Planning in the LLM Era: Building for Reliability and Efficiency

Michael Katz[1], **Harsha Kokel**[2], **Kavitha Srinivas**[1], **Shirin Sohrabi**[1]

[1]IBM T. J. Watson Research Center, Yorktown Heights, NY 10598, USA
[2]IBM San Jose, CA 95141, USA
{michael.katz1,harsha.kokel,kavitha.srinivas}@ibm.com, ssohrab@us.ibm.com

Abstract

Growing attention to intelligent agents has put a spotlight on one of their central capabilities: planning. Early attempts to leverage large language models (LLMs) for planning relied on single-shot plan generation, followed by hybrid approaches that coupled LLMs with limited external search. These methods, unsound and incomplete by their very nature, often require substantial resources without yielding better solutions on unseen problems. As the limitations of LLMs become clearer, recent work has shifted toward using them at *solution construction time* – generating symbolic solvers for a family of problems that can be verified and then used efficiently at inference time. This trend reflects the growing need for agents that are both reliable and resource-efficient. It also offers a path towards generating maintainable planners with minimal dependence on language models at inference time. In this paper, we argue that this shift reflects a broader realignment of the planning field in the LLM era. We examine three major categories of planner-generation methods, discuss their current limitations, and outline research steps towards a more reliable and efficient LLM-based generation of planners.

Introduction

Planning is one of the defining capabilities of intelligent agents, enabling them to reason about actions, anticipate their consequences, and synthesize solutions that achieve complex goals. This recognition has recently brought renewed attention to planning within the broader surge of interest in language-driven agents. Early attempts to use large language models (LLMs) for planning treated each task independently by prompting the model to generate entire plans in a single step (Silver et al. 2022; Valmeekam et al. 2023a,b; Kambhampati et al. 2024). These approaches achieved limited success on simple problems but quickly revealed fundamental limitations: the inability to reconsider earlier decisions, poor long-horizon reasoning, and difficulty generalizing to unseen instances. More recent evaluations of frontier models reflect the rapid and multifaceted nature of progress in LLM-based planning, showing that systems such as GPT-5 can approach the coverage of a strong planner like LAMA on existing domains, while still degrading under obfuscation and requiring vastly greater computational resources (Corrêa, Pereira, and Seipp 2025). Hybrid methods that incorporated restricted forms of backtracking or external search into LLM-based exploration (Yao et al. 2023; Besta et al. 2024; Sel et al. 2024) are unsound and incomplete, and often require dramatically more LLM calls without delivering reliably better performance (Katz et al. 2024). Further, all that hard work is wasted – each problem is solved separately and no computation is reused for the next problem in the same dataset.

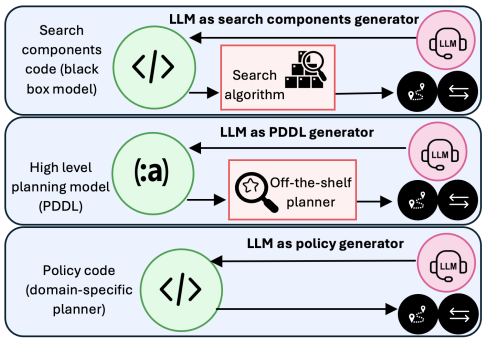

Figure 1: An overview of the planner generation methods

These realizations, coupled with increased awareness of the environmental and monetary costs of LLM inference gave rise to the emergence of a promising paradigm: instead of relying on LLMs at inference time, use them at construction time. In this view, LLMs are not planners themselves but generators of domain-specific planners that can be validated, maintained, and run efficiently without repeated model calls. This shift is visible across multiple lines of work. In this position paper, we discuss three directions outlined in Figure 1. Search-based planner generation (NL2Search) uses LLMs to generate code for search components, i.e., successor generator and goal test (Katz et al. 2024; Cao et al. 2026), as well as heuristic functions (Corrêa, Pereira, and Seipp 2025; Tuisov, Vernik, and Shleyfman 2026). Planning via planning models generation (NL2PDDL) focuses on translating natural language task description into formal planning language that can be consumed by off-the-shelf planners (Guan et al. 2023; Oswald et al. 2024; Gestrin, Kuhlmann, and Seipp 2024; Huang, Lipovetzky, and Cohn 2025; Tantakoun, Muise, and Zhu 2025). Policy code generation (NL2Policy) leverages LLMs to synthesize domain-specific strategies and executable poli-

cies that generalize across problem instances (Silver et al. 2024; Hodel 2024; Stein et al. 2026). We discuss the current state of these directions and the next steps that must be addressed to operationalize these ideas in trustworthy and effective language-driven agents.

Generating Search Based Planners

An emerging trend in solving planning tasks with language models focused on simply asking the model to produce a plan by providing a few examples (Silver et al. 2022). These models were able to find plans only in the simplest cases, when the solutions were very similar in structure to the example plans. Any need to generate something novel or unseen resulted in failure (Valmeekam et al. 2023b). This quickly reinforced a familiar insight from classical planning: generating a plan often requires search. Hence, an alternative line of work shifted toward integrating search, the ability to backtrack and consider many possibilities before finding a valid one. As language models excel at text-based input, the focus was on planning problems expressed in natural language. A variety of approaches were proposed, exploiting search methods for the heavy lifting of exploring the problem's state space, while the language model was used to define the search space, with the required search components successor function $succ$ and goal test is_goal implemented as calls to the model (Hao et al. 2023; Shinn et al. 2023; Yao et al. 2023; Besta et al. 2024; Sel et al. 2024).

All these attempts have the same common pitfall, their search components implementation calls a language model, which takes significant time and is impractical in planning, where problems have combinatorially large state spaces and it is imperative to have an *efficient* implementation of these search components. The aforementioned literature made a different choice, instead heavily restricting the search algorithms, bounding their exploration depth, width, frontier size, etc. Consequently, these search algorithms (e.g., modified BFS) lost their *soundness* and *completeness* properties (Katz et al. 2024), without gaining in precision (Katz, Kokel, and Sreedharan 2025). A more promising alternative has been to use LLMs only at construction time to generate code (e.g., Python code) that implements search components, avoiding LLM calls during search or inference time. Such implementations can then be tested and reflected upon, improving gradually with case-specific feedback, like in the case of (Automated) Thought of Search (ToS/AutoToS) (Katz et al. 2024; Cao et al. 2026).

That initial work on using the language models to obtain code for the $succ$ and is_goal search components has paved the way for the next step, obtaining code for a domain-specific heuristic h (Corrêa, Pereira, and Seipp 2025; Tuisov, Vernik, and Shleyfman 2026). That was a major step, allowing to move away from a blind exploration into the realm of an *informed* search, making the generated solvers substantially more efficient. Both of these investigations focus on deterministic tasks expressed in PDDL, one for purely propositional and the other for numeric planning. In both cases, it should be rather straightforward to move away from PDDL to tasks that are specified purely in natural language.

The Next Steps

While progress on generating search-based planner components is encouraging, these methods are not yet ready for realistic settings. The foundation laid by ToS and AutoToS demonstrates that code for search components can be synthesized and iteratively refined, but several aspects of these approaches remain in their simplest form. One major example is the **state representation**. In ToS, the state representation was hand-crafted and somewhat similar to a multi-valued encoding (Bäckström and Nebel 1995), while in heuristic-generation work it was propositional and derived from a formal task specification. This raises the question of whether certain representations work better for generating search components. More importantly, it raises the question of where state features should come from in the first place. Prior work assumes that state features are known, and relaxing this assumption is a critical step toward adapting to new domains, such as webpage navigation in WebArena (Zhou et al. 2024) and API calling in AppWorld (Trivedi et al. 2024). One might expect that language models could automatically infer such state features, but current models struggle with this task (Vafa et al. 2024). Nonetheless, prior work on learning state representations from images (Asai and Fukunaga 2018), text (Lindsay et al. 2017; Sohrabi et al. 2018), and reinforcement learning (Konidaris, Kaelbling, and Lozano-Perez 2018; Echchahed and Castro 2025) offers potential clues for addressing this challenge. A central question here is how to determine the right **level of abstraction** for both the actions and the state representation.

Other limitations arise when dealing with **partial information**. Generating successors might require interaction with the environment, like in the aforementioned AppWorld. Such cases might be handled either via planning and execution loop or via knowledge compilation (Palacios and Geffner 2009). Regardless, it can be challenging to determine the initial state, goal formulation, or even whether additional information-gathering actions are required.

Another limitation of the implementation strategy used in AutoToS is its **linear** nature, where each single alternative suggested by the language model is accepted without comparison. In more complex domains, generating and evaluating multiple candidate components should be a global search process with the ability to backtrack on earlier decisions.

Planning via PDDL Generation

PDDL (McDermott et al. 1998) has long served as de-facto the standard formalism for expressing planning tasks, providing the input for a wide range of planners. However, representing a planning task in PDDL is one of the few remaining challenges that require manual human labor. Recent research has turned to language models for help with closing this gap by translating natural language task descriptions into formal PDDL specifications. The developed approaches, that we collectively refer to as NL2PDDL, aim to make planning more accessible, while reducing the burden of manual knowledge engineering and enabling the use of existing planners in settings where formal specifications in PDDL were not yet readily available. The plan generation hence can be done with

existing domain-independent planners, exploiting decades of research and engineering efforts in making efficient planners.

Among the first is the work of Guan et al. (2023), which explores using pre-trained LLMs to construct PDDL (domain and problem). The approach takes as input the detailed natural language description of the task, in-context examples, description of the domain, any physical constraints, and a dynamically updated list of predicates that is initially empty. The LLM then incrementally proposes action argument lists, preconditions, and effects, together with any newly introduced predicates and their natural language descriptions, and refines them through validator feedback and human guidance. To reduce the reliance on human experts for the in-the-loop feedback, Huang, Lipovetzky, and Cohn (2025) propose a fully automated method. They would first generate a diverse library of candidate action schemas to capture multiple interpretations of natural language task description, then apply semantic filtering using sentence encoders to automatically validate, filter, and rank the generated action schemas. Their experiments show that this approach can produce sound plans and better accommodate natural language ambiguity without human feedback. These approaches laid the foundation for the technical construction of the feedback flows. The work of Oswald et al. (2024) focused on evaluation of constructed models, going beyond the semantic filtering and addressing syntactic validity, semantic correctness, and usability of the generated results by existing planners. It did so by focusing on the operational semantics instead of action schema similarity, comparing models in terms of similarity of the solution spaces they implicitly encode. Another approach by Gestrin, Kuhlmann, and Seipp (2024) aims for robustness from minimal textual specifications (rather than assuming rich textual specifications). Their framework, NL2Plan, incrementally extracts necessary information to build the PDDL planning task in a particular order, similar to how students are taught to model in PDDL.

To learn more about the plethora of available techniques that use language models to help construct planning models, one can look at the survey by Tantakoun, Muise, and Zhu (2025). They also introduce Language-to-Plan (L2P), an open-source Python library that re-implements several NL2PDDL approaches. L2P is designed to streamline reuse and enable systematic, reproducible experimental results across a variety of NL2PDDL approaches.

The Next Steps

Since this method deals with generating models in a restrictive form, all the shortcomings of the search-based methods described in the previous section persist here as well. Choosing the right level of abstraction for both actions and predicates/numeric fluents as well as how to deal with features outside of the classical fragment (full observability, deterministic action dynamics, and offline access to task structure) are arguably the biggest open questions. The interplay between planning and execution often provides information beyond the common knowledge - strong suit of language models - that might be exploited via means of transformation. To give an example, in AppWorld, determining whether a concrete *Venmo* payment action is applicable may require observing the response of an API call, which cannot be known a priori. Another limitation is that some applications require object creation (Corrêa et al. 2024), conditional plans and loops, some of which might need to be handled outside of the planning problem, for example in AppWorld, sending *Venmo* payments to "all" friends, formulating an appropriate goal state requires quantifying over an unknown number of objects, a well-known limitation of classical PDDL. Furthermore, it is not clear how to integrate the procedural glue code that is required between APIs, or actions in this case, at certain abstraction levels, such as dealing with multi-page API responses, transforming returned data formats, or chaining heterogeneous tool outputs. All of these fall outside the capabilities of current NL2PDDL pipelines.

Beyond addressing these modeling limitations, there is also substantial room for improving the feedback used during PDDL construction. Existing approaches typically explore modifications to PDDL models in a greedy, linear fashion, evaluating the newly generated model on its own, checking only whether it is valid and solvable without comparing it to the previous model it was intended to refine. This prevents systems from determining whether a newly proposed model represents a true improvement; whether the new version actually improves or fixes issues rather than introducing new ones. While some work begins to move beyond single-path refinement (Huang, Lipovetzky, and Cohn 2025), a more systematic model-space search—such as the one explored in (Caglar et al. 2024) is still largely missing.

There is also considerable room to improve how semantic errors are detected and corrected. Recent work on knowledge engineering in the LLM era (Vallati et al. 2025) emphasizes that knowledge engineering is not solely about drafting action schemas but involves iterative refinement, validation, maintenance, and ensuring that models accurately reflect operational semantics. Hybrid workflows, in which LLMs assist human designers and are complemented by principled model-validation and repair techniques, may therefore offer a more robust foundation. Most NL2PDDL systems rely on VAL for syntactic validation, but deeper semantic inconsistencies—incorrect invariants, missing preconditions, unintended side effects—are rarely addressed. Incorporating richer feedback mechanisms, including domain-analysis tools, invariants, or landmarks, could substantially strengthen model quality. Moreover, current pipelines accept a single LLM suggestion at each refinement step; exploring multiple candidate models or applying meta-reasoning over the refinement trajectory could prevent error accumulation and improve convergence.

Planning via Policy Code Generation

Generalized planning deals with finding policies that can efficiently solve the entire family of planning problems (Jiménez, Segovia-Aguas, and Jonsson 2019). The conceptual idea of exploring the use of LLMs for generating a generalized policy in the form of code (Silver et al. 2024) is naturally appealing, as the generated code can be validated, adapted, and when deemed working, deployed. In this original two-phase formulation, a language model is asked to summarize and describe in natural language both the problem and a strategy to solve

the problem. Then, a language model is asked to implement the strategy in code. The generated code can be tested with a held out set of instances to produce a feedback in case of an error. A key limitation of such an approach is the inherent assumption that the produced natural language strategy is correct; indeed, while the method works remarkably well on some domains, there are domains where the produced strategy is incorrect (Hodel 2024). To mitigate this limitation, the follow up work proposed generating the strategy in the form of pseudo-code (Stein et al. 2026). The pseudo-code can then be verified with held out instances, asking the language model to emulate an execution of the pseudo-code on these inputs. In this way, mistakes can be detected and corrected prior to the generation of the generalized plan, substantially improving the reliability across diverse IPC domains.

Approaches that leverage LLMs' programming abilities to generate a policy are also emerging in related areas of task and motion planning (Singh et al. 2023; Sun et al. 2023; Liang et al. 2023) as well as web interaction domains (Song et al. 2025). Code-As-Policies (Liang et al. 2023) define a hierarchical code generation approach with LLMs such that new functions are defined recursively to create complex policies for robot. Policies ProgPrompt (Singh et al. 2023) proposes to prompt the language model to implement a python function as a policy for a new task, given a policy as in-context example. CodeActAgent (Wang et al. 2024) proposes to reduce LLM calls by prompting LLM to provide executable python code that can iterate over multiple actions in a single generation. AdaPlanner (Sun et al. 2023) proposed adaptive closed-loop planning with LLMs through code-prompting for AlfWorld, with a refinement flow that is triggered when the generated program fails. In web-interaction domain CoAct (Song et al. 2025) proposes to query LLM for actions as well as CodeActions, to improve efficiency.

The Next Steps

While the concept of two-phase iterative generation of code policies using language models was established by the original work (Silver et al. 2024), the adaptation of the first phase to take a form that can be evaluated is significant (Stein et al. 2026). Coming up with the strategy and implementing the strategy are the two conceptual steps that need to be performed, but the actual realization can differ. The current work validates the two steps separately, moving to the next once the first one was deemed valid. One can however imagine that during the second step, an evidence is obtained that the strategy/pseudo-code obtained in the first step is incorrect, resulting in revisiting the first step.

Similarly to the case of AutoToS, a limitation of the current approach is its linear nature, where alternatives are weighted against each other and local decisions are made. This local search through the space of alternative changes proposed by the language model is an improvement over the paradigm adopted by AutoToS, but here as well, the better choice would be a global, systematic search procedure. Another limitation is the restriction to PDDL in the input. While extending to natural language specified inputs would bring forward the many of the issues pointed out in the previous sections, it might be unavoidable in real-world applications.

Cross-Category Observations

We have discussed each of the categories in isolation. Here, we propose a cross-category view. Policy-code generation appears to get us closer to realistic agentic environments, with many of the limitations observed in other approaches being easier to tackle within this paradigm. Unlike NL2Search or NL2PDDL, policy-code generation can directly express procedural glue code, interleave sensing with acting (addressing partial observability more naturally), and incorporate reusable components that arise frequently in real applications. Such components natively capture the hierarchical nature of some applications. For example, in AppWorld, one can imagine writing policies that capture sub-tasks, such as sending Venmo payments to friends, co-workers, or roommates. It is also possible to define macro-actions or reusable code fragments that support multiple tasks, including authentication, retrieval, filtering, or error handling. The reusable code can help warm-start the generation of policy, and can especially help in cases when the task description is changed slightly.

It is also worth noting that these methods can be used to improve each other. As a simple example, one can imagine NL2Search first used to obtain the search components, passing these generated components code as an additional input to NL2PDDL. More complex iterations are also possible and sometimes are unavoidable. In domains where search is needed and pure policy-code generation is not possible, a mix of approaches might work best. Higher-level actions/subtasks might be realized via policy-code, while the search-requiring planning can be done via one of the other two methods. This kind of integration of the discussed methods seems to be most promising for handling realistic scenarios.

To sum up, while the progress to date in LLM-driven policy synthesis is promising, opportunities remain for the planning community to shape how planning concepts, policy generation, procedural knowledge, and hierarchical structure, can make meaningful impact in real-world agentic systems.

Conclusions

In this position paper, we examined three major categories of planner-generation methods: search-based planner generation (NL2Search), planning via PDDL or model generation (NL2PDDL), and planning via policy or solver code generation (NL2Policy). The methods focus on using LLMs at construction time, to generate the search components, the planning models, or policy that can be validated and used efficiently at inference time. We examined each direction, highlighting the progress made to date while outlining the challenges, and the future steps needed to make these approaches usable in real-world applications. Although each line of work currently has limitations and/or makes simplifying assumptions, these challenges also reveal opportunities for future research and can help motivate research to address the identified gaps. Ultimately, the opportunity ahead is to both advance planning research along these directions and to have a critical view and/or framework for assessing their impact in the LLM era, ensuring the planning contributions meaningfully influence the efficiency, reliability, and capabilities of agentic systems in the years to come.

References

Asai, M.; and Fukunaga, A. 2018. Classical Planning in Deep Latent Space: Bridging the Subsymbolic-Symbolic Boundary. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI 2018)*, 6094–6101. AAAI Press.

Bäckström, C.; and Nebel, B. 1995. Complexity Results for SAS^+ Planning. *Computational Intelligence*, 11(4): 625–655.

Besta, M.; Blach, N.; Kubicek, A.; Gerstenberger, R.; Podstawski, M.; Gianinazzi, L.; Gajda, J.; Lehmann, T.; Niewiadomski, H.; Nyczyk, P.; and Hoefler, T. 2024. Graph of Thoughts: Solving Elaborate Problems with Large Language Models. In Dy, J.; and Natarajan, S., eds., *Proceedings of the Thirty-Eighth AAAI Conference on Artificial Intelligence (AAAI 2024)*, 17682–17690. AAAI Press.

Caglar, T.; Belhaj, S.; Chakraborti, T.; Katz, M.; and Sreedharan, S. 2024. Can LLMs Fix Issues with Reasoning Models? Towards More Likely Models for AI Planning. In Dy, J.; and Natarajan, S., eds., *Proceedings of the Thirty-Eighth AAAI Conference on Artificial Intelligence (AAAI 2024)*, 20061–20069. AAAI Press.

Cao, D.; Katz, M.; Kokel, H.; Srinivas, K.; and Sohrabi, S. 2026. Automating Thought of Search: A Journey Towards Soundness and Completeness. In *Proceedings of the Thirty-Sixth International Conference on Automated Planning and Scheduling (ICAPS 2026)*. AAAI Press.

Corrêa, A. B.; Giacomo, G. D.; Helmert, M.; and Rubin, S. 2024. Planning with Object Creation. In Bernardini, S.; and Muise, C., eds., *Proceedings of the Thirty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2024)*, 104–113. AAAI Press.

Corrêa, A. B.; Pereira, A. G.; and Seipp, J. 2025. The 2025 Planning Performance of Frontier Large Language Models. arXiv:2511.09378 [cs.AI].

Corrêa, A. B.; Pereira, A. G.; and Seipp, J. 2025. Classical Planning with LLM-Generated Heuristics: Challenging the State of the Art with Python Code. In *Proceedings of the Thirty-Ninth Annual Conference on Neural Information Processing Systems (NeurIPS 2025)*.

Echchahed, A.; and Castro, P. S. 2025. A Survey of State Representation Learning for Deep Reinforcement Learning. *Transactions on Machine Learning Research*, 2025.

Gestrin, E.; Kuhlmann, M.; and Seipp, J. 2024. NL2Plan: Robust LLM-Driven Planning from Minimal Text Descriptions. In *ICAPS 2024 Workshop on Human-Aware and Explainable Planning (HAXP)*.

Guan, L.; Valmeekam, K.; Sreedharan, S.; and Kambhampati, S. 2023. Leveraging Pre-trained Large Language Models to Construct and Utilize World Models for Model-based Task Planning. In *Proceedings of the Thirty-Seventh Annual Conference on Neural Information Processing Systems (NeurIPS 2023)*, 79081–79094.

Hao, S.; Gu, Y.; Ma, H.; Hong, J.; Wang, Z.; Wang, D.; and Hu, Z. 2023. Reasoning with Language Model is Planning with World Model. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP 2023)*, 8154–8173.

Hodel, N. 2024. *Exploring the Use of LLMs in Generalized Planning*. Bachelor's thesis, Saarland University.

Huang, S.; Lipovetzky, N.; and Cohn, T. 2025. Planning in the Dark: LLM-Symbolic Planning Pipeline Without Experts. In Shah, J.; and Kolter, Z., eds., *Proceedings of the Thirty-Ninth AAAI Conference on Artificial Intelligence (AAAI 2025)*, 26542–26550. AAAI Press.

Jiménez, S.; Segovia-Aguas, J.; and Jonsson, A. 2019. A Review of Generalized Planning. *The Knowledge Engineering Review*, 34: e5.

Kambhampati, S.; Valmeekam, K.; Guan, L.; Verma, M.; Stechly, K.; Bhambri, S.; Saldyt, L. P.; and Murthy, A. B. 2024. Position: LLMs Can't Plan, But Can Help Planning in LLM-Modulo Frameworks. In *Proceedings of the 41st International Conference on Machine Learning (ICML 2024)*, 22895–22907. JMLR.org.

Katz, M.; Kokel, H.; and Sreedharan, S. 2025. Seemingly Simple Planning Problems are Computationally Challenging: The Countdown Game. arXiv:2508.02900 [cs.AI].

Katz, M.; Kokel, H.; Srinivas, K.; and Sohrabi, S. 2024. Thought of Search: Planning with Language Models Through The Lens of Efficiency. In *Proceedings of the Thirty-Eighth Annual Conference on Neural Information Processing Systems (NeurIPS 2024)*, 138491–138568.

Konidaris, G.; Kaelbling, L. P.; and Lozano-Perez, T. 2018. From Skills to Symbols: Learning Symbolic Representations for Abstract High-level Planning. *Journal of Artificial Intelligence Research*, 61: 215–289.

Liang, J.; Huang, W.; Xia, F.; Xu, P.; Hausman, K.; Ichter, B.; Florence, P.; and Zeng, A. 2023. Code as Policies: Language Model Programs for Embodied Control. In *Proceedings of the 2023 IEEE International Conference on Robotics and Automation (ICRA 2023)*, 9493–9500. IEEE.

Lindsay, A.; Read, J.; Ferreira, J. F.; Hayton, T.; Porteous, J.; and Gregory, P. 2017. Framer: Planning Models from Natural Language Action Descriptions. In Barbulescu, L.; Frank, J.; Mausam; and Smith, S. F., eds., *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling (ICAPS 2017)*, 434–442. AAAI Press.

McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL – The Planning Domain Definition Language – Version 1.2. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, Yale University.

Oswald, J.; Srinivas, K.; Kokel, H.; Lee, J.; Katz, M.; and Sohrabi, S. 2024. Large Language Models as Planning Domain Generators. In Bernardini, S.; and Muise, C., eds., *Proceedings of the Thirty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2024)*, 423–431. AAAI Press.

Palacios, H.; and Geffner, H. 2009. Compiling Uncertainty Away in Conformant Planning Problems with Bounded Width. *Journal of Artificial Intelligence Research*, 35: 623–675.

Sel, B.; Al-Tawaha, A.; Khattar, V.; Wang, L.; Jia, R.; and Jin, M. 2024. Algorithm of Thoughts: Enhancing Exploration

of Ideas in Large Language Models. In *Proceedings of the 41st International Conference on Machine Learning (ICML 2024)*, 44136–44189. JMLR.org.

Shinn, N.; Cassano, F.; Gopinath, A.; Narasimhan, K.; and Yao, S. 2023. Reflexion: Language Agents with Verbal Reinforcement Learning. In *Proceedings of the Thirty-Seventh Annual Conference on Neural Information Processing Systems (NeurIPS 2023)*, 8634–8652.

Silver, T.; Dan, S.; Srinivas, K.; Tenenbaum, J.; Pack Kaelbling, L.; and Katz, M. 2024. Generalized Planning in PDDL Domains with Pretrained Large Language Models. In Dy, J.; and Natarajan, S., eds., *Proceedings of the Thirty-Eighth AAAI Conference on Artificial Intelligence (AAAI 2024)*, 20256–20264. AAAI Press.

Silver, T.; Hariprasad, V.; Shuttleworth, R. S.; Kumar, N.; Lozano-Pérez, T.; and Kaelbling, L. P. 2022. PDDL Planning with Pretrained Large Language Models. In *NeurIPS 2022 Workshop on Foundation Models for Decision Making*.

Singh, I.; Blukis, V.; Mousavian, A.; Goyal, A.; Xu, D.; Tremblay, J.; Fox, D.; Thomason, J.; and Garg, A. 2023. ProgPrompt: Generating Situated Robot Task Plans using Large Language Models. In *Proceedings of the 2023 IEEE International Conference on Robotics and Automation (ICRA 2023)*, 11523–11530. IEEE.

Sohrabi, S.; Riabov, A. V.; Katz, M.; and Udrea, O. 2018. An AI Planning Solution to Scenario Generation for Enterprise Risk Management. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI 2018)*, 160–167. AAAI Press.

Song, L.; Dai, Y.; Prabhu, V.; Zhang, J.; Shi, T.; Li, L.; Li, J.; Savarese, S.; Chen, Z.; Zhao, J.; et al. 2025. Coact-1: Computer-using Agents with Coding as Actions. arXiv:2508.03923 [cs.CL].

Stein, K.; Hodel, N.; Fišer, D.; Hoffmann, J.; Katz, M.; and Koller, A. 2026. Improved Generalized Planning with LLMs through Strategy Refinement and Reflection. In *Proceedings of the Thirty-Sixth International Conference on Automated Planning and Scheduling (ICAPS 2026)*. AAAI Press.

Sun, H.; Zhuang, Y.; Kong, L.; Dai, B.; and Zhang, C. 2023. AdaPlanner: Adaptive Planning from Feedback with Language Models. In *Proceedings of the Thirty-Seventh Annual Conference on Neural Information Processing Systems (NeurIPS 2023)*, 58202–58245.

Tantakoun, M.; Muise, C.; and Zhu, X. 2025. LLMs as Planning Formalizers: A Survey for Leveraging Large Language Models to Construct Automated Planning Models. In Che, W.; Nabende, J.; Shutova, E.; and Pilehvar, M. T., eds., *Findings of the Association for Computational Linguistics: ACL 2025*. Association for Computational Linguistics.

Trivedi, H.; Khot, T.; Hartmann, M.; Manku, R.; Dong, V.; Li, E.; Gupta, S.; Sabharwal, A.; and Balasubramanian, N. 2024. AppWorld: A Controllable World of Apps and People for Benchmarking Interactive Coding Agents. In Ku, L.; Martins, A.; and Srikumar, V., eds., *Findings of the Association for Computational Linguistics: ACL 2024*, 16022–16076. Association for Computational Linguistics.

Tuisov, A.; Vernik, Y.; and Shleyfman, A. 2026. Successor-Generator Planning with LLM-Generated Heuristics. In *Proceedings of the Thirty-Sixth International Conference on Automated Planning and Scheduling (ICAPS 2026)*. AAAI Press.

Vafa, K.; Chen, J. Y.; Rambachan, A.; Kleinberg, J.; and Mullainathan, S. 2024. Evaluating the World Model Implicit in a Generative Model. arXiv:2406.03689 [cs.CL].

Vallati, M.; Barták, R.; Chrpa, L.; McCluskey, T. L.; and Petrick, R. P. A. 2025. Knowledge Engineering for Planning and Scheduling in the LLM Era. In Lipovetzky, N.; Sardina, S.; Harabor, D.; and Ramirez, M., eds., *Proceedings of the Thirty-Fifth International Conference on Automated Planning and Scheduling (ICAPS 2025)*, 391–395. AAAI Press.

Valmeekam, K.; Marquez, M.; Olmo, A.; Sreedharan, S.; and Kambhampati, S. 2023a. PlanBench: An Extensible Benchmark for Evaluating Large Language Models on Planning and Reasoning about Change. In *Proceedings of the Thirty-Seventh Annual Conference on Neural Information Processing Systems (NeurIPS 2023) Track on Datasets and Benchmarks*, 38975–38987.

Valmeekam, K.; Marquez, M.; Sreedharan, S.; and Kambhampati, S. 2023b. On the Planning Abilities of Large Language Models - A Critical Investigation. In *Proceedings of the Thirty-Seventh Annual Conference on Neural Information Processing Systems (NeurIPS 2023)*, 75993–76005.

Wang, X.; Chen, Y.; Yuan, L.; Zhang, Y.; Li, Y.; Peng, H.; and Ji, H. 2024. Executable Code Actions Elicit Better LLM Agents. In *Proceedings of the 41st International Conference on Machine Learning (ICML 2024)*, 50208–50232. JMLR.org.

Yao, S.; Yu, D.; Zhao, J.; Shafran, I.; Griffiths, T.; Cao, Y.; and Narasimhan, K. 2023. Tree of thoughts: Deliberate Problem Solving with Large Language Models. In *Proceedings of the Thirty-Seventh Annual Conference on Neural Information Processing Systems (NeurIPS 2023)*, 11809–11822.

Zhou, S.; Xu, F. F.; Zhu, H.; Zhou, X.; Lo, R.; Sridhar, A.; Cheng, X.; Ou, T.; Bisk, Y.; Fried, D.; Alon, U.; and Neubig, G. 2024. WebArena: A Realistic Web Environment for Building Autonomous Agents. In *Proceedings of the Twelfth International Conference on Learning Representations (ICLR 2024)*, 15585–15606. OpenReview.net.