Integrating Classical Planners with GPT-based Planning Policies

Massimiliano Tummolo[§⋆], Nicholas Rossetti, 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. Recent works on Large Language Models (LLMs) have demonstrated their effectiveness in learning general policies in automated planning. In particular, a system called PlanGPT has achieved impressive performance in terms of coverage in various domains. However, it may produce invalid plans that either satisfy only some goal fluents of the corresponding planning problem or violate the planned actions' preconditions. To overcome this limitation, we propose a novel neuro-symbolic approach that combines PlanGPT with a planner capable of repairing (or completing) the plan generated by PlanGPT, thereby leveraging model-based reasoning. When PlanGPT generates a candidate plan for a specific planning problem, we validate it using a symbolic validator. If the generated plan is invalid, we execute the repair procedure of the planner LPG to obtain a valid solution plan from it. In this paper, we empirically evaluate the effectiveness of our approach and demonstrate its performances across various planning domains. Our results show significant improvements in the performance of both PlanGPT and LPG, highlighting the effectiveness of combining learning methods with traditional planning techniques.

1 Introduction

Attention-based architectures such as Transformers [33], BERT [4], and GPT [17] have driven recent advancements in Natural Language Processing (NLP). These models have achieved impressive results in machine translation and summarisation and have shown promise in understanding factual knowledge and common sense [7, 12, 16, 24]. However, their reasoning abilities, especially for planning tasks, remain limited [1, 30, 32]. Current prompting methods and fine-tuning Large Language Models (LLMs) struggle to generate valid plans for automated planning problems [1, 14, 15, 32].

More promising results were obtained by PlanGPT [20, 21], a novel GPT-based model specifically trained to generate plans for classical planning domains. However, a limitation of this system is that it cannot correct invalid plans at

⋆ M. Tummolo was enrolled in the National Doctorate on AI conducted by Sapienza, University of Rome with the University of Brescia [§]Corresponding authors.

generation time (plans with unsatisfied preconditions or unachieved goals), returning in these cases an incorrect solution for the planning problem.

As other machine learning models [2, 3, 8], PlanGPT is able to learn a general policy which can be used to solve many planning problems. However, they can be also used in combination with a planner: in the work of [29] general policies are exploited as heuristics by planners; in the work in [31] instead a LLM-generated plans are subsequently corrected by a planner. Following these works, in this paper we combine PlanGPT with the LPG planner [6] to address invalid or incomplete plans that may arise during generation. By integrating learning and model-based techniques, our aim is to improve the reliability and accuracy of plan generation by LLMs.

2 Related work

Recent works have studied and analysed the capabilities of LLMs in planning and reasoning in the last few years. In [1, 30], the authors showed how pre-trained GPT models (GPT-3.5 and GPT-4) can be used to generate plans starting from the problem description without a customised training. Unfortunately, their results highlight that pre-trained LLMs often fail to generate correct plans, even with a neural validator that checks the executability of the actions.

In [14, 15], the authors fine-tuned a Code-T5 model [34] with solved problems in several planning domains obtaining a new model, Plansformer. This model showed interesting results, producing more valid plans than the prompting approach in various domains limited in the number of objects. These approaches use an LLM to generate a plan given the initial state, the goal, and the domain description as input, similar to the context of general policies. The main difference from our approach is that we train GPT from scratch with a custom dataset of planning instances.

In another line of work, researchers have exploited different deep-learning architectures to learn general policies. Instead of generating the sequence of actions, these systems compute a heuristic for all the states reachable by applicable actions, starting from the current state and the goal. For instance, in [27, 28], authors proposed a Graph Neural Network (GNN) to address the problems in various benchmark domains. Following this work, the authors tried to integrate the learnt general policies with a greedy first search, obtaining a learned domain-specific planner called Muninn [29]. A similar work was presented in [2, 3, 8] where the authors use GNNs with other graph representations, ranking of the search state and machine learning methods to compute different heuristics.

These approaches produce a numeric heuristic that evaluates the current state and guides the search, and must be continually called upon by the planner. Instead, in our approach, the model directly generates the sequence of actions. For example, in [31], a pre-trained LLM generates a candidate plan that is given as the initial seed to a plan repair system. However, a crucial difference to our work is that the generated plan is provided as input to the planner without any modification.

Instead of using pre-trained LLMs, an initial experiment to train a Language Model from scratch, in the automated planning context, is available in [23]. Moreover, recently we proposed PlanGPT [20, 21], a general policy based on the GPT architecture trained from scratch on a dataset of planning instances. PlanGPT obtains SoTA performances in generating valid plans in various benchmark domains. However, PlanGPT can generate invalid or incomplete plans that violate an action or do not satisfy the goal. In this paper, to address this limitation, we integrate PlanGPT with a plan repair planner, LPG, to correct invalid plans using different candidate plan initialisations.

3 Background: Classical Planning, GPT and PlanGPT

3.1 Classical Planning

Following the formalisation presented in [20] 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.

A plan validator [10] 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 an 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 [11, 25, 26]. 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 given the initial state $Init$, the goal $Goal$ of the problem instance and the list of i actions previously obtained by π auto-regressively. A policy π solves a set of classical planning instances for the same domain D if each of these instances $I = (O, Init, Goal)$ is solved by the sequence of actions $A_\pi = (a_0, ..., a_i)$ obtained by applying auto-regressively π, i.e. $a_0 = (Init, Goal)$, $a_1 = (Init, Goal, a_0)$, $a_2 = (Init, Goal, a_0, a_1)$ ..., $a_i = (Init, Goal, a_{i-1})$.

3.2 Generative Pretrained Transformer (GPT)

In this section, we briefly introduce the architecture of GPT, the neural component of PlanGPT and how it works. A GPT model [17, 18] is the decoder stack of the Transformer architecture [33] developed to generate sequences of elements for NLP tasks. We selected the smallest GPT-2 version because it is the latest open-source version of GPT and, requires less training data and fewer computational resources for training than bigger versions. This model completes the sentence given as input, known as prompt, with the most likely next words. The completion process continues step-by-step, appending the generated word to the input and outputting the most likely next word. In our example, given the input *"Rome is the capital"*, GPT predicts the following words *"of"* and *"Italy"*. The main component of GPT-2 is the attention mechanism. This neural component tries to correlate each couple of tokens in the prompt to extract the various relationships between the sentence, obtaining a meaningful semantic of the prompt to generate the most valuable token. This generation process requires a preliminary tokenization step using a tokenizer, in this case, WordPiece [4], mapping the words in the prompt into individual words or word fragments called *tokens*.

More formally, this strategy is the standard strategy of GPT called **greedy decoding**. At each step, the model takes the prompt as input and outputs the token with the highest probability from the vocabulary (adding it to the prompt). This generation has the issue of producing less varied outputs, as it always selects the most probable token greedily without exploring other tokens. Another method called **top-p sampling** addresses these limitations. In top-p sampling, instead of always selecting the token with the highest probability, the model creates a reduced set of tokens at each step and then randomly samples the next one from the probability distribution over this reduced set. At each step, the strategy cumulatively sums the probabilities of the tokens in descending order until the sum exceeds a threshold p. This token selection ensures that only the most likely tokens, which account for a probability mass of at least p, are considered for sampling.

3.3 PlanGPT

In our previous work, PlanGPT [20], we trained from scratch the architecture of GPT-2 on a customised planning dataset, following the idea that LLMs could understand the grammar and solve a planning problem as shown by [5]. In the context of general policies, after training, PlanGPT can solve complex planning problems similar to those from the International Planning Competition (IPC), achieving state-of-the-art performance across various classical planning domains using the top-p sampling generation.

PlanGPT receives in input a set of fluents representing the initial state and the goal of a planning and has to compute a plan to solve a problem. In order to do that, it has to predict sequentially all the actions of a plan as if they were words. Mapping fluents and action into words is done through the tokenization. During tokenization, we split each fluent (and each action) into its components, 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*. After training, given as input the fluents of the initial state and the goal of a problem, PlanGPT generates the solution plan by generating the next token (the action name or the object name) in an auto-regressive mode with different generation strategies.

At the end of the generation, the validator unites the tokens into actions. Finally, the validator checks the validity of the plan. However, a current limitation of our approach is that if an action fails due to a violation of a precondition or the plan does not satisfy the goal, the plan is considered invalid by the validator, and PlanGPT cannot correct it.

3.4 Local search for Planning Graphs

The standard method for solving classical planning problems is the use of a planner. A planner is a system that addresses planning problems without leveraging particular domain properties or biases, but using only the domain definition and search methods. Given as input the domain description D and the problem I, a planner will produce a plan to solve the problem I. In literature, numerous planners such as LAMA [19], Fast-Downward [9] and LPG [6] exploit different kinds of heuristics and search to solve automated planning problems. In this work, we consider using the Local search for Planning Graphs (LPG) [6] as it offers the possibility of plan repair and completion.

The basic search scheme of LPG was inspired by Walksat [13], an efficient procedure to solve SAT problems. The search space of LPG consists of *action graphs*, particular sub-graphs of the planning graph representing partial plans. The search steps are graph modifications, transforming an action graph into another one. LPG exploits a compact representation of the planning graph to define the search neighbourhood and to evaluate its elements building relaxed planning graphs. This is achieved by an anytime process that produces a sequence of plans, each of which improves the quality of the previous ones.

In addition to generating a plan from scratch, LPG offers a plan repair solution: starting from an invalid or incomplete plan LPG solves the problem by leveraging the knowledge in that plan and correcting it.

4 PlanGPT seed as input of LPG

The main contribution of this work is integrating the LPG planner in the generation phase of PlanGPT. Given that PlanGPT has the potential to generate invalid plans, we use the LPG repair plan option to correct these invalid plans.

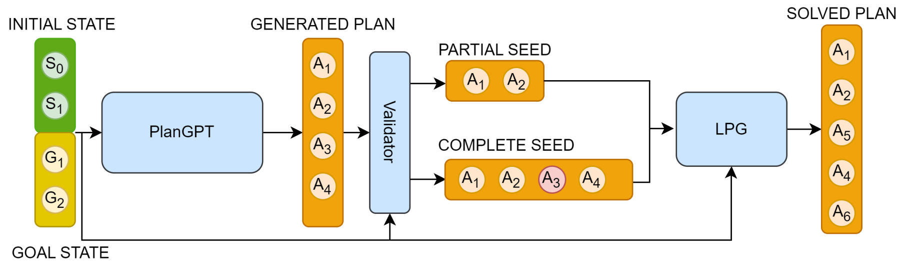

Fig. 1. Example of input/output for a planning problem with two fluents in the initial state (S_0 and S_1) and two fluents (G_0 and G_1) forming the goal. PlanGPT generates the plan A_1, ..., A_4. The validator verifies the correctness of the candidate plan and, if it finds it invalid, generates the partial and complete seeds. Then LPG generates a valid plan using the seed with the repair option.

Starting from an already generated plan, instead of an empty seed as usual, LPG can compute an action graph from the current invalid plan that corrects inconsistencies and improves the quality of the final solution, reducing the search time required and exploiting the knowledge in the generated plan.

The proposed architecture includes a neural system, PlanGPT, which produces a plan (valid or not), a symbolic validator that can check its validity, and LPG that can repair it. We refer to this combined system as R-PlanGPT.

Following Figure 1, the generation process of the system starts from the component PlanGPT, which takes the initial state and goals as input and generates a candidate plan. In this example, given the fluents S_0 and S_1 and the goal fluents G_1 and G_2, PlanGPT generates a plan composed of 4 actions: A_1, A_2, A_3, A_4. Then, the symbolic validator checks the actions' validity and the satisfaction of the goal. In the example, the validator analyses the actions A_1, A_2, A_3, A_4, discovering that the action A_3 is invalid.

We implement two strategies to create the LPG candidate seed: the **partial seed**, where the validator discards some actions from the original generated plan, or the **complete seed**, where LPG takes as input the whole generated plan.

In the partial seed, if the plan has an action that violates a precondition, the validator prunes all the actions from the violation, obtaining the candidate plan. Instead, suppose the plan has no preconditions' violations. In that case, the validator produces the candidate plan by selecting all the actions up to the last one that satisfies the problem's last solved goal. Furthermore, the validator checks the presence of loops in the candidate plan computed, i.e. sequences of actions that repetitively reach the same state without progressing towards the goals, and removes them. The idea is to provide LPG with a correct initial candidate plan to complete that satisfies various goal fluents and does not provide any violated preconditions or loops. In Figure 1, for the partial seed, since action A_3 is invalid, the validator prunes actions A_3, A_4 and produces the candidate plan A_1, A_2.

In the complete seed, instead, the candidate plan for LPG consists of the plan from PlanGPT without any modifications. This approach enabled LPG to take advantage of the knowledge of the plan generated after a precondition violation and, therefore, to provide details on the subsequent phases of the plan. By correcting the violation of the preconditions, if any, it is possible that the remaining plan is valid and helps LPG in its search. In Figure 1, for the complete seed, the candidate plan is A_1, A_2, A_3, A_4.

Finally, LPG searches for a solution starting from the input candidate plan and gives the final plan solution as output. In Figure 1, LPG takes as input the complete seed and produces the new plan A_1, A_2, A_5, A_4, A_6 substituting action A_3 with action A_5 and adding action A_6 to obtain a valid solution to the problem. In our experimental evaluation, we also considered an empty candidate plan as input to LPG.

5 Experimental Results

Starting from the available trained PlanGPT [20], we extended its generation process with the integration of a validator and the LPG planner in the eight benchmark domains used in [20]: blocksworld, depots, driverlog, floortile, logistics, satellite, visitall and zenotravel.

For each domain, we used more than 6000 testing problems (`Tset`). The planning problems in `Tset` are similar to those used in the International Planning Competition (IPC) and are created using the available PDDL generators [22]. The choice to use this dataset is to compare a larger number of problems since the test set of IPC contains only between 30 and 100 problems for each domain. The PlanGPT models are run on an NVIDIA A100 GPU with 40 GB. Regarding the planners, for each problem, we generate solution plans on an Intel (R) Xeon (R) Gold 6140M CPU @ 2.30GHz with a standard time limit of $300s$.

To evaluate our experiments, we use classical planning metrics such as the coverage, the plan length and the *IPCScore-Quality* (IPCQ) and *IPCScore-Agile* (IPCA) as defined in the last International Planning Competition (IPC 2023):

- **Coverage**: the percentage of valid plans over the total number of generated plans.
- **Plan Length**: the score of a problem is the number of actions of a solution plan. The score of a model is the mean of its score for the problems solved by all the models.
- **IPCScore-Quality**: The score of a problem is the ratio C^*/C where C is the cost of the plan discovered by the model and C^* is the cost of a reference plan (the cheapest plan obtained by all models for that problem). The score of an unsolved problem is 0. The score of a model is the sum of its scores for all problems.
- **IPCScore-Agile**: The score of a problem on a solved task is 1 if the task was solved within 1 second and 0 if not solved within the resource limits. If the problem is solved in T seconds ($1 \leq T \leq 300$) then its score is $1 - log(T)/log(300)$. The score of a model is the sum of its scores for all problems.

In the following, we evaluate PlanGPT and LPG in terms of coverage and plan length compared on `Tset` considering different execution time thresholds. Next, we evaluate R-PlanGPT, combining the repair process of LPG using partial, and complete seed on the invalid plans generated by PlanGPT in terms of coverage, plan length, and execution time to find the best solution compared to LPG with an empty seed. Finally, we compare the performances of R-PlanGPT with LAMA, FD, PlanGPT and LPG in the IPC benchmark problems for our domains in terms of coverage, IPCQ and IPCA.

5.1 PlanGPT vs LPG

This section compares PlanGPT (using top p sampling) and LPG (initialised with an empty seed) over the `Tset` on coverage and plan length, evaluating different execution times. PlanGPT generates a single solution, while LPG produces multiple solutions in an interval of 5 minutes of incremental quality. Considering that PlanGPT was trained on suboptimal planning instances solved by LPG, this comparison demonstrates the model's capabilities relative to its teacher, highlighting instances where it can surpass the performances of LPG. Table 1 shows the coverage of PlanGPT and LPG considering different execution time thresholds: 10 seconds, 20 seconds, 1 minute and 5 minutes. Both systems obtain high coverage in under 10 seconds of execution with almost perfect scores on blocksworld, satellite, visitall and zenotravel. Instead, depots and driverlog obtain lower levels of coverage due to the high number of states to evaluate in the search, while floortile's performances are affected by the numerous dead-ends of the domain. After 20 seconds, we can observe that even depots and driverlog obtain high coverages, while floortile obtains high coverages only after 1 minute for both systems. Finally, considering the time threshold of 5 minutes, LPG obtains higher coverages than PlanGPT having a difference up to 5% (0% in blocksworld, 5% in depots, 2% in driverlog and satellite, and 0.1% in visitall and zenotravel). We can observe a notable difference in the logistics domain where LPG obtains a higher coverage than PlanGPT (98% vs 77%). This domain is very complex for PlanGPT since it belongs to the C3-fragment of the first-order logic. Lastly, we observe that PlanGPT obtains higher coverage than LPG (99% vs 93%) in floortile, because the LPG search is affected by the numerous dead-ends present in this domain.

Table 2 shows the results of the plan length of PlanGPT and LPG of the problems solved by both systems considering the time ranges of 10 seconds, 20 seconds, 1 minute and 5 minutes. In general, we observe that, in less than 10 seconds, PlanGPT obtains plans of shorter length; after 1 minute, the two systems are comparable in different domains. Given more time, LPG obtains a better quality in all domains, except for floortile, where PlanGPT obtains the best plan length. Given that PlanGPT generates a single plan and stops the generation, we expect that LPG obtains shorter plans in the long run because it continues to produce solutions of incremental quality. In blocksworld and depots, within 1 minute, PlanGPT obtains shorter plans of an action.

	$<$ 10s		$<$ 20s		$<$ 1m		$<$ 5m	
domain	LPG	PlanGPT	LPG	PlanGPT	LPG	PlanGPT	LPG	PlanGPT
blocksworld	**99.3**	**99.3**	99.9	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
depots	**77.6**	76.0	**99.4**	94.5	**99.8**	94.8	**99.9**	94.9
driverlog	**61.1**	61.0	**98.1**	96.3	**99.8**	97.8	**99.9**	97.9
floortile	40.8	**41.2**	58.0	**62.6**	93.2	**99.3**	93.5	**99.6**
logistics	**98.0**	77.2	**99.9**	77.3	**99.9**	77.3	**99.9**	77.3
satellite	**100.0**	98.1	**100.0**	98.1	**100.0**	98.1	**100.0**	98.1
visitall	**99.8**	99.8	**100.0**	99.9	**100.0**	99.9	**100.0**	99.9
zenotravel	**100.0**	99.9	**100.0**	99.9	**100.0**	99.9	**100.0**	99.9

Table 1. Percentage coverage of PlanGPT and of LPG over `Tset`. For each time frame we highlight in bold the best results.

	$<$ 10s		$<$ 20s		$<$ 1m		$<$ 5m	
domain	LPG	PlanGPT	LPG	PlanGPT	LPG	PlanGPT	LPG	PlanGPT
blocksworld	40.4	**38.0**	39.4	**38.2**	38.3	**38.2**	**37.5**	38.2
depots	35.0	**33.9**	38.8	**37.6**	37.9	**37.7**	**36.8**	37.7
driverlog	69.5	**67.5**	85.9	**84.7**	**80.2**	85.6	**73.4**	85.7
floortile	51.5	**43.8**	76.5	**55.8**	70.4	**58.8**	61.9	**59.0**
logistics	**19.3**	21.7	**19.1**	21.7	**18.8**	21.7	**18.5**	21.7
satellite	**29.8**	29.9	**29.8**	29.9	**29.7**	29.9	**29.7**	29.9
visitall	**41.7**	44.8	**40.9**	44.9	**39.9**	44.9	**38.9**	44.9
zenotravel	41.3	**40.3**	**39.8**	40.3	**38.3**	40.3	**36.3**	40.3

Table 2. Plan length of PlanGPT and of LPG over `Tset` for the problem solved by both systems. For each time frame we highlight in bold the best results.

After 1 minute, LPG produces plans of similar length. In driverlog, we can observe that LPG obtains better plans after 1 minute with a more significant number of actions (5 after 1 minute and 12 after 5 minutes). Overall, in the domain of logistics, there is always a difference of 3 actions between the systems. In visitall, this difference is around 3 to 5 actions in favour of LPG. In zenotravel, first PlanGPT has a higher quality, but then LPG obtains lesser and lesser actions up to a difference of 4 actions. In satellite, the two systems obtain the same plan length. Finally, we can observe that in floortile, PlanGPT generates plans having better quality for each timeframe: a difference of 8 actions before 10 seconds, 11 before 1 minutes, since PlanGPT has already generated all the plans, and 3 after 2 minutes when LPG completes its search. This surprising result in the floortile domain implies that learning a policy from valid plans can be helpful when the search uses a negative-effect relaxation in the heuristics leading to explore numerous undetected dead-ends.

5.2 PlanGPT as initial seed of LPG

In Table 3, we show the integration of PlanGPT and LPG, R-PlanGPT, using different initialisation strategies (partial and complete plan seed) given to the planner LPG to solve the unsolved problem of PlanGPT and comparing against LPG with an empty seed. We show the coverage, plan length, and the average execution time used by LPG to obtain the best solution for each seed. We analyse depots (356), driverlog (158), logistics (1504) and satellite (123)

	LPG			R-PlanGPT					
	Empty			Partial			Complete		
domain	cov	length	time	cov	length	time	cov	length	time
depots	**100.0**	55.2	128.9	**100.0**	56.4	**84.9**	97.8	**54.8**	105.9
driverlog	99.9	**127.6**	169.6	**100.0**	128.8	**101.1**	87.4	128.2	126.6
logistics	99.5	52.6	104.5	**99.8**	**51.8**	**86.9**	97.3	51.9	89.9
satellite	**100.0**	**40.5**	**14.8**	**100.0**	40.6	18.7	**100.0**	40.8	18.2

Table 3. Percentage coverage (cov), plan length (length), CPU time of problem unsolved by PlanGPT and corrected by the complete system R-PlanGPT, using the partial and complete seed strategies. In the first columns we show also the results obtained by LPG without seed on the same set of problems.

since these domains have at least 100 invalid plans generated by PlanGPT. The execution time of the generation of PlanGPT and the validation (in the partial and complete seed) is added to the LPG computation time to show a fair comparison to the empty seed, which does not require PlanGPT. On the other side, the LPG search time limit is 5 minutes.

From Table 3, concerning coverage, the quality of the empty and partial seeds is comparable, solving most of the problems unsolved by PlanGPT. However, we have a drop in performances with the complete seed. More specifically, we observe that using the partial seed in driverlog and logistics slightly enhances the coverage of LPG. On the contrary, in satellite, using a seed is not helpful to LPG. However, using the complete seed, we observe a drop of the coverage in satellite (−2.2%), depots (−12.5%) and logistics (−2.2%). LPG struggles to correct very long plans (also with cycles) or plans with multiple violations and, therefore, the useful actions that can be extracted from the plan are insufficient to compensate for this problems. Although individually LPG and PlanGPT cannot compute a valid solution for 20 problems in floortile, when combined using the partial seed, R-PlanGPT solves these problems, obtaining valid plans exploiting the knowledge of PlanGPT and refining the plan with the LPG search.

Concerning the plan length, the quality of the plans for the 3 seeds is comparable overall, with a mean difference of a single action. For depots and satellite, the best seed is the complete seed. In the case of driverlog, LPG without PlanGPT obtains the best length. While for logistics, the best seed is the partial seed.

Regarding the time to generate a solution, using the partial or complete seeds allows a notable reduction in the heuristic search of LPG to obtain the best solution. We observe that the time between the plan generated by LPG from scratch compared to the candidate plans of partial seed decreases with a delta from $44s$ in depots, $68s$ in driverlog, $18s$ in logistics. Also in the complete seed, we note a decrease in the execution time, but it is less remarkable than the partial seed, because LPG must correct and remove loops in the candidate plan: $23s$ in depots, $43s$ in driverlog and $15s$ in logistics. The only exception is in SATELLITE, where having a seed for LPG does not lower the time for the heuristic search, but neither increases it. This result shows how introducing a plan seed (which contains helpful information on the plan) helps the planner to carry out a more targeted heuristic search, reducing its execution time.

Therefore, from our results, the partial seed is a better strategy for the combined system R-PLANGPT because the cutting strategy can obtain a helpful prefix seed, which the planner can use to start from. The integration of PLANGPT and LPG produces plans of comparable plan length with a significant decrease in the generation of the solution.

5.3 Comparison with SoTA Planners on IPC

In Table 4, we compare the different state-of-the-art classical planners with LPG, PLANGPT, and the combined system R-PLANGPT with the partial seed strategy using a time limit of 5 minutes. It is important to note that the system R-PLANGPT uses different configurations for each domain, so it is a domain dependent approach, and the classical planners considered are domain independent. We used the IPCQuality and IPCAgile metrics. We evaluated the system using the IPC competition problems. Each domain contains 20 planning instances except for BLOCKSWORLD with 35 instances, DEPOTS, 22 and LOGISTICS, 30.

Regarding coverage, FD and LAMA solve all the problems except for DEPOTS (77% and 95%) and FLOORTILE (10%). The last instances of DEPOTS are complex because they have many objects. Although FLOORTILE is complex because it contains multiple dead-ends that are not detected by delete-relaxation heuristics. PLANGPT managed to solve with high coverage in almost all the domains (90% in DEPOTS and 95% in DRIVERLOG and VISITALL) with the exceptions of LOGISTICS (53%) and SATELLITE (70%), providing various invalid plans. In SATELLITE, PLANGPT struggles when a precondition is bound to an object selected by a previous action which is far away from the current one. For instance, in LOGISTICS, PLANGPT struggles to correlate the positions of the truck and the object simultaneously. In contrast, in SATELLITE, PLANGPT struggles to select an instrument with the correct mode to take an image. The best-performing systems proved to be LPG and R-PLANGPT, which solved all the provided planning problems.

Regarding the IPCQ metric, we observe that R-PLANGPT reaches the highest score in DEPOTS (21.40), FLOORTILE (19.64), and LOGISTICS (27.12). In the case of FLOORTILE, the results are provided by the solution plans generated by PLANGPT. In the case of DEPOTS and LOGISTICS, instead, the integration with the plan-repair procedure of LPG produces the shortest plans. LAMA reaches the best quality in DRIVERLOG (19.73), VISITALL (19.60), and ZENOTRAVEL (19.82), while LPG obtains the shortest plan in BLOCKSWORLD (35.00) and SATELLITE (19.74). Finally, PLANGPT obtains the best quality in FLOORTILE and comparable performances in the other domains.

Regarding the IPCA metric, we observe that all the planners are faster than PLANGPT and, therefore, than R-PLANGPT. PLANGPT is generally slower

domain	**FD**		**LAMA**		**LPG**		**PlanGPT**		**R-PlanGPT**	
	IPCQ	IPCA	IPCQ	IPCA	IPCQ	IPCA	IPCQ	IPCA	IPCQ	IPCA
blocksworld	27.61	**35.00**	33.93	34.88	**35.00**	**35.00**	34.73	34.20	34.73	34.20
depots	15.15	14.65	19.45	20.22	18.59	**21.85**	17.42	14.27	**21.40**	17.17
driverlog	18.79	**19.06**	**19.73**	18.96	18.59	18.79	17.08	15.95	17.46	16.32
floortile	2.00	1.46	2.00	1.44	17.36	**15.99**	**19.64**	10.39	**19.64**	10.39
logistics	26.41	**30.00**	26.55	**30.00**	24.99	29.90	14.02	13.05	**27.12**	22.53
satellite	18.51	**20.00**	18.64	**20.00**	**19.74**	**20.00**	11.94	11.61	19.68	17.12
visitall	19.07	15.46	**19.60**	12.38	19.46	**19.86**	16.53	16.41	17.46	16.48
zenotravel	18.85	**20.00**	**19.82**	**20.00**	19.22	19.38	15.71	16.01	17.51	17.50
total	146.39	155.63	159.72	157.88	172.95	**180.77**	147.07	131.89	**175.00**	151.62

Table 4. Comparison of IPCQ and IPCA metrics between classical planners (FD, LAMA and LPG), PlanGPT and R-PlanGPT with the partial seed using the IPC test set.

to compute a solution because it generates only a single solution with higher quality. In satellite, all the planners compute the plans in less than 1 second. In blocksworld FD and LPG take less time to compute one solution compared to the other systems. The same behaviour happens in logistics and zenotravel with FD and LAMA. FD is the faster planner in driverlog (19.06), while LPG obtains the best results in depots (21.85), floortile (15.99) and visitall (19.86).

In conclusion, considering the IPC test set, we evaluated the ability of learned general policies derived from LLM, combined with a plan-repair planner, to provide enhanced results in various domains and show comparable performances with SoTA planners. More in-depth, the floortile domain is particularly appealing because planners based on delete-relaxation heuristics struggle to acquire a solution, while R-PlanGPT obtains perfect coverage.

6 Conclusions

In this work, we investigated the possibility of integrating a neural solution based on the GPT architecture, PlanGPT, with a symbolic one, LPG. Firstly, we analysed the comparison between LPG and PlanGPT, where PlanGPT showed comparable performance to LPG in some domains in a limited time window and better coverage and quality in the case of floortile. Then, we proposed our combined system, R-PlanGPT. We showed that PlanGPT can give information, as a seed, for LPG and guide the heuristic search to a solution in less time than LPG alone. In fact, R-PlanGPT corrects the majority of the test problems using the partial or complete seed obtained from the invalid plans generated from PlanGPT. Finally, we evaluated R-PlanGPT in the context of the IPC competition against state-of-the-art classical planners with comparable performances. Current and future work includes training PlanGPT on other challenging domains for planners, integrating the domain knowledge 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.

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 initiatives "Il Piano Lombardia - Interventi per la ripresa economica" and "Programma degli interventi per la ripresa economica: sviluppo di nuovi accordi di collaborazione con le università per la ricerca, l'innovazione e il trasferimento tecnologico" - DGR n. XI/4445/2021 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. Chen, D., Thiébaux, S., Trevizan, F.: Learning Domain-Independent Heuristics for Grounded and Lifted Planning. In: Proc. of 38th AAAI Conference on Artificial Intelligence (2024), http://felipe.trevizan.org/papers/chen24:goose.pdf
3. Chen, D., Trevizan, F., Thiébaux, S.: Return to Tradition: Learning Reliable Heuristics with Classical Machine Learning. In: Proc. of 34th Int. Conf. on Automated Planning and Scheduling (ICAPS) (2024), http://felipe.trevizan.org/papers/chen24:wlkernel.pdf
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. Geib, C.W., Steedman, M.: On natural language processing and plan recognition. In: IJCAI. pp. 1612–1617 (2007)
6. 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
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. Hao, M., Trevizan, F., Thiébaux, S., Ferber, P., Hoffmann, J.: Guiding GBFS through Learned Pairwise Rankings. In: Proc. of 33rd Int. Joint Conf. on AI (IJCAI) (2024), http://felipe.trevizan.org/papers/hao24:ranking.pdf
9. Helmert, M.: The fast downward planning system. J. Artif. Intell. Res. **26**, 191–246 (2006)
10. 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)
11. Hu, Y., De Giacomo, G.: Generalized planning: Synthesizing plans that work for multiple environments. In: IJCAI. pp. 918–923. IJCAI Org. (2011)
12. 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)
13. Kautz, H., Selman, B.: Pushing the envelope: Planning, propositional logic, and stochastic search. Proceedings of the National Conference on Artificial Intelligence **2** (02 1999)

14. Pallagani, V., Muppasani, B., Murugesan, K., Rossi, F., Horesh, L., Srivastava, B., Fabiano, F., Loreggia, A.: Plansformer: Generating symbolic plans using transformers. CoRR **abs/2212.08681** (2022)
15. 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)
16. 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)
17. Radford, A., Narasimhan, K.: Improving language understanding by generative pre-training. In: preprint (2018), api.semanticscholar.org/CorpusID:49313245
18. Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I.: Language models are unsupervised multitask learners (2019), https://api.semanticscholar.org/CorpusID:160025533
19. Richter, S., Westphal, M.: The LAMA planner: Guiding cost-based anytime planning with landmarks. J. Artif. Intell. Res. **39**, 127–177 (2010)
20. Rossetti, N., Tummolo, M., Gerevini, A., Putelli, L., Serina, I., Chiari, M., Olivato, M.: Learning general policies for planning through GPT models. In: 34th International Conference on Automated Planning and Scheduling (2024)
21. Rossetti, N., Tummolo, M., Gerevini, A.E., Putelli, L., Serina, I., Olivato, M.: Enhancing gpt-based planning policies by model-based plan validation. Proceedings of the 18th International Conference on Neural-Symbolic Learning and Reasoning (2024)
22. Seipp, J., Torralba, A., Hoffmann, J.: Pddl generators (2022), https://github.com/AI-Planning/pddl-generators
23. Serina, L., Chiari, M., Gerevini, A.E., Putelli, L., Serina, I.: A preliminary study on BERT applied to automated planning. In: Proceedings of the 10th Italian workshop on Planning and Scheduling (IPS 2022), RCRA Incontri E Confronti (RiCeRcA 2022), and the workshop on Strategies, Prediction, Interaction, and Reasoning in Italy (SPIRIT 2022) co-located with 21st International Conference of the Italian Association for Artificial Intelligence (AIxIA 2022), November 28 - December 2, 2022, University of Udine, Udine, Italy. CEUR Workshop Proceedings, vol. 3345. CEUR-WS.org (2022)
24. Serina, L., Putelli, L., Gerevini, A.E., Serina, I.: Synonyms, antonyms and factual knowledge in BERT heads. Future Internet **15**(7), 230 (2023). https://doi.org/10.3390/FI15070230, https://doi.org/10.3390/fi15070230
25. Srivastava, S., Immerman, N., Zilberstein, S.: Learning generalized plans using abstract counting. In: AAAI. pp. 991–997. AAAI Press (2008)
26. Srivastava, S., Immerman, N., Zilberstein, S.: A new representation and associated algorithms for generalized planning. Artif. Intell. **175**(2), 615–647 (2011)
27. Ståhlberg, S., Bonet, B., Geffner, H.: Learning general optimal policies with graph neural networks: Expressive power, transparency, and limits. In: ICAPS. pp. 629–637. AAAI Press (2022)
28. Ståhlberg, S., Bonet, B., Geffner, H.: Learning generalized policies without supervision using gnns. In: KR. pp. 474–483. IJCAI Org. (2022)
29. Stahlberg, S., Bonet, B., Geffner, H.: Muninn (2023), https://ipc2023-learning.github.io/abstracts/muninn.pdf
30. 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)

31. Valmeekam, K., Marquez, M., Sreedharan, S., Kambhampati, S.: On the planning abilities of large language models - A critical investigation. In: NeurIPS (2023)
32. 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)
33. 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)
34. 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)