On Planning Through LLMs

Mattia Chiari[a;*], Luca Putelli[a], Nicholas Rossetti[a], Ivan Serina[a] and Alfonso Emilio Gerevini[a;**]

[a]University of Brescia

Abstract. In recent years, various studies have been carried out to assess whether Large Language Models (LLMs) possess different reasoning capabilities, including those required in automated planning. Typically, these studies provide the LLM with a planning domain and a problem, specified by an initial state and a goal, and require the LLM model to generate a plan solving the problem. Despite this common configuration, such studies significantly differ in the used models, the information provided to the model, the possible involvement of symbolic planners, and the experimental approaches used for the evaluation. Motivated by the growing interest in LLMs and in the understanding of their reasoning abilities, in this work we offer a concise review of recent studies on using LLMs for planning. We outline the main research trends and discuss their most notable findings. Furthermore, we identify key challenges and highlight critical aspects to consider when evaluating a LLM in terms of learning to plan and generating solution plans.

1 Introduction

The remarkable success obtained by pre-trained Large Language Models (LLMs) based on the Transformer architecture [43], such as the GPT models developed by OpenAI [27], have been opening new research lines which aim to understand the capabilities of such models. Due to the vast amount of textual data used in their training, they possess huge knowledge about real-world entities in subjects like geography or history [14]. Additionally, they can perform simple lexical operations [21], some common sense reasoning [6], and solve mathematical problems [45]. However, the claim that LLMs possess genuine reasoning abilities remains a subject of intense debate within the scientific community. Some studies, such as [1, 11, 46], argue that LLMs often simulate reasoning in constrained domains by leveraging statistical patterns in the data, allowing them to solve reasoning tasks without truly "understanding" them.

As a result, there has been a growing interest in studying LLMs using more rigorous and challenging tasks, such as those found in automated planning. Solving automated planning tasks involves understanding complex relationships between entities and objects, determining when actions can be executed, comprehending their consequences, and organizing actions to achieve a specific goal. These capabilities are interesting not only from a theoretical point of view but they could lead to an increase of efficiency, speed and quality with respect to the systems already available. Moreover, although LLMs (like all machine learning models) do not offer formal guarantees of always providing a correct solution, they hold the potential for integration with traditional planners to enhance performance. This approach mirrors existing efforts where machine learning systems calculate heuristics that are subsequently used by planners [16, 32], or where deep neural networks are applied for generalized planning [38] and goal recognition [3].

In recent years, several studies have explored the application of LLMs in automated planning [25, 33, 34, 40]. Despite the common goal, these studies vary in several key aspects, such as the LLM they consider and how that model is used: whether it is exploited through zero-shot or few-shot prompting [44], with a Chain-of-Thought [45], through a more complex fine-tuning process [25], or even training a GPT model specifically for planning [31]. Most notably, the results obtained, how they were evaluated, and the conclusions that can be drawn from them differ significantly. While the authors of [40] claim that "LLMs still can't plan", more promising results were obtained by [10, 31].

This paper aims to review and survey these studies, offering guidelines to contextualize their findings and highlighting their similarities and differences. We also discuss the methodologies used for evaluation and the achieved results. Finally, we examine the primary challenges and future directions in this emerging field, considering perspectives from both automated planning and deep learning.

2 Background

In this section, we provide an overview and the background on Large Language Models and Automated Planning.

2.1 *Transformer-based models and LLMs*

In 2017, Vaswani et al. propose a deep learning architecture called **Transformer** [43]. Although it was originally conceived for machine translation, in the following years Transformer-based architecture (such as GPT) became the state-of-the-art in most Natural Language Processing (NLP) tasks.

A transformer is made by two main parts: the Encoder, which converts a text sequence into an embedded representation, and the Decoder, which exploits the embedded representation in an autoregressive procedure to generate the translated sentence iteratively word by word. Both these parts are made by a stack of several layers (for instance, 12 in the original Transformer, 96 for GPT-3). A transformer processes text sequences first by separating them into smaller units called tokens (words or parts of words). Different NLP models use different tokenization methods and algorithms, such as Byte-Pair Encoding and WordPiece. After tokenization, an embedding layer converts each token into a corresponding real-valued vector. Therefore, an input text is transformed into a sequence of vectors.

The most important component in the Transformer architecture is the self-attention mechanism introduced in [43]. Intuitively, the self-

* Corresponding Author. Email: mattia.chiari@unibs.it
** Corresponding Author. Email: alfonso.gerevini@unibs.it

attention makes it possible to "pay attention" to different parts of the sentence and to incorporate this information into the embedded representation of each token. More formally, first the model projects the embedded representation of each word E into three new representations called *key* (K), *query* (Q) and *value* (V) by multiplying it with three weight matrices W_k, W_q and W_v. The new representation Z, is then calculated as $Z = softmax(\frac{QK^T}{\sqrt{d_k}})V$, where d_k is the size of the embedded representation. Transformer architectures combine several parallel attention mechanisms (in the so called Multi-Head Attention) with feed-forward neural layers and residual connections across all the encoding and decoding layers.

The works in [4] and [27] derived two models based on the two main parts of the Transformer architecture: **Bidirectional Encoder Representations from Transformers** (BERT), which is based on the Encoder, [4], and **Generative Pre-trained Transformer** (GPT) [27], which is based on the Decoder. BERT, as other Encoder-based models, is typically trained with *Masked Language Modeling*, i.e. by masking several words in input and having the model predict them. In contrast, GPT (which is the most famous Decoder-based model) is trained to generate text starting from an initial textual prefix. This procedure is also called *Causal Language Modeling*.

Training, Fine-tuning and Prompting Transformer-based models are typically pre-trained on a large corpora of unlabeled text to understand and generate language, enabling them to acquire vast knowledge. These models are referred to as pre-trained language models. Due to their scale, often comprising billions of parameters, they are also known as large language models (LLMs).

Pre-trained LLM's capabilities can be further enhanced by refining them on a smaller, task-specific dataset through a process called *fine-tuning*. Fine-tuning enables the model to specialize its general knowledge and better manage the complexities of a specific task. This is typically performed by attaching a simple feed-forward neural network to the model. This neural network is devoted to solving the given task and during its training the LLM weights are updated accordingly. Moreover, the performance of LLMs can be improved through the instruction tuning process, which also exploits reinforcement learning and human feedback[22]. Alternatively, it is possible to train a language model from scratch in order to perform a specific task [2]. However, this comes with the limitation that the model is restricted to perform only the specific task it was trained for, lacking the wide-ranging knowledge that pre-trained LLMs possess.

For Decoder-based models, and in particular GPT-3 and GPT-4 models, users can interact with LLMs by asking something in natural language and receiving the answer provided by the model. The request made to the model is typically called *prompt* and the overall process of interacting with the LLM in different ways is called *prompt engineering* [20]. More specifically, there are three main prompting approaches:

- **Zero-Shot**, in which the LLM is asked something without any examples;
- **Few-Shot**, in which the LLM is provided with some examples which can be used for understanding a more general strategy to address the user's request;
- **Chain-of-Thought**, in which the LLM is provided with a prompt that encourages a more refined inference process requiring multiple intermediate steps, which may correct mistakes and provide useful information progressively.

2.2 *Classical Planning*

We assume that the reader is familiar with the standard planning language PDDL [7] for representing deterministic, fully observable planning problems.

A classical planning task is a pair $P = (D, I)$ where D is a planning domain and I is a planning problem. The planning domain 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. The problem instance 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 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 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 next 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 plan if $s_{n+1} \in S_G$. The cost of a plan is assumed to be given by its length, and a plan is optimal if there is no shorter plan.

3 Literature selection

We conducted a comprehensive survey of the existing literature to explore the recent trend of using language models as tools for automated plan generation. This research identified 18 published papers that present different approaches to leverage language models for plan generation within automated planning. Our search spanned multiple academic databases, conferences, and journals, using keywords such as *LLM in planning*, *Reasoning with LLM*, *Heuristics with LM*, and *Transformer*, among others. Over the past three years, most of the papers we reviewed have been presented at prominent conferences in artificial intelligence (IJCAI, AAAI), deep learning (NIPS, ICML, EMNLP), automated planning (ICAPS), and robotics (ICRA) highlighting the growing interest in LLMs and their potential reasoning capabilities in this field. From the initial pool of results, our selection was driven by the currently available evidence on the significance of the works following two criteria:

(i) Peer-reviewed publications: We prioritized papers that have undergone rigorous peer review, ensuring that they have been critically evaluated and recognized as valuable contributions by experts in the field. Many of these works are published in top-tier conferences, where the acceptance process is highly competitive.

(ii) Citation impact: While we acknowledge that citation count alone does not determine the quality of a work, it does serve as an indicator of its consideration in the community. We included works that have gained substantial citations ($\geq$ 20 citations), even if they had not been peer-reviewed.

With respect to the general survey presented in [26], we carefully examined these works and selected the studies that specifically address the use of LLMs for plan generation only. Therefore, we are able to provide a much more in-depth analysis of LLM-based planning methods according to their input-output type, their procedures and takeaways, and how they are integrated with symbolic tools.

4 Overview of LLM-based methods for Generating Plans

First, we present a general description of the claims and contributions from the articles we reviewed.

A significant line of work in the examined literature concerns the prompting capabilities of pre-trained LLMs in terms of reasoning and planning without external validation. [40] analyzed GPT-3, and proposed a benchmark to address these experiments in a thorough way [41]. In these works, planning problems are provided to an LLM (in the form of natural language or PDDL), and the study focuses on evaluating their behaviour under different conditions and prompts. As summed up in [40], the takeaway of these works is that LLMs by themselves cannot solve planning tasks.

Nonetheless, several studies have attempted to leverage the basic reasoning capabilities of LLMs and to refine them for planning tasks in more sophisticated ways. These approaches include using chain-of-thought prompting combined with LLM-based validators [35] and symbolic validators [8, 15, 36, 47] or integrating search algorithms and planners [9, 19, 34, 42]. Another interesting approach is presented in [33], in which GPT-4 is used to generate Python programs that can solve various planning problems in the same domain. The results and the conclusions across these studies vary significantly. Some works express optimism about LLMs' potential in planning, such as [9, 33, 47], while others, like [15], offer a more cautious and pessimistic outlook on their effectiveness in these tasks.

Most of the studies mentioned above focus on closed-source, pre-trained models like GPT-3.5 and GPT-4, which have gained significant attention due to their high relevance and commercial success. However, a major limitation of using such models is the challenge of adapting them to specific tasks like automated planning. To address this, several studies have fine-tuned or even trained from scratch smaller Transformer-based models, aiming to create specialized language models tailored to solving planning tasks. One of the first specialized models is Plansformer [25], which utilizes a fine-tuned T5 model [28] and generates a corresponding solution plan given a formalized planning problem in PDDL. PlanGPT [31] achieved better results by training a GPT-2 model from scratch, and integrating a validator [30] and a planner [39]. A more complex configuration has been studied in [10] by using three different Language Models to generate and evaluate actions. Another interesting approach is presented in [18], in which the authors train a Transformer to emulate the A* algorithm. Finally, the works in [12] focuses on generating valid actions and heuristic values with LLMs.

In the following sections, we propose additional categorizations of these works, examining key aspects such as their input and output, how they utilize LLMs, what LLMs they used, and whether they incorporate validators or planners into their analyses. For clarity of the pictures, in Figures 1,2 and 3 we named the considered papers with the first three letters of the first author's surname, followed by the last two digits of the publication year. For instance, [8] is reported as [GUA23].

5 LLM Procedures to Solve Planning Problems

In this section, we provide a more detailed description on how LLMs are exploited to solve planning problems. First, we discuss their input and output. Next, we analyze the procedures used for interacting with the LLM and the results.

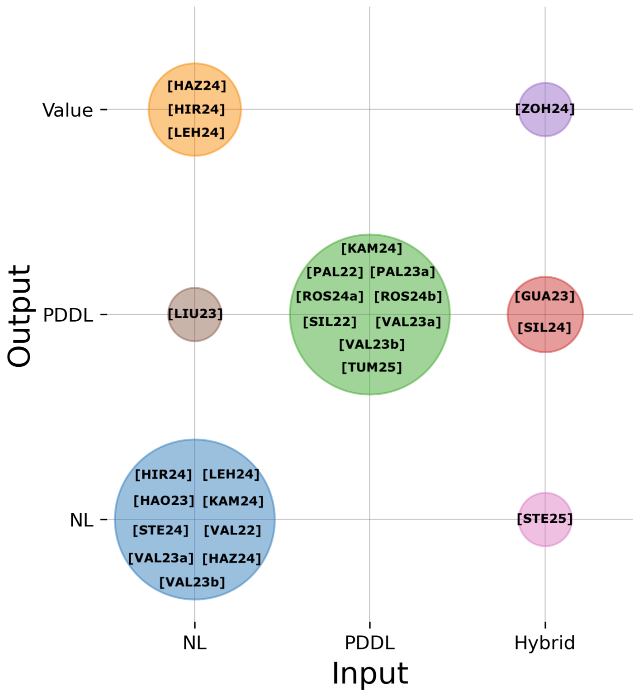

Figure 1: Visualization of the input categories (on the x-axis) and output categories (on the y-axis) of the considered LLM approaches for automated planning. Each work is identified by the first three letters of the first author's surname, followed by the last two digits of the publication year. NL stands for Natural Language, Heur stands for Heuristics.

5.1 *Input and Output*

As illustrated on the x-axis of Figure 1, there are three main ways to provide an automated planning problem as input to a LLM: the first is using **Natural Language** (**NL**), typically English without an explicit logical formalism; the second is using **PDDL**; the third (**Hybrid**) is through a combination of PDDL and prompts based on natural language.

More specifically, the approaches based on Natural Language [40, 42] translate a PDDL domain and a specific problem into a series of simple sentences understandable by a LLM using a custom domain specific translator. Other approaches, such as [25, 31, 39], work directly with PDDL, focusing on processing formalized planning problems based on their logical structure. Hybrid approaches typically use the PDDL formalism combined with natural language to interact with the model via external verifiers and assist the generation [8, 33, 36]. Notably, the work by Silver et al. employs a hybrid input strategy, where the system receives not only PDDL but also code fragments and error traces.

In terms of the output generated by the LLM, as shown on the y-axis of Figure 1, the works we considered can be divided into three main categories. The first is composed by works (such as [15, 36, 42]) in which the LLM generates an answer written in natural language (NL) from which a PDDL plan is derived. Similarly, the second category is composed by works in which the LLM directly generates a valid sequence of actions to solve problem in PDDL, such as [8, 19]. A notable alternative approach is presented in [33], where the LLM generates a Python program that includes a policy designed to solve the planning problem writing the plan in PDDL. In

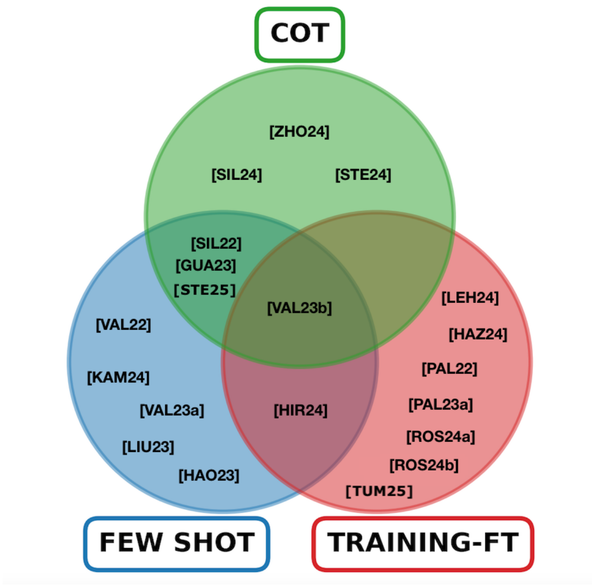

Figure 2: Venn Diagram of the different ways to exploit LLMs for generating plans: Zero and Few-Shot Prompting, in blue, Training and Fine-Tuning, in red and Chain-of-Thought, in green.

the third category, called Heuristics in Figure 1, LLMs are used to compute heuristic values, which can then be combined with search algorithms to generate valid plans. For instance, in [12] the authors train the decoder component of a T5 model to predict heuristic values, while Hao et al. [9] calculates heuristic values by combining the action's likelihood with the LLM's confidence. A hybrid approach is adopted by Hazra, Martires, and De Raedt [10], where knowledge learned by two LLMs, focused on action applicability and best action selection, guides a beam search carried out by a third pre-trained LLM, integrating multiple models to enhance the planning process.

5.2 *Procedures and Takeaways*

Another critical aspect of the considered works is how LLMs are exploited to solve planning problems, and the results they obtained. As we show in Figure 2, we identified three main strategies attempted: **Zero and Few-Shot** prompting, **Chain-of-thought** (CoT) prompting, and **Fine-tuning and Training**.

In the studies belonging to the first category, the authors analyze the reasoning capabilities of pre-trained LLMs using a prompting setting without external validation. They show that LLMs possess poor planning abilities, even when handling simple problems, as they struggle to reason about action applicability and its effects, generating invalid plans. These capabilities do not increase by providing examples [40].

For the second category, some researches have proposed enhancing LLM capabilities by making them process the problem step-by-step, as in the Chain-of-Thought, or even combining them with reasoning tools, for instance the VAL validator [13] or a Python interpreter [33, 36]. Instead, the work in [35] utilizes another LLM in a sort of self-verification tool to detect and correct errors. The use of external validators has yielded mixed results. The approach has produced excellent outcomes in [33, 47]. However, in other studies, like [8, 34, 35], the integration of validation tools led to only modest improvements.

Finally, considering the studies of the third category, some authors

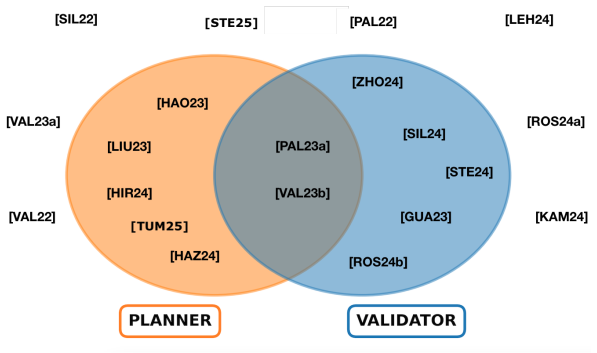

Figure 3: Venn Diagram of the integration with Validators (reported in orange) and Planners (reported in blue). The approaches that do not integrate with either a planner or a validator are reported outside the two sets.

fine-tune and train smaller Transformer-based models on planning datasets to instruct these models how to generate a plan. This marks an important difference between the other two categories. In fact these works, instead of relying solely on the knowledge previously gained by a pre-trained LLM, try to inject some domain-specific planning knowledge into them. Among these works, the creators of Plansformer [23, 24, 25] fine-tuned a Transformer-base model (called CodeT5 [28]) trained on code of several programming language. The results show that the fine-tuning significantly improves the planning capabilities of the model, up to a coverage higher than 80% across 6 planning domains.

Instead of fine-tuning an LLM, in [30, 31, 39], a new GPT model, called PlanGPT, is trained from scratch to learn a general policy to solve many planning instances in a given domain. Exploiting a large amount of training data (about 63000 planning problems per domain), PlanGPT obtains impressive results in terms of coverage (more than 90% on IPC problems in Blocksworld, Driverlog, Floortile, Visitall and Zenotravel).

6 Integration with Validators and Planners

This section provides an overview of how the considered studies integrate LLMs with symbolic tools, in particular with reasoning-based validators, such as VAL [13] and planners. These three forms (No Integration, Validator and Planner) are shown in Figure 3.

Analyzing the approaches which exploit validators, the work in [8] integrates the VAL validator [13] during the LLM-based plan generation, reporting poor results. Conversely, the study in [47] achieves improved coverage over standalone LLM performance on various benchmarks using the same approach. Additionally, [33] combines VAL and Python validators within the LLM workflow to enhance program synthesis and address domain-specific planning tasks.

Among the approaches that integrate LLMs with planners, three works leverage a planner to modify a candidate plan, which may be valid or invalid, generated by a LLM [34, 39, 42]. The core idea behind these works is that even an imperfect, LLM-generated plan can provide helpful information that a planner can leverage to enhance performance. More specifically, in [42], a pre-trained GPT model on a general text corpus is combined with the LPG planner [5]. Differently, the work in [34] employs the plan resulting from the LLM to initialize the queue of expanded nodes in Greedy Best First Search (GBFS), using the LLM-generated candidates as a starting point for the search process. These two studies demonstrate that combining neural-based LLMs with symbolic planning approaches improves overall performance over LLMs alone, and reduces the search space compared to planners alone. However, these approaches are slower than using satisficing planners such as LPG [5] and LAMA [29].

Considering the approaches that focus on computing heuristics with LLMs rather than generating plans directly, we can see also a more in-depth integration among LLMs and search-based techniques. The most notable papers are [12] and [9], which combine LLM-derived heuristics with Greedy Best First Search (GBFS) and Monte Carlo Tree Search (MCTS), respectively. In these works, starting from the current state s of a planning problem, the LLM assigns a heuristic value to the next states derived from $A(s)$. This process guides the search by prioritizing the expansion of states with the highest heuristic values until all goals are satisfied. Although both these works share a similar idea, it is important to note that the scoring function in [12] derives directly from the actions which the LLM associates to a higher probability. A more complex approach is presented in [9] which combines the probabilities of actions and states calculated by the LLM with a task-specific heuristic to obtain the final heuristic value. However, computing such heuristics is computationally very expensive, as an LLM must be prompted for each new applicable state. Moreover, there is no in-depth study of how these approaches compare to heuristics based on reasoning and planners.

7 Planning Capabilities Evaluation

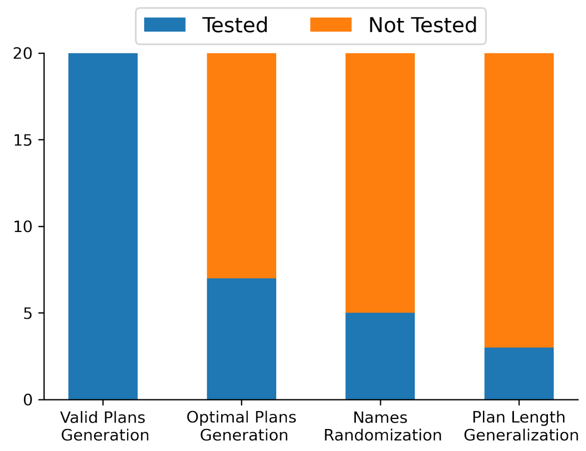

Figure 4: Types of evaluation conducted by the considered studies, in terms of Valid Plans Generations, Optimal Plans Generation, and whether an experiment with randomized names (Names Randomization) or with plans outside the training/fine-tuning distribution (Plan Length Generalization) is present . For each column, the blue bar represents the number of papers which perform the actual test, while the orange bar represents the number of papers which does not perform the test.

Among the works we considered in this study, there is a great variety in how the LLMs are evaluated in terms of their planning capabilities. While establishing a standardized evaluation method across these systems, which vary significantly in architectures and setups (as explained in the previous sections), is not the main scope of this paper, we can identify several common evaluation points shared across the studies.

Figure 4 reports an overview of the main features commonly evaluated among the considered works. The most important aspect is undoubtedly whether they are able to generate valid plans. This is evaluated in terms of coverage, i.e. the percentage of planning problems correctly solved by the LLM-based technique proposed by the paper. In fact, as expected, all the papers assess whether the proposed architectures can generate a valid plan (i.e. that does not violate any action precondition and reaches all the problem goals). However, several studies also verify whether the solution provided is optimal [10, 24, 40, 41] or perform a more general evaluation of the plan quality, by comparing the quality of the generated plan with the optimal one [40, 41]. Interestingly, to improve the quality of the generated plans, the works by [24] and [10] proposed neural language models specifically fine-tuned with optimal plans.

Another interesting experiment performed in the considered works is evaluating the impact of object and action names in PDDL domains and problems on the models' performance [24, 34, 42]. In most benchmark domains, the names used for objects and actions are derived from English words, which the LLMs likely encountered during their training. This evaluation aims to strip away the models' reliance on their linguistic knowledge to assess purely their reasoning ability to generate a plan. In [34], the objects and actions names are replaced by English words with different meanings. The work in [41, 42] tests the effects of randomization by using both random strings and different words. The results indicate that all tested LLMs have a substantial drop in performance, producing almost no valid plan.

Considering the works that fine-tuned or even trained LLMs from scratch, those in [31, 39] assessed only the plan generation capability on problems matching the complexity level of the training set. Moreover, they do not test the effects of randomization and word substitution. In contrast, Plansformer [24] testes both randomization and used problem with a small increase in the number of objects than those seen during training. Both tests lead to a performance decline, with Plansformer solving only a small portion of the tested problems. Comparable generalization limitations were also observed in SayCanPay [10]. More promising results on generalization are shown in [12], where the proposed hybrid architecture manages to generalize to more complex problems that were not seen during training.

8 Discussion and Conclusions

We have analyzed the main characteristics of the works concerning the relationship among LLMs and automated planning, identifying various approaches for testing and developing new models. Our brief survey and categorization of these works highlight several differences in terms of input and output (typically, text versus PDDL), exploitation of pre-trained models (prompting, fine-tuning, or even training from scratch), integration of a planner or a plan validator. In the following, we offer some conclusions and a discussion to place the works we analyzed within a broader context.

8.1 Are LLMs Capable of Planning?

In order to understand whether LLMs can be used effectively as planners, it is important to understand the most crucial planners' properties and how they align with LLMs. Three fundamental properties in planning are completeness, soundness and domain independence. A planner is complete if it always finds a solution when a solution exists, and it is sound if the generated plans are guaranteed to be valid solutions. The models presented in the reviewed works meet at most one of these properties. For instance, approaches that exploit zero-shot and few-shot prompting on pre-trained LLMs, such as GPT-3, are neither sound nor complete. Moreover, as shown in [34, 40, 42], they have limited performance in terms of problem coverage.

Considering works that train or fine-tune an LLM (such as [10, 23]), they can be considered sound for a limited set of instances if their solution is authenticated by a validator. However, these approaches are not complete. Since they are trained on a finite vocabulary that includes a specific number of objects, the models are constrained by this vocabulary and extending the model's capability to handle a greater number of objects would require a complete retraining of the system. Clearly, this limits the flexibility and generalization capabilities of LLMs beyond the specific instances they were trained on. Moreover, all these approaches have been realised for specific domains, and therefore they are not domain independent. Finally, the preliminary results from these fine-tuned or trained LLMs suggest they struggle with solving more complex problems, especially those that exceed the complexity of the tasks used during training [24]. Consequently, while these models can be helpful within their trained scope, they cannot fully generalize to a broader range of planning tasks without significant adjustments.

The highlighted limitations of LLMs in planning can be considered as part of a bigger problem that is currently being addressed by the research community, called the "stochastic parrot" problem [1, 11]. This problem posits that LLMs only mimic human language without possessing relevant reasoning capabilities and a real "understanding" of semantics. In planning, the stochastic parrot problem would justify the identified limitations and the scarce generalization performance that the LLMs (in particular, the trained and fine-tuned ones) achieve. Consider for instance the generated plans reported in [40] for the Blocksworld domain. We can see that LLMs can generate plausible-sound solutions, in which the generated plan consists of stacking and unstacking blocks. However, they result in invalid plans. Similar results can be seen in [31, 33]. In other words, these results could suggest that LLMs may parrot back learnt patterns, without a proper understanding of complex reasoning tasks, such as planning. However, these aspects are widely debated in the scientific community, and today there is no definitive conclusion regarding the true nature of LLMs capabilities or the best way to address these challenges.

8.2 How Can We Have a Fair Evaluation?

In the evaluation process, typically the ability of LLMs to generate valid plans is assessed. Additionally, some works test generation robustness in terms of names and actions randomization, and problem complexity. However, a contention point among the considered articles concerns the datasets and benchmarks employed.

A first benchmark, PlanBench, was proposed in [41], where the authors introduce 600 hand-crafted problems in natural language for the Blocksworld and Logistics domains in order to test the planning capabilities of LLMs, even with respect to several forms of randomizations of names and fluents. However, this benchmark has not been adopted in other works yet. Furthermore, it covers only two domains, which limits its ability to represent the full complexity of automated planning tasks. Other planning benchmarks can be obtained from the 2023 International Planning Competitions (IPC) [37], which includes a broader range of domains but offers a limited number of problems per domain. While these problems are sufficient to test symbolic planners, they are often too few to evaluate deep learning systems effectively, which typically require large and diverse test sets (such as the 20% of the overall dataset, which is typically built of thousands of instances) with problems of different types and sizes. This is particularly relevant given the importance of ensuring that a deep learning model (such as a LLM) has genuinely learned to solve planning tasks, rather than just memorizing solutions from the training data. One of the first attempts to solve this problem appears in [31], where the authors published a dataset containing 5000 problems for eight planning domains with a complexity similar to the IPC setup.

Another factor that needs to be evaluated is LLMs' capability to generate correct plans following order permutations of the fluents in the initial state and the goals or name randomization (of objects, fluents, and actions). This capability is fundamental because the reasoning process should be independent of the order of its fluents or name permutation, meaning that the same logical plan should be generated regardless of these variations. For example, considering the Blocksworld domain, the LLM should produce the same plan whether an object named $BlockA$ is renamed to $BlockY$ or if the fluents describing the initial state or goal are presented in a different order.

However, it is important to note that the knowledge of LLMs trained on text has been acquired from processing documents that reflect and describe the real world. Therefore, their ability to solve, for instance, a logistic problem stems not only from the logical structure of the problem, but also from understanding its main components, such as what is a *plane* object. Randomizing or replacing these meaningful object names with gibberish, while keeping the formal structure of the problem, could significantly impair the LLM's capabilities. The model may rely on some associations between real-world entities, learned during its pre-training, and their roles in planning contexts. By removing these associations, the LLM could struggle to generate a correct plan, as it would no longer recognize the objects or actions in a way that reflects their real-world functions. This behaviour contrasts sharply with the reasoning process of traditional planners. Unlike LLMs, planners are designed to be unaffected by word randomization because they treat words as mere placeholders without any intrinsic meaning. Planners rely solely on the logical relationships and formal structures defined by the problem specification, not on any semantic understanding of the objects or actions *names*.

Finally, an important aspect to assess is how LLMs handle complex planning problems, particularly those involving an increased numbers of objects, potential dead ends, and a high branching factor of fluents and actions within the domain. For training and fine-tuning approaches, where models are specifically trained on problems of a certain complexity, it is crucial to determine whether these models can generalize to more complex problems (in terms of number of objects and a higher branching factor) than those used during training. Suppose an LLM can solve problems beyond the complexity of its training dataset, in that case, we have clear evidence that the model has learned generalizable reasoning rather than merely memorizing solutions from the training data. However, if the model's performance deteriorates when faced with more complex problems, this indicates that the training process has limited generalization properties and that the LLM may only be effective within a narrow range of specific problems [24].

8.3 Limitations

It is important to emphasize that all fine-tuning and training models are domain-specific. Even the current best-performing architecture, PlanGPT [31], was implemented and tested by training several domain-specific models, each with its vocabulary and dataset. These models can learn a general policy and, therefore, solve many planning problems in a specific domain, but need different training for different domains. Therefore, their results should be analyzed and understood carefully to detect whether they demonstrate some forms of reasoning or, instead, learn some specific statistical patterns from the training data. In contrast, general-purpose LLMs trained on text are not specifically built for any particular domain and should be more robust. However, to date, they are not able to plan even if they are trained with massive datasets and refined with human feedback [41].

Moreover, the training and evaluation of these architectures require huge computational resources and, as shown in [34], they can be slower than classical planners such as LAMA. These issues, along with the reliance on world knowledge (often expressed in PDDL) for approaches that use chain-of-thought reasoning, external validators, or human experts, leave open the question of whether in practice it is convenient to use LLMs to planning tasks, given that current domain independent (classical) planners are already highly efficient and optimized. An interesting direction for future work is combining these two technologies, taking the best from both worlds [9, 12, 17, 39].

9 Acknowledgements

This work has been supported by MUR PRIN-2020 project RIPER (n. 20203FFYLK), and by PNRR MUR project PE0000013-FAIR, cascade funding call, ResilientPlans.

References

[1] Emily M. Bender, Timnit Gebru, Angelina McMillan-Major, and Shmargaret Shmitchell, 'On the dangers of stochastic parrots: Can language models be too big?', in *FAccT*, pp. 610–623. ACM, (2021).

[2] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Michael Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch, 'Decision transformer: Reinforcement learning via sequence modeling', in *NeurIPS*, pp. 15084–15097, (2021).

[3] Mattia Chiari, Alfonso Emilio Gerevini, Francesco Percassi, Luca Putelli, Ivan Serina, and Matteo Olivato, 'Goal recognition as a deep learning task: The grnet approach', in *ICAPS*, pp. 560–568. AAAI Press, (2023).

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova, 'BERT: pre-training of deep bidirectional transformers for language understanding', in *NAACL-HLT (1)*, pp. 4171–4186. Association for Computational Linguistics, (2019).

[5] Alfonso Gerevini and Ivan Serina, 'LPG: A planner based on local search for planning graphs with action costs', in *AIPS*, pp. 13–22. AAAI, (2002).

[6] Mor Geva, Daniel Khashabi, Elad Segal, Tushar Khot, Dan Roth, and Jonathan Berant, 'Did Aristotle use a laptop? A question answering benchmark with implicit reasoning strategies', *Trans. Assoc. Comput. Linguistics*, **9**, 346–361, (2021).

[7] Malik Ghallab, Craig Knoblock, David Wilkins, Anthony Barrett, Dave Christianson, Marc Friedman, Chung Kwok, Keith Golden, Scott Penberthy, David Smith, Ying Sun, and Daniel Weld, 'Pddl - the planning domain definition language', (08 1998).

[8] Lin Guan, Karthik Valmeekam, Sarath Sreedharan, and Subbarao Kambhampati, 'Leveraging pre-trained large language models to construct and utilize world models for model-based task planning', in *NeurIPS*, (2023).

[9] Shibo Hao, Yi Gu, Haodi Ma, Joshua Jiahua Hong, Zhen Wang, Daisy Zhe Wang, and Zhiting Hu, 'Reasoning with language model is planning with world model', in *EMNLP*, pp. 8154–8173. Association for Computational Linguistics, (2023).

[10] Rishi Hazra, Pedro Zuidberg Dos Martires, and Luc De Raedt, 'Saycanpay: Heuristic planning with large language models using learnable domain knowledge', in *AAAI*, pp. 20123–20133. AAAI Press, (2024).

[11] Michael Townsen Hicks, James Humphries, and Joe Slater, 'Chatgpt is bullshit', *Ethics Inf. Technol.*, **26**(2), 38, (2024).

[12] Eran Hirsch, Guy Uziel, and Ateret Anaby-Tavor, 'What's the plan? evaluating and developing planning-aware techniques for llms', *CoRR*, **abs/2402.11489**, (2024).

[13] Richard Howey, Derek Long, and Maria Fox, 'VAL: automatic plan validation, continuous effects and mixed initiative planning using PDDL', in *ICTAI*, pp. 294–301. IEEE Computer Society, (2004).

[14] Zhengbao Jiang, Frank F. Xu, Jun Araki, and Graham Neubig, 'How can we know what language models know', *Trans. Assoc. Comput. Linguistics*, **8**, 423–438, (2020).

[15] Subbarao Kambhampati, Karthik Valmeekam, Lin Guan, Mudit Verma, Kaya Stechly, Siddhant Bhambri, Lucas Saldyt, and Anil Murthy, 'Position: Llms can't plan, but can help planning in llm-modulo frameworks', in *ICML*. OpenReview.net, (2024).

[16] Rushang Karia and Siddharth Srivastava, 'Learning generalized relational heuristic networks for model-agnostic planning', in *AAAI*, pp. 8064–8073. AAAI Press, (2021).

[17] Michael Katz, Harsha Kokel, Kavitha Srinivas, and Shirin Sohrabi, 'Thought of search: Planning with language models through the lens of efficiency', in *NeurIPS*, (2024).

[18] Lucas Lehnert, Sainbayar Sukhbaatar, Paul McVay, Michael Rabbat, and Yuandong Tian, 'Beyond a*: Better planning with transformers via search dynamics bootstrapping', *CoRR*, **abs/2402.14083**, (2024).

[19] Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone, 'LLM+P: empowering large language models with optimal planning proficiency', *CoRR*, **abs/2304.11477**, (2023).

[20] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig, 'Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing', *ACM Comput. Surv.*, **55**(9), (January 2023).

[21] Avinash Madasu and Shashank Srivastava, 'What do large language models learn beyond language?', in *EMNLP (Findings)*, pp. 6940–6953. Association for Computational Linguistics, (2022).

[22] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F. Christiano, Jan Leike, and Ryan Lowe, 'Training language models to follow instructions with human feedback', in *NeurIPS*, (2022).

[23] Vishal Pallagani, Bharath Muppasani, Keerthiram Murugesan, Francesca Rossi, Lior Horesh, Biplav Srivastava, Francesco Fabiano, and Andrea Loreggia, 'Plansformer: Generating symbolic plans using transformers', *CoRR*, **abs/2212.08681**, (2022).

[24] Vishal Pallagani, Bharath Muppasani, Keerthiram Murugesan, Francesca Rossi, Biplav Srivastava, Lior Horesh, Francesco Fabiano, and Andrea Loreggia, 'Understanding the capabilities of large language models for automated planning', *CoRR*, **abs/2305.16151**, (2023).

[25] Vishal Pallagani, Bharath Muppasani, Biplav Srivastava, Francesca Rossi, Lior Horesh, Keerthiram Murugesan, Andrea Loreggia, Francesco Fabiano, Rony Joseph, and Yathin Kethepalli, 'Plansformer tool: Demonstrating generation of symbolic plans using transformers', in *IJCAI*, pp. 7158–7162. ijcai.org, (2023).

[26] Vishal Pallagani, Bharath C. Muppasani, Kaushik Roy, Francesco Fabiano, Andrea Loreggia, Keerthiram Murugesan, Biplav Srivastava, Francesca Rossi, Lior Horesh, and Amit P. Sheth, 'On the prospects of incorporating large language models (llms) in automated planning and scheduling (APS)', in *ICAPS*, pp. 432–444. AAAI Press, (2024).

[27] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al., 'Improving language understanding by generative pre-training', (2018).

[28] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu, 'Exploring the limits of transfer learning with a unified text-to-text transformer', *J. Mach. Learn. Res.*, **21**, 140:1–140:67, (2020).

[29] Silvia Richter and Matthias Westphal, 'The LAMA planner: Guiding cost-based anytime planning with landmarks', *J. Artif. Intell. Res.*, **39**, 127–177, (2010).

[30] Nicholas Rossetti, Massimiliano Tummolo, Alfonso Emilio Gerevini, Matteo Olivato, Luca Putelli, and Ivan Serina, 'Enhancing gpt-based planning policies by model-based plan validation', in *NeSy (2)*, volume 14980 of *Lecture Notes in Computer Science*, pp. 328–337. Springer, (2024).

[31] Nicholas Rossetti, Massimiliano Tummolo, Alfonso Emilio Gerevini, Luca Putelli, Ivan Serina, Mattia Chiari, and Matteo Olivato, 'Learning

general policies for planning through GPT models', in *ICAPS*, pp. 500–508. AAAI Press, (2024).

[32] William Shen, Felipe W. Trevizan, and Sylvie Thiébaux, 'Learning domain-independent planning heuristics with hypergraph networks', in *ICAPS*, pp. 574–584. AAAI Press, (2020).

[33] Tom Silver, Soham Dan, Kavitha Srinivas, Joshua B. Tenenbaum, Leslie Pack Kaelbling, and Michael Katz, 'Generalized planning in PDDL domains with pretrained large language models', in *AAAI*, pp. 20256–20264. AAAI Press, (2024).

[34] Tom Silver, Varun Hariprasad, Reece S Shuttleworth, Nishanth Kumar, Tomás Lozano-Pérez, and Leslie Pack Kaelbling, 'Pddl planning with pretrained large language models', in *NeurIPS 2022 Foundation Models for Decision Making Workshop*, (2022).

[35] Kaya Stechly, Karthik Valmeekam, and Subbarao Kambhampati, 'On the self-verification limitations of large language models on reasoning and planning tasks', *CoRR*, **abs/2402.08115**, (2024).

[36] Katharina Stein, Daniel Fišer, Jörg Hoffmann, and Alexander Koller, 'Automating the generation of prompts for llm-based action choice in pddl planning', in *ICAPS 2025*, (2025).

[37] Ayal Taitler, Ron Alford, Joan Espasa, Gregor Behnke, Daniel Fiser, Michael Gimelfarb, Florian Pommerening, Scott Sanner, Enrico Scala, Dominik Schreiber, Javier Segovia-Aguas, and Jendrik Seipp, 'The 2023 international planning competition', *AI Mag.*, **45**(2), 280–296, (2024).

[38] Sam Toyer, Sylvie Thiébaux, Felipe W. Trevizan, and Lexing Xie, 'Asnets: Deep learning for generalised planning', *J. Artif. Intell. Res.*, **68**, 1–68, (2020).

[39] Massimiliano Tummolo, Nicholas Rossetti, Alfonso Emilio Gerevini, Luca Putelli, Ivan Serina, and Matteo Olivato, 'Integrating classical planners with gpt-based planning policies', in *AI*IA*, Lecture Notes in Computer Science. Springer, (2025).

[40] Karthik Valmeekam, Alberto Olmo Hernandez, Sarath Sreedharan, and Subbarao Kambhampati, 'Large language models still can't plan (A benchmark for llms on planning and reasoning about change)', *CoRR*, **abs/2206.10498**, (2022).

[41] Karthik Valmeekam, Matthew Marquez, Alberto Olmo Hernandez, Sarath Sreedharan, and Subbarao Kambhampati, 'Planbench: An extensible benchmark for evaluating large language models on planning and reasoning about change', in *NeurIPS*, (2023).

[42] Karthik Valmeekam, Matthew Marquez, Sarath Sreedharan, and Subbarao Kambhampati, 'On the planning abilities of large language models - A critical investigation', in *NeurIPS*, (2023).

[43] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin, 'Attention is all you need', in *NIPS*, pp. 5998–6008, (2017).

[44] Jason Wei, Maarten Bosma, Vincent Y. Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V. Le, 'Finetuned language models are zero-shot learners', in *ICLR*. OpenReview.net, (2022).

[45] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou, 'Chain-of-thought prompting elicits reasoning in large language models', in *NeurIPS*, (2022).

[46] Honghua Zhang, Liunian Harold Li, Tao Meng, Kai-Wei Chang, and Guy Van den Broeck, 'On the paradox of learning to reason from data', in *IJCAI*, pp. 3365–3373. ijcai.org, (2023).

[47] Zhehua Zhou, Jiayang Song, Kunpeng Yao, Zhan Shu, and Lei Ma, 'ISR-LLM: iterative self-refined large language model for long-horizon sequential task planning', in *ICRA*, pp. 2081–2088. IEEE, (2024).