing, planning and verification, our framework lays the groundwork for developing robust AI systems capable of sophisticated reasoning and dynamic interaction.

Disclaimer

References

[1] L. De Moura and N. Bjørner, "Bugs, moles and skeletons: Symbolic reasoning for software development," in *International Joint Conference on Automated Reasoning*. Springer, 2010, pp. 400–411.

[2] A. Souri, A. M. Rahmani, N. J. Navimipour, and R. Rezaei, "A symbolic model checking approach in formal verification of distributed systems," *Human-centric Computing and Inf. Sciences*, vol. 9, no. 1, p. 4, 2019.

[3] A. Pnueli, "The temporal logic of programs," in *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, 1977, pp. 46–57.

[4] K. Valmeekam, M. Marquez, A. Olmo, S. Sreedharan, and S. Kambhampati, "Planbench: An extensible benchmark for evaluating large language models on planning and reasoning about change," in *Advances in Neural Information Processing Systems*, vol. 36. Curran Associates, Inc., 2023, pp. 38 975–38 987.

[5] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins, "Pddl-the planning domain definition language," 1998.

[6] S. Kripke, "Semantical considerations on modal logic," *Acta Philosophica Fennica*, vol. 16, pp. 83–94, 1963.

[7] V. Tawosi, K. Ramani, S. Alamir, and X. Liu, "ALMAS: an autonomous llm-based multi-agent software engineering framework," in *Proceedings of the 1st International Workshop on Multi-Agent Systems using Generative Artificial INtelligence for Automated Software Engineering (MAS-GAIN)*, 2025.

[8] V. Agarwal, Y. Pei, S. Alamir, and X. Liu, "Codemirage: Hallucinations in code generated by large language models," 2025. [Online]. Available: https://arxiv.org/abs/2408.08333

[9] R. I. T. Jensen, V. Tawosi, and S. Alamir, "Software vulnerability and functionality assessment using llms," in *2024 IEEE/ACM International Workshop on Natural Language-Based Software Engineering (NLBSE)*. IEEE, 2024, pp. 25–28.

[10] M. Sevenhuijsen, K. Etemadi, and M. Nyberg, "Vecogen: Automating generation of formally verified c code with large language models," 2025. [Online]. Available: https://arxiv.org/abs/2411.19275

[11] H. Wu, C. Barrett, and N. Narodytska, "Lemur: Integrating large language models in automated program verification," 2024. [Online]. Available: https://arxiv.org/abs/2310.04870

[12] S. K. Jha, S. Jha, R. Ewetz, and A. Velasquez, "Co-synthesis of code and formal models using large language models and functors," in *MILCOM 2024-2024 IEEE Military Communications Conference (MILCOM)*. IEEE, 2024, pp. 215–220.

[13] M. Hassan, S. Ahmadi-Pour, K. Qayyum, C. K. Jha, and R. Drechsler, "Llm-guided formal verification coupled with mutation testing," in *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2024, pp. 1–2.

[14] X. Lin, Q. Cao, Y. Huang, H. Wang, J. Lu, Z. Liu, L. Song, and X. Liang, "Fvel: Interactive formal verification environment with large language models via theorem proving," 2024. [Online]. Available: https://arxiv.org/abs/2406.14408

[15] C. P. Lee, D. Porfirio, X. J. Wang, K. C. Zhao, and B. Mutlu, "Veriplan: Integrating formal verification and llms into end-user planning," in *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, ser. CHI '25. ACM, Apr. 2025, p. 1–19. [Online]. Available: http://dx.doi.org/10.1145/3706598.3714113

[16] F. Härer, "Specification and evaluation of multi-agent llm systems – prototype and cybersecurity applications," 2025. [Online]. Available: https://arxiv.org/abs/2506.10467

[17] M. Cemri, M. Z. Pan, S. Yang, L. A. Agrawal, B. Chopra, R. Tiwari, K. Keutzer, A. Parameswaran, D. Klein, K. Ramchandran, M. Zaharia, J. E. Gonzalez, and I. Stoica, "Why do multi-agent llm systems fail?" 2025. [Online]. Available: https://arxiv.org/abs/2503.13657

[18] M. Crouse, I. Abdelaziz, R. Astudillo, K. Basu, S. Dan, S. Kumaravel, A. Fokoue, P. Kapanipathi, S. Roukos, and L. Lastras, "Formally specifying the high-level behavior of llm-based agents," 2024. [Online]. Available: https://arxiv.org/abs/2310.08535

[19] N. Tihanyi, T. Bisztray, M. A. Ferrag, B. Cherif, R. A. Dubniczky, R. Jain, and L. C. Cordeiro, "Vulnerability detection: From formal verification to large language models and hybrid approaches: A comprehensive overview," 2025. [Online]. Available: https://arxiv.org/abs/2503.10784

[20] D. Borrajo, G. Canonaco, T. de la Rosa, A. Garrachón, S. Gopalakrishnan, S. Kaur, M. Morales, S. Patra, A. Pozanco, K. Ramani, C. Smiley, P. Totis, and M. Veloso, "Genplanx. generation of plans and execution," 2025. [Online]. Available: https://arxiv.org/abs/2506.10897

[21] S. Kambhampati, K. Valmeekam, L. Guan, M. Verma, K. Stechly, S. Bhambri, L. Saldyt, and A. Murthy, "Llms can't plan, but can help planning in llm-modulo frameworks," 2024. [Online]. Available: https://arxiv.org/abs/2402.01817

[22] F. Giunchiglia and P. Traverso, "Planning as model checking," in *Recent Advances in AI Planning*, S. Biundo and M. Fox, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 1–20.

[23] A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri, "Nusmv: A new symbolic model verifier," in *International conference on computer aided verification*. Springer, 1999, pp. 495–499.

[24] A. Biere, "Bounded model checking," in *Handbook of satisfiability*. IOS press, 2021, pp. 739–764.