Enhancing GPT-based Planning Policies by Model-based Plan Validation

Nicholas Rossetti[§⋆], Massimiliano Tummolo, Alfonso Emilio Gerevini[§], Matteo Olivato, Luca Putelli, and Ivan Serina

Dipartimento di Ingegneria dell'Informazione, Università degli Studi di Brescia, Via Branze 38, Brescia, Italy

Abstract. Despite Large Language Models (LLMs) have revolutionised Natural Language Processing (NLP), their capability of performing logical reasoning and automated planning is still debated. In this context, the state of the art is PlanGPT, a GPT-2 model specifically trained for planning tasks. This recent approach provides GPT-based planning policies with remarkable performance, but it can generate invalid plans containing violated action preconditions or unsatisfied goals. To address this limitation, we propose an extension of PlanGPT that integrates a plan validator into the generation process. The validator is exploited to prune invalid plan prefixes during the GPT token generation, obtaining a more robust and powerful solution to planning via GPT. We empirically evaluate the effectiveness of our approach and demonstrate its potential in various planning domains.

1 Introduction

Attention-based architectures such as Transformer [22], BERT [4], GPT [14] and Large Language Models (LLMs) have dramatically advanced the field of Natural Language Processing (NLP), achieving remarkable results in tasks like machine translation and summarisation. They even show promise in capturing factual knowledge and basic common sense [7, 10, 13, 23]. However, their reasoning capability, particularly for planning tasks, remains doubtful and methods based on prompting or fine-tuning LLMs struggle to generate valid plans for automated planning problems [1, 20, 21]. More promising results were obtained by fine-tuning a pre-trained language model with planning instances [12, 17], and by applying typical NLP techniques in the planning context (e.g., for goal recognition [3]).

Recently we proposed PlanGPT [15], a novel GPT-based model specifically trained to generate plans for classical planning domains. A limitation of this state-of-the-art system is that it cannot self-identify invalid plans (plans with unsatisfied preconditions or unachieved goals) at generation time, possibly returning an incorrect solution for the planning problem.

⋆ N. Rossetti was enrolled in the National Doctorate on AI conducted by Sapienza, University of Rome with the University of Brescia. §Corresponding authors.

In this paper, we introduce and experimentally evaluate a novel technique to address this issue that integrates a plan validator into the generation process of PlanGPT. The validator identifies and prunes invalid plans during the generation, allowing PlanGPT to exploit a symbolic PDDL [11] representation of the domain and plan reasoning for achieving better planning performance.

2 Background: Classical Planning, GPT and PlanGPT

In this section, we give the necessary background on classical/generalised planning, Generative Pretrained Transformers (GPT), and PlanGPT, a recent GPT-2 model trained from scratch to plan in various domains that is investigated in this paper.

2.1 Classical Planning

Following the formalisation given in [15] to represent deterministic, fully observable planning problems, a classical planning problem is a pair $P = (D, I)$ where D is a planning domain and I is a problem instance. More specifically, D contains a set of predicate symbols p and a set of action schemas with preconditions and effects given by atoms $p(x_1, ..., x_k)$, where each x_i is an argument of the schema; I is a tuple $I = (O, Init, Goal)$ where O is a (finite) set of objects names c_i, and $Init$ and $Goal$ are sets of ground atoms $p(c_1, ..., c_k)$ representing the initial state and the goal of the problem. A classical planning problem $P = (D, I)$ encodes a state model $S(P) = (S, s_0, S_G, Act, A, f)$ where each state $s \in S$ is a set of ground atoms from P, s_0 is the set of fluents of the initial state $Init$, S_G is the set of goal states $s \in S$ such that $Goal \subseteq s$, Act is the set of ground actions in P, $A(s)$ is the set of ground actions whose preconditions are true in s, and f is the transition function so that $f(a, s)$ for $a \in A(s)$ represents the state resulting from applying action a to state s. An action sequence $a_0, ..., a_n$ is applicable in P if $a_i \in A(s_i)$ and $s_{i+1} = f(a_i, s_i)$, for $i = 0, ..., n$, and it is a solution plan if $s_{n+1} \in S_G$. The plan cost is assumed to be its length; therefore, a plan is optimal if no shorter plan exists.

The plan validator [8] is a reasoning tool that, given a domain, a problem instance, and a plan, validates whether the plan solves the problem. The validator checks the applicability of each plan action a_i by verifying the truth of its preconditions in the current state s_i ($a_i \in A(s_i)$), and progresses s_i to s_{i+1} ($s_{i+1} = f(a_i, s_i)$). If an action violates the preconditions ($a_i \notin A(s_i)$), then the validator terminates returning invalid plan. Otherwise the validator verifies whether the last state of the plan reaches the problem goal ($s_{i+1} \in S_G$); in that case, it returns the generated plan as a solution plan, or the invalid plan.

Generalised planning studies the representation and computation of general policies to solve multiple problems in the same planning domain [9, 18, 19]. A general policy can be defined as a function $\pi(Init, Goal, (a_0, ..., a_{i-1}))$ that provides the next action in Act to apply to the current state given the

problem initial state $Init$, the problem goal $Goal$, and the sequence of the actions previously obtained auto-regressively by π and applied starting from $Init$. A policy π solves a set of classical planning instances for the same domain D if, for each of these instances $I = (O, Init, Goal)$, the sequence of actions $a_0 = \pi(Init, Goal, ())$, $a_1 = \pi(Init, Goal, (a_0))$, $a_2 = \pi(Init, Goal, (a_0, a_1))$ ..., $a_k = \pi(Init, Goal, (a_0, ..., a_{k-1}))$ obtained by applying auto-regressively π starting from $Init$ solves I (for some $k \geq 0$; with $k = 0$ when $Init \models Goal$).

2.2 Generative Pretrained Transformer

GPT (Generative Pretrained Transformer [14]) is the decoder of the Transformer architecture [22] that originally was designed to produce a completion sentence given an input prompt for NLP tasks, such as the translation of a sentence into another language, or an answer to a question. For example, given the input prompt question *Where is Rome?*, GPT has the objective to generate *Rome is in Italy.*

As part of the preprocessing phase, the WordPiece tokenizer[4] tokenizes the input prompt. This process converts the words in the prompt into individual words or word fragments, known as *tokens*, used as input to GPT. At training time, the tokenizer collects all distinct tokens into a vocabulary of size v. It is important to note that GPT only processes tokens in this set. After tokenization, GPT is trained using the standard teacher-forcing training by minimising the cross-entropy loss between the training data distribution over prompt-response pairs and the generated response.

The main intuition behind the training of the GPT architecture is that the trained model can learn correlations over tokens and create a hidden representation of the overall input sentence using multiple self-attention layers. After training, GPT is used to predict the next token to extend the input sentence using the learned knowledge.

To generate the full response (sequence of tokens), GPT employs a step-by-step process known as **greedy decoding**. In the first step, the model takes the prompt as input and selects from the vocabulary the token with the highest probability (computed by the model). This token is then added to the prompt, and the extended prompt is used as the input for the next step in an autoregressive process. The generation continues until GPT outputs the *<EndOfSequence>* token, or it reaches the maximum context length. In our example, GPT generates the first token *Rome* for prompt *Where is Rome?*; then the new prompt becomes *Where is Rome? Rome*, and the process is repeated generating *is*, and so on, until *<EndOfSequence>* is produced.

This method could ignore a good prompt's response because, at each step, it selects only the token with the highest probability, cutting off all other possible tokens. GPT can overcome this weakness using a different generation method: the **Multi-Beam Search (MB)**. In the first step, GPT tracks the best N successive tokens and their associated probabilities, where N is a hyperparameter. In our example, given prompt *Rome is*, for $N = 2$ GPT expands the prompt with the most probable 2 tokens, suppose *the* (0.5) and *in* (0.3), discarding the

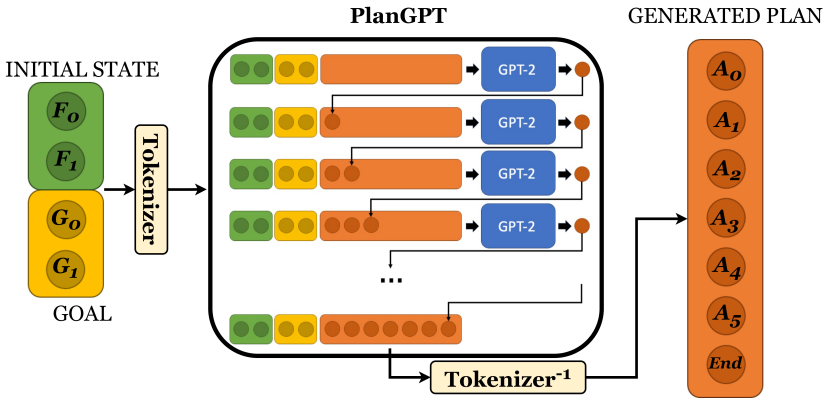

Fig. 1. Architecture of PlanGPT and example of input/output for a planning problem with two fluents in the initial state (F_0 and F_1) and two fluents forming the goal (G_0 and G_1). PlanGPT generates the plan A_0, A_1, ..., *End*.

other tokens. This is repeated creating two sequences of tokens. At each step, for each of these sequences, the model computes the probabilities of all the tokens in the vocabulary and ranks them based on the values of combined probabilities of the tokens in the full sequence.

In our example, the two best-ranked sequences for prompt *Rome is in* are *Rome is in Italy* and *Rome is in Lazio*, with combined probabilities 0.19 and 0.12, respectively; for the other prompt, *Rome is the*, the two best-ranked completion are *Rome is the capital* and *Rome is the best* with probabilities 0.2 and 0.10, respectively. The best two sequences selected by the Beam search are then *Rome is the capital* and *Rome is in Italy*, which are those with the highest probability over the four sequences generated using $N = 2$.The generation in each beam continues until GPT outputs token *<EndOfSequence>* or it reaches the maximum context length. When *<EndOfSequence>* is generated, GPT returns the sequence with the highest combined probability as the completion of the input prompt.

2.3 PlanGPT

PlanGPT is a recent GPT-2 model that we trained from scratch for learning general planning policies achieving SoTA performances on various classical planning domains [15]. Figure 1 shows the architecture of PlanGPT. We selected the smallest GPT-2 version because it is the latest open-source version of GPT, and it requires less training data and fewer computational resources for training than bigger versions. Within our framework, the input of PlanGPT is a set of fluents describing the initial state and the goal of the problem. During training, the completion sentence is the solution plan for the input problem obtained by a planner.

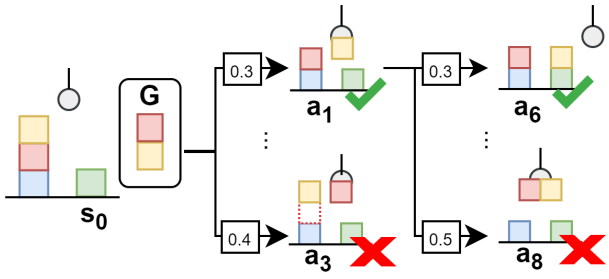

Fig. 2. Example of Validated Multi-Beam Search on a single hypothesis: given the (tokenized) prompt for s_0 and G, PlanGPT generates the following actions with associated probability: $a_1(0.3)$ and $a_3(0.4)$. The validator prunes a_3 and select the second most probable action (a_1) to continue the generation. In the following step the validator prunes a_8 and select a_6.

We build our dataset by generating 70000 planning problems for each domain, following the same setups of the International Planning Competition, but removing some biases in the available generators and benchmarks. For each problem we produced up to 4 plans using the LPG planner[6]. Furthermore, we perform a randomisation on the object names to avoid biases related to their specific names in the training problems and plans.

During the tokenization, we split each fluent (and each action) into its components (our tokens), comprising the fluent (or action) name and its associated objects. For example, for fluent (*At Truck1 Loc1*) we have three tokens: *At*, *Truck1* and *Loc1*; for action (*Drive Truck1 Loc1 Loc3*), four tokens: *Drive*, *Truck1*, *Loc1* and *Loc3*.

Each token generated by GPT is the name of an action or one of its objects. For example, if the output of PlanGPT is the sequence *Drive, Truck1, Loc1, Loc3, ..., <end>*, the first action of the generated plan is *(Drive Truck1 Loc1 Loc3)*. PlanGPT generates one token after the other. E.g., first PlanGPT generates *Drive*, then it adds *Drive* to the input sequence and outputs *Truck1*, and so on.

After the generation of the plan by PlanGPT, the validator unites the tokens into actions and checks the validity of the plan. A limitation of this approach is that, if an action fails due to a violation of a precondition, the plan is considered invalid by the validator, and PlanGPT cannot correct it.

3 Validated Multi-Beam Search

The **Validated Multi-Beam Search (Val-MB)** is a new strategy for generating valid plans for PlanGPT. This generation integrates a plan validator in the generation phase to prune invalid candidate plans due to violated preconditions of an action.

The original version of PlanGPT cannot recognise if it generates an invalid action, and continues to extend the output sequence of (possibly invalid) plan actions until the special token <EndOfSequence> is generated [15]. The validator is called only at the end of the generation to validate the complete plan, and not during the incremental action-by-action plan generation, In our Validated Multi-Beam Search (Val-MB), we integrate the validator in the Multi-Beam (MB) generation. A toy example with $N = 1$ is shown in Figure 2. To visualize this example with a limited number of generations and beams, at each step, PlanGPT generates the complete action (action name and its objects). In the actual generation, different actions are obtained by several beams ($N = 10$) and multiple generation steps to generate the action name and objects. In MB, at each step PlanGPT expands the N hypothesis (candidate partial plans) and selects those with the highest cumulative probability. Starting from the prompt (the problem initial state s_0 and goal G), PlanGPT generates the next actions with their associated probabilities: $a_1(0.3)$ and the invalid $a_3(0.4)$. Note that a_1 is the action picking up the yellow block from the tower (which is valid), while a_3 is the action picking up the red block, which is not admissible given that there is the yellow block on top of it. With MB, PlanGPT selects a_3, deriving an invalid plan, because the probability of a_3 (0.4) is greater than the probability of a_1 (0.3). By introducing the validator, the plan under construction is validated at each generation step, and it is discarded if it contains an action that violates a precondition, such as a_3. This is implemented by revising the probability of the invalid action to 0. In our example, the validator associates probability 0 to a_3, which induces the algorithm to prune this action, and to select action a_1, which is valid, instead of a_3.

The pruning of an action and the selection of an alternative action is possible due to the MB setup in which various hypotheses are developed in parallel during the generation. If we use the validator in the simple greedy generation, when an invalid action is generated, the generation terminates immediately, without finishing the plan. In fact, this approach generates only one candidate plan at a time, and so pruning that candidate means terminating the entire generation procedure. Our Val-MB strategy ensures that all the hypotheses generated up to the current action are valid, since otherwise they would have been previously discarded. Considering a single hypothesis, for each generated token, the algorithm checks if the last group of consecutive tokens forms a grounded action; if this is the case it calls the validator, otherwise it continues with the next token.

It is important to note that in principle Val-MB can fail if all hypotheses have only actions with violated preconditions or if the state is a dead-end, i.e., a state to which no action can be applied. Furthermore, during the hypothesis ranking process, the validator is also used to check if the plan satisfies all the goals, and hence to terminate the generation by returning the solution plan.

domain	Greedy(%)	MB(%)	Val-MB(%)	Δ Val-MB/MB
BLOCKSWORLD	99.5	99.6	**99.9**	20
DEPOTS	79.7	91.0	**97.3**	446
DRIVERLOG	71.3	80.4	**87.8**	568
FLOORTILE	94.4	96.6	**100.0**	213
LOGISTICS	66.6	63.7	**79.5**	1049
SATELLITE	85.1	95.0	**97.0**	128
VISITALL	94.0	97.8	**99.0**	88
ZENOTRAVEL	90.2	98.4	**99.9**	102

Table 1. Coverage of PLANGPT using Greedy, (MB) and Validator Multi-Beam Search (Val-MB) on test set. In the Δ Val-MB/MB column, we report the difference between the number of problems solved by Val-MB and MB.

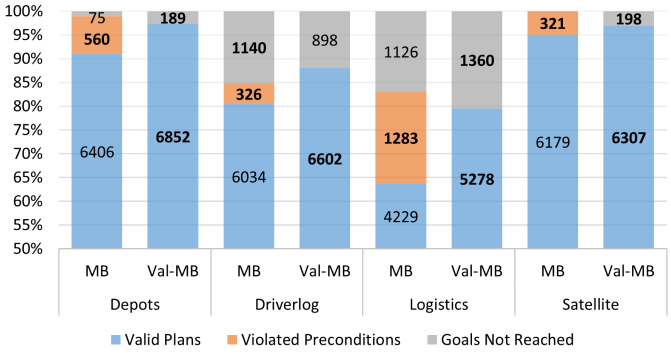

Fig. 3. Bar chart showing the percentage and number of solution plans (blue), invalid plans due to violated preconditions (in orange) and valid plans with goals not reached (grey) for Multi-Beam Search (MB) and Validated Multi-Beam Search (Val-MB).

4 Experimental Results

Starting from the available trained PLANGPT [15], we extended its generation process with the Val-MB strategy considering the same eight benchmark domains (and corresponding PLANGPT models) used in [15]: BLOCKSWORLD, DEPOTS, DRIVERLOG, FLOORTILE, LOGISTICS, SATELLITE, VISITALL and ZENOTRAVEL. The planning problems in each test dataset have difficulty similar to the problems used in the International Planning Competition. They were created via available PDDL generators [16] and solved by the LPG planner [5]. For each domain, we used more than 6000 testing problems. the PLANGPT models were run on a NVIDIA A100 GPU with 40 GB. PLANGPT without the new Val-MB strategy on average generates plans in 10 seconds, with the exception of DRIVERLOG (20 seconds on average) due to its longer plans. With the Val-MB strategy, the generation time increases by 4 to 5 seconds on average.

We tested PlanGPT with Val-MB, using hyperparameter N set to 10, in terms of percentage of generated plans (*coverage*), and compared Val-MB with two standard generation strategies (Greedy and MB with $N = 10$).

Table 1 shows the results of PlanGPT using the Greedy, Multi-Beam (MB) and Validated Multi-Beam (Val-MB) generation strategies. Val-MB outperforms Greedy and performs better than the standard MB for all the considered domains, obtaining a coverage of around 90% or more for all domains except logistics, which obtains coverage 79.5. More in detail, Val-MB allows PlanGPT to obtain the following increments of coverage with respect to MB: 6% in depots (which corresponds to 446 plans), 7% in driverlog (568 plans), 16% in logistics (1049 plans) and 2% in satellite (128 plans).

Figure 3 shows the distribution of the output solutions, valid and invalid plans, for depots, driverlog, logistics and satellite, which are the domains with at least 100 invalid plans in the two compared systems (using Val-MB and MB). The blue bar shows the number of output valid plans, the orange bar the invalid plans that contain at least one precondition violated, and the grey bar the plans with all action preconditions satisfied but that do not reach the goal. We can see that the use of Val-MB increases the number of valid solution plans w.r.t. MB for all domains considered, avoiding generating any plan with violated preconditions (notice the absence of the orange in the Val-MB columns). Therefore, the introduction of a symbolic component such as the validator, which uses domain knowledge representing the planning problem for pruning invalid plans, improves the overall performance, and avoids generating plans that are not executable. However, we observe that Val-MB can lead to generate plans in which the problem goal is not achieved even more often than MB (189 vs 75 in depots and 1360 vs 1126 in logistics). For driverlog we observe that Val-MB reduces both the solution plans with unsatisfied goals (12% vs 15%) and those with violated preconditions (4% in MB).

5 Conclusions

We have investigated a way of integrating symbolic knowledge and reasoning into the generation process of a GPT-based approach to planning (PlanGPT). In particular, we included a PDDL plan validator in the Multi-Beam Search performed by PlanGPT in order to validate plans during their generation, and to prune actions with unsatisfied preconditions. Our experimental analysis on several benchmark domains demonstrates the effectiveness of the extended approach. The enhanced PlanGPT solves the large majority of the test problems with a significant increment w.r.t. the original version.

Current and future work includes integrating of planning knowledge also in the training phase, and overcoming the current limits due to the maximum number of objects in the vocabulary and the length of the context window. Moreover, following [2] we aim to design a system which potentially identifies the problems most solvable by PlanGPT and those which require a symbolic approach.

Acknowledgements

This work was been supported by EU H2020 project AIPlan4EU (GA 101016442), EU ICT-48 2020 project TAILOR (GA 952215), MUR PRIN project RIPER (No. 20203FFYLK), Climate Change AI project (No. IG-2023-174), Regione Lombardia through the initiative "Il Piano Lombardia - Interventi per la ripresa economica", and by SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU.

References

1. Arora, D., Kambhampati, S.: Learning and leveraging verifiers to improve planning capabilities of pre-trained language models. CoRR **abs/2305.17077** (2023)
2. Chiari, M., Gerevini, A.E., Loreggia, A., Putelli, L., Serina, I.: Fast and slow goal recognition. In: Dastani, M., Sichman, J.S., Alechina, N., Dignum, V. (eds.) Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2024, Auckland, New Zealand, May 6-10, 2024. pp. 354–362. ACM (2024)
3. Chiari, M., Gerevini, A.E., Percassi, F., Putelli, L., Serina, I., Olivato, M.: Goal recognition as a deep learning task: The grnet approach. In: Koenig, S., Stern, R., Vallati, M. (eds.) Proceedings of the Thirty-Third International Conference on Automated Planning and Scheduling, Prague, Czech Republic, July 8-13, 2023. pp. 560–568. AAAI Press (2023)
4. Devlin, J., Chang, M., Lee, K., Toutanova, K.: BERT: pre-training of deep bidirectional transformers for language understanding. In: NAACL-HLT (1). pp. 4171–4186. Association for Computational Linguistics (2019)
5. Gerevini, A., Saetti, A., Serina, I.: Planning through stochastic local search and temporal action graphs in lpg. Journal of Artificial Intelligence Research **20**, 239 – 290 (2003). https://doi.org/10.1613/jair.1183
6. Gerevini, A., Serina, I.: LPG: A planner based on local search for planning graphs with action costs. In: AIPS. pp. 13–22. AAAI Press (2002)
7. Geva, M., Khashabi, D., Segal, E., Khot, T., Roth, D., Berant, J.: Did Aristotle use a laptop? A question answering benchmark with implicit reasoning strategies. Trans. Assoc. Comput. Linguistics **9**, 346–361 (2021)
8. Howey, R., Long, D., Fox, M.: VAL: automatic plan validation, continuous effects and mixed initiative planning using PDDL. In: ICTAI. pp. 294–301. IEEE Computer Society (2004)
9. Hu, Y., De Giacomo, G.: Generalized planning: Synthesizing plans that work for multiple environments. In: IJCAI. pp. 918–923. IJCAI Org. (2011)
10. Jiang, Z., Xu, F.F., Araki, J., Neubig, G.: How can we know what language models know. Trans. Assoc. Comput. Linguistics **8**, 423–438 (2020)
11. McDermott, D., Ghallab, M., Howe, A.E., Knoblock, C.A., Ram, A., Veloso, M.M., Weld, D.S., Wilkins, D.E.: Pddl-the planning domain definition language (1998), https://api.semanticscholar.org/CorpusID:59656859
12. Pallagani, V., Muppasani, B., Srivastava, B., Rossi, F., Horesh, L., Murugesan, K., Loreggia, A., Fabiano, F., Joseph, R., Kethepalli, Y.: Plansformer tool: Demonstrating generation of symbolic plans using transformers. In: IJCAI. pp. 7158–7162. IJCAI Org. (2023)

13. Petroni, F., Rocktäschel, T., Riedel, S., Lewis, P.S.H., Bakhtin, A., Wu, Y., Miller, A.H.: Language models as knowledge bases? In: EMNLP/IJCNLP (1). pp. 2463–2473. Association for Computational Linguistics (2019)
14. Radford, A., Narasimhan, K.: Improving language understanding by generative pre-training. In: preprint (2018), api.semanticscholar.org/CorpusID:49313245
15. Rossetti, N., Tummolo, M., Gerevini, A.E., Putelli, L., Serina, I., Chiari, M., Olivato, M.: Learning general policies for planning through gpt models. Proceedings of the International Conference on Automated Planning and Scheduling **34**(1), 500–508 (May 2024)
16. Seipp, J., Torralba, A., Hoffmann, J.: Pddl generators (2022), https://github.com/AI-Planning/pddl-generators
17. Serina, L., Chiari, M., Gerevini, A.E., Putelli, L., Serina, I.: A preliminary study on BERT applied to automated planning. In: IPS/AI*IA. vol. 3345. CEUR-WS.org (2022)
18. Srivastava, S., Immerman, N., Zilberstein, S.: Learning generalized plans using abstract counting. In: AAAI. pp. 991–997. AAAI Press (2008)
19. Srivastava, S., Immerman, N., Zilberstein, S.: A new representation and associated algorithms for generalized planning. Artif. Intell. **175**(2), 615–647 (2011)
20. Valmeekam, K., Hernandez, A.O., Sreedharan, S., Kambhampati, S.: Large language models still can't plan (A benchmark for llms on planning and reasoning about change). CoRR **abs/2206.10498** (2022)
21. Valmeekam, K., Sreedharan, S., Marquez, M., Hernandez, A.O., Kambhampati, S.: On the planning abilities of large language models (A critical investigation with a proposed benchmark). CoRR **abs/2302.06706** (2023)
22. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is all you need. In: NIPS. pp. 5998–6008. Curran Associates Inc. (2017)
23. Wang, Y., Wang, W., Joty, S.R., Hoi, S.C.H.: Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. In: EMNLP (1). pp. 8696–8708. Association for Computational Linguistics (2021)