PLANSFORMER: GENERATING SYMBOLIC PLANS USING TRANSFORMERS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large Language Models (LLMs) have been the subject of active research, significantly advancing the field of Natural Language Processing (NLP). From BERT to BLOOM, LLMs have surpassed state-of-the-art results in various natural language tasks such as question answering, summarization, and text generation. Many ongoing efforts are focused on understanding LLMs' capabilities, including their knowledge of the world, syntax, and semantics. However, extending the textual prowess of LLMs to symbolic reasoning has been slow and predominantly focused on tackling problems related to the mathematical field. In this paper, we explore the use of LLMs for automated planning - a branch of AI concerned with the realization of action sequences (plans) to achieve a goal, typically for execution by intelligent agents, autonomous robots, and unmanned vehicles. We introduce Plansformer[1]; an LLM fine-tuned on planning problems and capable of generating plans with favorable behavior in terms of correctness and length with minimal knowledge-engineering efforts. We also demonstrate the adaptability of Plansformer in solving different planning domains with varying complexities, owing to the transfer learning abilities of LLMs. For one configuration of Plansformer, we achieve 97% valid plans, out of which 95% are optimal for Towers of Hanoi - a puzzle-solving domain.

1 INTRODUCTION

Large Language Models (LLMs), based on the transformer-based (neural) architecture Vaswani et al. (2017); Devlin et al. (2018); Brown et al. (2020); Chowdhery et al. (2022), have greatly advanced the field of Natural Language Processing (NLP). Their employment has grown dramatically in recent times Li (2022), as researchers develop newer and bigger LLMs. From BERT to the most recent BLOOM, language models have surpassed state-of-the-art results in various natural language tasks. For example, PaLM Chowdhery et al. (2022), achieved breakthrough performance on a plethora of natural language tasks such as inference, question answering, and commonsense reasoning, and outperformed an average human performance on the BIG-bench benchmark.

Despite the textual prowess of LLMs, their significance is very limited in the domains that involve symbols. For example, LLMs for domains with symbols such as mathematics Hendrycks et al. (2021b); Cobbe et al. (2021), and coding problems Hendrycks et al. (2021a); Chen et al. (2021) deliberates the failures of LLMs when it comes to handling symbols. In Automated Planning, Valmeekam et al. (2022) suggests that even state-of-the-art LLMs cannot reason with symbolic data and offer a new suite of benchmarks to test their reasoning capabilities. LLMs have also been studied to generate code, see, for example, CodeT5 Wang et al. (2021). In this paper, we propose to exploit LLMs trained to generate code and adapt them to generate valid plans. To achieve this, we first create a training and test dataset for several planning domains to support the research in LLM-based automated planning. Then, we present, Plansformer, an LLM trained to generate symbolic plans of high quality in terms of correctness and length. Instead of using a natural language-based pretrained model such as T5, in this study, we use CodeT5 (base), a transformer-based code generation model that achieves state-of-the-art results in CodeXGlue, as the pre-trained LLM due to its ability to generate goal-directed, sequential instruction and semantically meaningful program codes with syntactic and structural constraints. Our experimental tests show that the syntactic/symbolic

[1]The dataset, code, and all fine-tuned model checkpoints will be released for public usage.

knowledge learned from different programming languages in the CodeT5 model can be beneficial for the PDDL-based automated planning task. One configuration of Plansformer tested on a puzzle solving domain, Towers of Hanoi - **hanoi**, was able to generate 97% valid plans, out of which 95% are shortest length plans. This reveals a promising direction to harness LLMs also for symbolic tasks such as planning.

In the remainder of the paper, we present preliminaries on planning and language models and then describe, Plansformer, our main contribution. We then present results comparing our approach with off-the-shelf planners and other large language models. Furthermore, we discuss and show the ability of Plansformer to adapt to other domains and brief about the relevance to instruction generation. We conclude with a discussion of the results and presentation of ongoing work.

2 BACKGROUND

2.1 AUTOMATED PLANNING

An agent solving a planning problem is given the initial and goal states and a set of legal actions, and the objective is to find a sequence of actions that will take it from the initial to the goal state. In this paper, we adopt the Planning Domain Description Language (PDDL) McDermott et al. (1998); Fox & Long (2003) notations. In PDDL, a planning environment is described in terms of objects in the world, predicates that describe relations between these objects, and actions that bring change to the world by manipulating these relations. The output plan consists of a series of time steps, each of which can have one or more instantiated actions with concurrency semantics Ghallab et al. (2004).

A planner finds plans by searching in the space of states, where a state is a configuration of the physical objects or partial plans. In the most basic formulation, called classical planning, there is a single agent, the actions have unit cost, take constant time to execute, have deterministic effects, with the fully observable world, domain-specific conditions/constraints and all goals have to be achieved Ghallab et al. (2004). In more sophisticated planning settings, many of these conditions are relaxed - there may be multiple agents, the cost and duration of actions can be non-uniform while its effects can be non-deterministic, the world can be partially observable, and the agent may maximize as many goals as it can achieve in a given time and resource budget. A plan that just achieves the goals is called a *satisficing plan* while a plan which achieves the goals with minimum cost is called an *optimal plan*.

2.2 LARGE LANGUAGE MODELS AND SYMBOLIC TASKS

Large Language Models (LLM) such as BERT Wolf et al. (2020), RoBERTa Liu et al. (2019), GPT3 Brown et al. (2020) etc., pretrained on large unstructured knowledge from public data such as Wikipedia, Bookcorpus, Commoncrawl, etc, have shown impressive results in several NLP tasks. It shows the ability to generalize to multiple tasks from question answering and machine translation to story generation and instruction following Wang et al. (2018; 2019). LLMs have shown the ability to generate output in natural language Wolf et al. (2020); Raffel et al. (2020), adapt to novel tasks in a zero or few-shot approach Brown et al. (2020); Radford et al. (2019) and decode with constraints on output space Hokamp & Liu (2017); Welleck et al. (2019); Kumar et al. (2021). Recent progress in LLMs has demonstrated the generation of structured output that requires precise syntactic/symbolic knowledge with structural constraints such as knowledge graphs Petroni et al. (2019), protein structure Unsal et al. (2022); Ferruz & Höcker (2022), programming languages Ahmad et al. (2021), etc.

As the LLMs collect the related knowledge necessary to solve an NLP task, Petroni et al. (2019) have shown that the LLMs are potential representations of the large knowledge bases. In protein data Unsal et al. (2022); Ferruz & Höcker (2022), LLMs generate the functional properties of the proteins by enforcing the structural constraints specific to protein science and determining the complex functional relationships in the protein binding. Code generation has recently become very popular in the LLM research community. Several models such as CodeBERT Feng et al. (2020), Codex Chen et al. (2021), CodeT5 Wang et al. (2021) have shown significant improvement in transfer from pretrained models for natural language to structured codes. One of the key contributors to the success of these LLMs in code generation is fine-tuning of the models on task-specific data. For instance, CodeXGlue Lu et al. (2021), a benchmark dataset for code understanding and generation

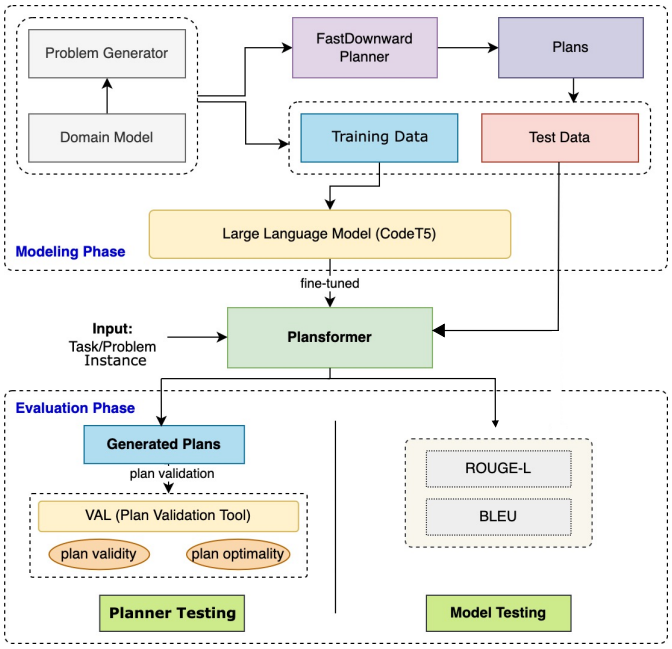

Figure 1: Plansformer Model Architecture showing modeling and evaluation phases. Modeling phase involves finetuning CodeT5 with data from planning domain. Evaluation phase shows both the planner and model testing.

with sample codes from several programming languages is used to finetune CodeBERT, CodeT5, etc. In this paper, we use the CodeT5 for further finetuning to the classical automated planning domain due to its ability to generate goal-directed, sequential instruction and semantically meaningful program codes with syntactic and structural constraints.

The most relevant works to this paper test the ability of LLM to generate symbolic plans Hernandez et al. (2021); Valmeekam et al. (2022); Huang et al. (2022). Hernandez et al. (2021) looks at different ways to generate plans using GPT-3 versions, a prominent generative LLM. Valmeekam et al. (2022) discusses different scenarios for generating plans like finding a satisficing plan or adapting a previous one and discuss encodings to generate plans and verify using a plan validator. Huang et al. (2022) generates step-by-step instructions for an user-defined task using LLM prompting. All these works require the (human-guided) mapping of a natural language-based sequence of instructions generated by the LLM to the admissible action in the planning domain as an additional step.

3 Plansformer for Symbolic Plans

Figure 1 provides an illustrative overview of how we have generated and tested our planner, called Plansformer. The main idea here is to employ an existing LLM and further train it on planning problem instances with corresponding valid plans. We then test if it can generate valid plans (or almost valid plans for unseen planning problem instances). However, the testing consists of two distinct stages: we first test if Plansformer could generate responses as given by the test dataset, and then we test if the generated plans are valid/optimal (independently of whether they were the same plans as in the test dataset). The first phase, which we call the *modeling* phase, takes care of creating Plansformer from the given LLM. The second phase, *evaluation*, deals with assessing the competency of Plansformer as a model and as a planner.

3.1 Modeling Phase

The modeling phase starts by creating a planning-based dataset, which is used for fine-tuning an LLM to generate Plansformer. The top of Figure 1 depicts the different modules employed.

Task	Problem Instance	Plan
blocksworld	<GOAL> on b1 b2, on b2 b3, ontable b3, on b4 b1, clear b4 <INIT> handempty, ontable b1, clear b1, on b2 b3, ontable b3, on b4 b2, clear b4 <ACTION> pick-up <PRE> clear x, ontable x, handempy <EFFECT> not ontable x, not clear x, not handempty, holding x <ACTION> put-down <PRE> holding x <EFFECT> not holding x, clear x, handempty, ontable x <ACTION> stack <PRE> holding x, clear y <EFFECT> not holding x, not clear y, clear x, handempty, on x y <ACTION> unstack <PRE> on x y, clear x, handempty <EFFECT> holding x, clear y, not clear x, not handempty, not on x y	unstack b4 b2, put-down b4, pick-up b1, stack b1 b2, pick-up b4, stack b4 b1

Figure 2: Snapshot of one instance of the plan dataset for blocksworld domain.

3.1.1 PLANNING DATASET

We create and release a PDDL-based dataset[2] which, besides being useful for creating Plansformer, can also act as a benchmark to facilitate further research at the intersection of LLMs and automated planning. We make use of the domain model (in PDDL) to automatically generate corresponding valid problem files with varying complexities. In this proof-of-concept study, we focus on four different classical planning domains, i.e., `Blocksworld, Towers of Hanoi, Grippers, Driverlog.`

Blocksworld, or **bw**, is a well studied domain Gupta & Nau (1991) consisting of blocks placed on a table or arranged in vertical stacks. Here, with the available actions of pick-up, put-down, stack, and unstack, one can alter the arrangement of the blocks. Problems here are generated with configurations of 2 to 5 blocks.

Towers of Hanoi, or **hn**, consists of 3 pegs and multiple disks of varying diameters. Initially, all disks are placed in the first peg, and the end goal is to move the disks to the last peg. The only limitation to consider when moving the disks is that only a smaller disk can be placed on top of a bigger disk. Although the domain has only one action, the problem solving is recursive Gerety & Cull (1986). Problems here are generated with configurations of 2 to 5 disks.

Grippers, or **gr** domain involves moving balls across rooms using robotic grippers. It has problems generated with configurations of 2 to 5 balls, 3 to 5 robots, and 2 to 4 rooms.

Driverlog or **dl** domain involves moving packages on trucks between locations driven by drivers. It has problems generated with configurations of 1 to 3 drivers, 1 to 3 trucks, 2 to 4 packages, and 3 to 6 locations.

With multiple problem instances associated with each domain model, corresponding plans for each problem instance are generated using FastDownward planner Helmert (2006). FastDownward is a classical planning system based on heuristic search, capable of generating optimal plans when used with A^* LM-Cut heuristic, thereby, prompting us to use it for generating a planning dataset consisting of optimal plans. An input format, which includes both the description of the planning domain and of the problem instance, is coupled with a corresponding valid plan to make a single instance in the dataset. The input format captures all the essential information present in the domain and problem instance, such as the goal, the initial state, and the possible actions that can be taken in that domain. A snapshot of the generated dataset is shown in Figure 2, with more examples in the Supplementary Material. This format considerably reduces the knowledge-engineering efforts required for a traditional planner to generate plans by eliminating the need for explicitly creating two different files - domain and problem. For each domain, the generated dataset consists of 18,000 plans generated for different problem configurations. For training, we use 5-fold cross validation with an 80%-20% split of the generated dataset for each domain. The average plan length for blocksworld is 9, gripper is 9, driverlog is 10, and hanoi is 12.

3.1.2 TOKENIZER

We use a Byte-level BPE tokenizer, following the standard practice in LLMs, with a vocabulary size of 32,005. We add PDDL-specific tokens, namely, `[GOAL], [INIT], [ACTION], [PRE],`

[2]The link is omitted during the review process.

[EFFECT] to represent the goal state, initial state, possible actions with their associated preconditions and effects those actions bring about in the environment respectively. We do not re-train a specific tokenizer for this task from scratch following the suite of Chen et al. (2021), where GPT-3's tokenizer was reused to generate code.

3.1.3 FINE-TUNING CODET5

With a plethora of LLMs to choose from, we shortlisted the models that have been pre-trained on code to exploit the code syntactic information implicitly captured in their weights. Although Codex Chen et al. (2021), which is built using GPT-3, has reported the best performance in solving code-related tasks, its unavailability to public access drove us to opt for a similar alternative: CodeT5 Wang et al. (2021). CodeT5 is a masked language model, consisting of an encoder-decoder stack inspired by the transformer architecture Vaswani et al. (2017), capable of performing a wide range of code intelligence applications including code understanding and generation tasks. More specifically, CodeT5 can be fine-tuned to perform 4 generation tasks (code summarization, code generation, translation, and refinement) plus 2 understanding tasks (code defect detection and clone detection) on 8 programming languages (Python, Java, JavaScript, PHP, Ruby, Go, C, and C#). Its pre-training tasks include identifier awareness and bimodal generation, which optimizes code-to-code understanding. The CodeT5 model possesses several properties amenable to the planning domain such as its ability to generate goal-directed, sequential instruction and semantically meaningful program codes with syntactic and structural constraints. With this pre-trained knowledge already embedded in CodeT5, we further fine-tuned it with 14400 samples for each independent domain from the planning dataset. As a result of this, the weights of CodeT5 are updated to account for the task of plan generation. The model thus obtained with updated weights capable of generating symbolic plans is termed Plansformer.

3.2 EVALUATION PHASE

Plansformer is an LLM that ingests a new problem instance as input and outputs a plan for that problem instance. Therefore, to evaluate its competency, we need to test both its quality as a model and as a planner. The evaluation phase is described in the lower part of Figure 1 showing both testing phases.

3.2.1 PLANNER TESTING

Symbolic plans have much richer information content, inherently captured in their structure as opposed to text. Thus, we have an additional evaluation phase of plan validation to check how well Plansformer can mimic an automated planner. The sequence of actions generated by Plansformer must help an agent to navigate from the initial state to the goal state modeled in the input problem, for the plan to be valid. If the generated plan is also the shortest possible plan, then it is said to be cost-optimal [3]. There are other possible metrics to evaluate a plan, but for this line of work, we consider *validity* and *optimality*. The plan generated from Plansformer is passed through a plan validation tool, called VAL Howey et al. (2004), to check for its optimality and validity. VAL is an automatic validation tool for PDDL. VAL takes as input the task posed to Plansformer as well as the corresponding generated plan and applies PDDL-based relaxation conditions to perform a check for validity and optimality.

3.2.2 MODEL TESTING.

LLMs dealing with tasks such as summarization or generation are usually evaluated using metrics such as BLEU and ROUGE, widely used metrics in NLP tasks. In general, BLEU measures precision and helps understand how closely a machine translation (here, plan generated by Plansformer) is compared to a human translation (here, plan generated by an automated planner). ROUGE measures recall, i.e., how many of the words referenced in human summaries appeared in the summaries generated by the machine. In particular, we adopt ROUGE-L, which considers sentence-level structure similarity by identifying the longest co-occurring sequence n-grams. Although ROUGE and BLEU have no direct intuition in Automated Planning, we use these metrics as we look at the task of plan

[3] We also refer it to as optimality interchangeably

Models	ROUGE-L_{recall}	ROUGE-$L_{precision}$	ROUGE-$L_{fmeasure}$	BLEU
Codex	0.72	0.52	0.60	0.36
GPT-2	0.04	0.14	0.06	0.07
T5-base	0.16	0.70	0.26	0.02
CodeT5-base	0.41	0.28	0.33	0.02
Plansformer	**0.93**	**0.93**	**0.93**	**0.89**
Plansformer-bw	0.97	0.99	0.98	0.90
Plansformer-hn	0.99	0.96	0.97	0.95
Plansformer-gr	0.94	0.94	0.94	0.92
Plansformer-dl	0.82	0.83	0.82	0.79

Table 1: Results of model testing (best performance in bold).

generation using an LLM. These metrics give us an overall idea of the performance of Plansformer as a language model but are not conclusive evidence for deciding how well Plansformer generates the plans. Hence, as described earlier, there is a need for testing Plansformer as a planner. Thus, the initial stage in our evaluation phase is model testing using the metrics mentioned above.

4 Experimental Results

In this section, we present the quantitative and qualitative results obtained using Plansformer to generate symbolic plans for multiple domains of varying complexities. A test-bed of 3600 unique and unseen problems for each domain is selected for testing Plansformer and all the results reported are averaged over 5 random train-test splits. For all the Plansformer variants, the reported results are obtained by testing on the same training domain, unless specified. For example, `Plansformer-bw`'s results are reported based on the performance obtained on `bw` test-bed. Evaluating Plansformer using our multi-testing strategy helps find its efficiency as a *language model* and a *planner*.

4.1 Is Plansformer a Good Model?

Table 1 presents the model evaluation metrics (ROUGE and BLEU) comparing all the Plansformer models with other LLMs. Actual generations from these various models for various domains are present in the Supplementary Material. In our selection of LLMs for the experiments, we have included best-performing models from masked language models (e.g., T5 Raffel et al. (2020)) and causal language models (e.g., GPT-2 Radford et al. (2019)). The major difference between these two types of LLMs is that masked language models are bidirectional in nature whereas causal language models are unidirectional. Plansformer has an encoder-decoder pair, where the encoder attends to tokens on either side of the masked word, whereas, the decoder auto-regressively generates plans.

The results reported are a mean of the model(s) performance on independent planning domains. For Plansformer, we show the performance on each separate domain as well. Plansformer performs best on all metrics, followed by Codex, having a considerable ROUGE-L_{recall} score. Codex's better performance compared to other baseline models can be accounted for by the natural language understanding capabilities inherited from GPT-3 and its ability to generate and understand code. It is also interesting to note that CodeT5, from which Plansformer is formed, largely underperforms, demonstrating the advantages of the proposed approach. We also conclude that the models pre-trained with code-related tasks have an advantage when it comes to the task of plan generation due to PDDL having some similarities with other programming languages. Despite having the best model testing metrics, Plansformer must be tested for plan validation to see its effectiveness as a planner.

4.2 Is Plansformer a Good Planner?

The results from the second stage of evaluation where the plan generations are checked for validity, optimality, and the average time taken to solve the test-bed of problems are reported in this section. Table 2 shows the plan validation scores obtained by different models. The considered ground truth for plan validation is FastDownward Helmert (2006), which is an automated planner. Given a correct input (i.e., a valid combination of domain description and problem instance), an automated planner always generates valid plans. FastDownward is a classical planning system based on a heuristic search. It can generate optimal plans when the landmark-cut heuristic is used within a standard A*

Models	**Valid Plans** (%)	**Invalid Plans**		**Optimal Plans (%)**	**Avg. Time (sec)**
		Incomplete/Wrong (%)	Failed (%)		
FastDownward (Ground Truth)	100%	-	-	100%	10.28s
Codex	0.15%	0%	99.85%	0.15%	1s
GPT-2	0%	100%	0%	0%	0.05s
T5-base	0.25%	82.7%	17.3%	0.25%	0.47s
CodeT5-base	0.6%	99.4%	0%	0.6%	0.68s
Plansformer	**83.64%**	16.18%	0.19%	**73.27%**	**0.06s**
Plansformer-bw	90.04%	9.94%	0.02%	88.44%	0.05s
Plansformer-hn	84.97%	14.72%	0.31%	82.58%	0.05s
Plansformer-gr	82.97%	16.61%	0.42%	69.47%	0.06s
Plansformer-dl	76.56%	23.44%	0%	52.61%	0.09s

Table 2: Results of plan validation.

search framework Helmert & Domshlak (2011). It can be seen that FastDownward generates 100% valid plans which are also optimal.

`Plansformer-bw` generated 90.04%, out of which 88.44% are optimal and have the highest performance. This better performance is analogous to the fact that `bw` is the easiest domain among the four domains (although finding optimal plans is hard, any problem has a satisficing solution linear in the number of blocks by putting them down and picking from the table Gupta & Nau (1991)).

On a relatively more complex domain, i.e., **dl**, `Plansformer-dl` attains 76.56% valid plans, out of which 52.61% are optimal. There is a 20% difference between valid and optimal plans for **dl** showcasing that the model was able to come up with completely new and valid action sequences, although not optimal. The number of optimal plans generated reduces with the increasing complexity of the domains. Both incomplete/wrong generations from the models and failed plans are included when reporting invalid plans. An incomplete/wrong generation is a partially correct ordering of action sequences but incomplete, whereas a failed plan is a complete generation but having impossible ordering of actions which isn't allowed by the domain definition. Figure 3 shows an example of incomplete generation and failed plans. All Plansformer models generate $\tilde{0}$% failed plans for respective domains. It is a testament to Plansformer models' ability to understand and generate valid action sequences for a given domain. The failed plans can be repaired Van Der Krogt & De Weerdt (2005) as they have a complete sequence of actions, out of which some are correct.

Failed Plans

Actual Plan: unstack b2 b4, put-down b2, unstack b4 b1, put-down b4, unstack b1 b3, put-down b1, pick-up b2, stack b2 b1, pick-up b4, stack b4 b2

Generated Plan: unstack b2 b4, put-down b2, unstack b4 b1, put-down b4, unstack b1 b3, put-down b4, unstack b4 b1, put-down b4, unstack b1 b3, put-down b4, unstack b4 b2, put-down b4, unstack b2 b4, stack b2 b1, pick-up b4, stack b4 b2

Incomplete Generations

Actual Plan: unstack b1 b3, put-down b1, pick-up b4, stack b4 b1

Generated Plan: unstack b1 b3, put-down b1, pick-up

Figure 3: Different types of invalid plans generated by Plansformer on Blocksworld domain (Plansformer-bw).

Codex, the second best performing model according to ROUGE and BLEU scores, only generates 0.15% valid plans, emphasizing the need for a two-stage evaluation phase - where both model and generated plans are tested. The average time taken by Plansformer to completely solve the test-bed of problems is $\tilde{2}$00x faster than the FastDownward planner, which generated ground truth plans, offering an immense advantage in generating approximately correct plans in real-time applications. Interestingly, CodeT5, which is used to build Plansformer, takes considerable time to solve the same test-bed. Despite this, Plansformer is faster as it generates valid and likely optimal plans shorter in length than usually long incoherent sequences generated by CodeT5, which tend to be time-consuming. The difference between plans generated by CodeT5 and Plansformer for the same problem is shown in Figure 4.

There isn't much prior work to adopt as a baseline and perform a comparison with Plansformer. Although, some recent works such as Huang et al. (2022), make use of LLMs to "plan", it is different from Automated Planning and is not symbolic in nature. These "plans" are step-by-step actions to perform a trivial daily task such as *"Apply lotion"*. These methods make use of a user-constructed prompt to enable an LLM to generate relevant steps from its prior knowledge. The work by Valmeekam et al. (2022) is the closest to ours, still making use of a PDDL-based prompt, to obtain symbolic plans with the help of GPT-3. The major difference between their dataset and ours is the difference in the object names. Their approach of converting the information captured in the PDDL-based domain and problem files to natural language and adding a solved prompt to the problem instance generated 0.6% valid plans, whereas Plansformer, on the same dataset generated 66% valid plans. This enables us to validate the advantage of our approach in generating symbolic plans. Plansformer, trained and tested on the same domain, shows superior performance both as a model and a planner. However, LLMs are also well known for transfer learning, i.e., a model trained for solving one task can be re-purposed to solve other related tasks. Subsequently, we explore how Plansformer trained on one domain can be adapted to another.

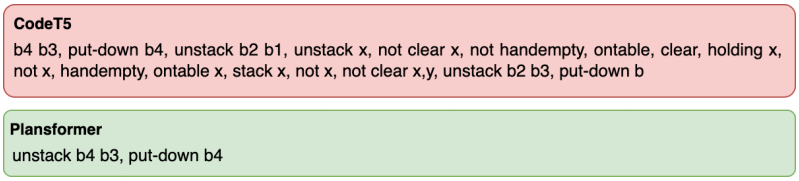

Figure 4: Long invalid generation from CodeT5 as opposed to a valid optimal plan from Plansformer-bw for the same problem.

4.3 CAN PLANSFORMER ADAPT TO ANOTHER DOMAIN?

The *base models*, i.e., Plansformer-x, where x can bw, hn, gr, and dl, cannot generate valid plans for other domains due to each domain being different from the other in terms of action space, complexity, and solving. For example, Plansformer-bw cannot generate valid plans for the non-training domains - hn, gr, and dl. However, LLMs offer two approaches to enable a model trained in one domain to adapt to another - (a) making use of prompt conditioning and (b) transfer learning using further fine-tuning on the new domain. The model's performance in a new domain in the prior approach depends on a manually identified prompt. A small perturbation to the prompt can significantly affect the model's performance, and creating a perfect prompt requires both understanding of LLMs' inner workings and trial-and-error. In recent times, researchers have started looking at automatic prompt generation Shin et al. (2020), which we would like to explore in the future.

We adopted the approach of fine-tuning Plansformer *base models* with problem instances from other domains to check for the ability to adapt to new domains. Figure 5 shows different Plansformer-bw models and their plan validation scores on respective testing domains. For brevity, we demonstrate variants of Plansformer-bw models on three other domains. Overall, the results for other Plansformer *base models* convey the same insights as Plansformer-bw and are presented in the Appendix.

For fine-tuning Plansformer-bw on a given domain, we have considered varying sizes of problems to see how the performance varies. The model naming convention conveys the information regarding the amount of problems it is fine-tuned on, i.e., `bw-hn[500]` implies that the *base model* used is `Plansformer-bw` further fine-tuned using 500 problems from **hn**. In Figure 5, we can see an overall increase in the number of valid plans for every testing domain as we increase the fine-tuning data points. Models fine-tuned with 2000, which is 14% of the training size of *base models*, achieves 50% of the valid plans recorded by `Plansformer-hn`, `Plansformer-gr`, and `Plansformer-dl`. On increasing the fine-tuning samples to that of the training size of *base models*, we obtain >90% valid plans for all testing domains, despite their complexity. `Plansformer-bw-hn[14400]` obtains the best performance among all models, by achieving 97.05% valid plans, out of which 95.22% are optimal.

Failed plans keep decreasing with additional fine-tuning data points for all domains. Using the same number of data points as that of training (14,400), the failed plans are lesser than that of the *base models* built for the respective domains. The number of optimal plans consistently increases with the number of fine-tuning data points for **hn** and is 13% more than `Plansformer-hn`, whereas for

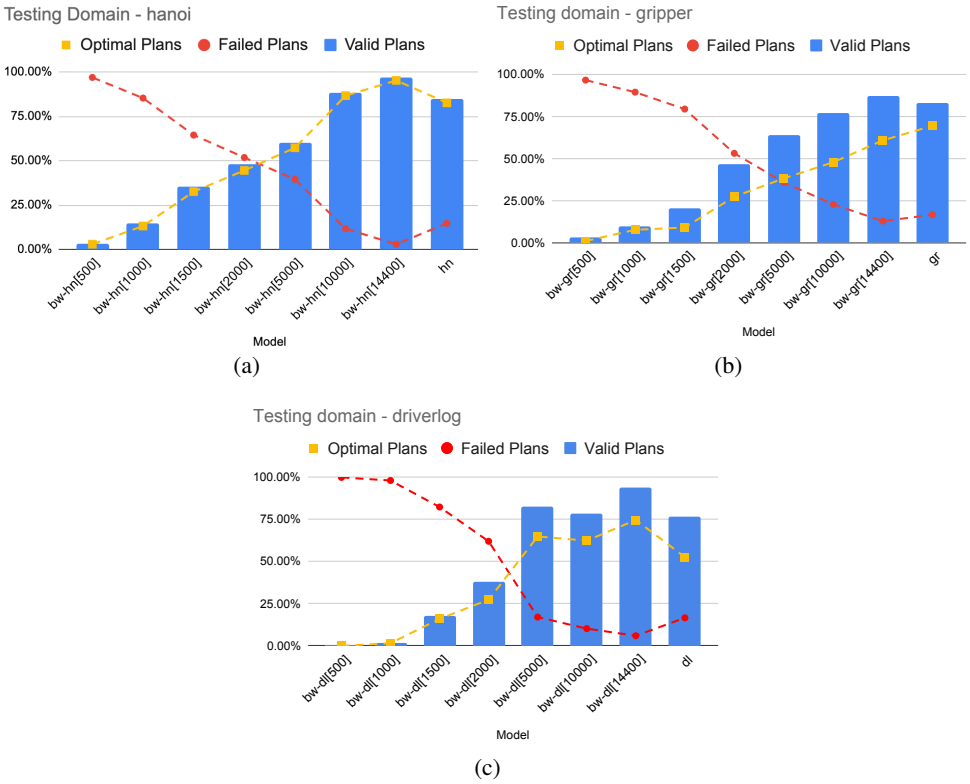

Figure 5: Plansformer-bw as the base model fine-tuned with and tested on (a) **hanoi** (b) **grippers** (c) **driverlog**

the other two domains we can see some variations. Fine-tuning results of all possible Plansformer models and their performance are reported in the Supplementary Material.

5 Conclusions and Ongoing Work

In this paper, we have explored the use of LLMs for generating symbolic plans for multiple domains. We have taken an LLM, trained it further over a set of planning problem instances and corresponding valid plans, and tested its capability to generate plans for unseen planning problem instances, evaluating the correctness and length of such plans. We then compared its behavior to an existing state-of-the-art planner, showing that our LLM-based planner, called Plansformer, can solve most instances with high quality both in terms of correctness and length while needing *much less time* to generate such plans. Although Plansformer shows promising results, this work is not done and can be extended in many directions. One can seek to eliminate invalid plans comprising both failed or incomplete plans, drive efficiency further, explore more planning domains, e.g., from competitions ICAPS (2022), and tackle other planning settings Valmeekam et al. (2022).

Beyond serving as a plan generator, Plansformer can also be used as a building block of general and adaptive planners. We are using Plansformer in combination with an existing symbolic planner in the context of a cognitive architecture inspired by the thinking fast and slow theory Kahneman (2011). In this environment, Plansformer provides complementary capabilities to the classical planner and will be used when resources (time, space, and knowledge) are limited and a small degrade on plan validity can be acceptable. A meta-cognition module decides when to use the Plansformer or the classical planner, and the system stores its output as experiences that build over time. Moreover, the plans generated by the classical planner are used as additional training data for Plansformer. Therefore, Plansformer combined with a symbolic planner and a meta-cognition module constitutes a "learning planner" that over time can exploit more experience. Preliminary results indicate that this approach is promising in terms of the number of solved problem instances attainable within a time limit, plans validity, and plan generation time.

REFERENCES

Wasi Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. Unified pre-training for program understanding and generation. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 2655–2668, 2021.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. Codebert: A pre-trained model for programming and natural languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pp. 1536–1547, 2020.

Noelia Ferruz and Birte Höcker. Controllable protein design with language models. *Nature Machine Intelligence*, pp. 1–12, 2022.

M. Fox and D. Long. Pddl2.1: An extension to pddl for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20:61–124, Dec 2003. ISSN 1076-9757. doi: 10.1613/jair.1129. URL http://dx.doi.org/10.1613/jair.1129.

C Gerety and P Cull. Time complexity of the towers of hanoi problem. *SIGACT News*, 18(1):80–87, mar 1986. ISSN 0163-5700. doi: 10.1145/8312.8320. URL https://doi.org/10.1145/8312.8320.

Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning: Theory and Practice*. The Morgan Kaufmann Series in Artificial Intelligence. Morgan Kaufmann, Amsterdam, 2004. ISBN 978-1-55860-856-6. URL http://www.sciencedirect.com/science/book/9781558608566.

Naresh Gupta and Dana S. Nau. Complexity results for blocks-world planning. In *In Proceedings of AAAI-91*, pp. 629–633, 1991.

Malte Helmert. The fast downward planning system. *Journal of Artificial Intelligence Research*, 26: 191–246, 2006.

Malte Helmert and Carmel Domshlak. Lm-cut: Optimal planning with the landmark-cut heuristic. *Seventh international planning competition (IPC 2011), deterministic part*, pp. 103–105, 2011.

Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, et al. Measuring coding challenge competence with apps. *arXiv preprint arXiv:2105.09938*, 2021a.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021b.

Alberto Olmo Hernandez, Sarath Sreedharan, and Subbarao Kambhampati. Gpt3-to-plan: Extracting plans from text using GPT-3. *CoRR*, abs/2106.07131, 2021. URL https://arxiv.org/abs/2106.07131.

Chris Hokamp and Qun Liu. Lexically constrained decoding for sequence generation using grid beam search. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1535–1546, 2017.

Richard Howey, Derek Long, and Maria Fox. Val: Automatic plan validation, continuous effects and mixed initiative planning using pddl. In *16th IEEE International Conference on Tools with Artificial Intelligence*, pp. 294–301. IEEE, 2004.

Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. *arXiv preprint arXiv:2201.07207*, 2022.

ICAPS. International planning competitions at international conference on automated planning and scheduling (icaps). In *https://www.icaps-conference.org/competitions/*, 2022.

Daniel Kahneman. *Thinking, fast and slow*. Macmillan, 2011.

Sachin Kumar, Eric Malmi, Aliaksei Severyn, and Yulia Tsvetkov. Controlled text generation as continuous optimization with multiple constraints. *Advances in Neural Information Processing Systems*, 34:14542–14554, 2021.

Hang Li. Language models: Past, present, and future. *Commun. ACM*, 65(7):56–63, jun 2022. ISSN 0001-0782. doi: 10.1145/3490443. URL https://doi.org/10.1145/3490443.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin Clement, Dawn Drain, Daxin Jiang, Duyu Tang, et al. Codexglue: A machine learning benchmark dataset for code understanding and generation. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021.

Drew McDermott, 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. Technical report, Technical Report, 1998.

Fabio Petroni, Tim Rocktäschel, Sebastian Riedel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, and Alexander Miller. Language models as knowledge bases? In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 2463–2473, 2019.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

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

Taylor Shin, Yasaman Razeghi, Robert L. Logan IV, Eric Wallace, and Sameer Singh. AutoPrompt: Eliciting knowledge from language models with automatically generated prompts. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2020.

Serbulent Unsal, Heval Atas, Muammer Albayrak, Kemal Turhan, Aybar C Acar, and Tunca Doğan. Learning functional properties of proteins with language models. *Nature Machine Intelligence*, 4 (3):227–245, 2022.

Karthik Valmeekam, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. Large language models still can't plan (a benchmark for llms on planning and reasoning about change). *arXiv preprint arXiv:2206.10498*, 2022.

Roman Van Der Krogt and Mathijs De Weerdt. Plan repair as an extension of planning. In *ICAPS*, volume 5, pp. 161–170, 2005.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pp. 353–355, 2018.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. Superglue: A stickier benchmark for general-purpose language understanding systems. *Advances in neural information processing systems*, 32, 2019.

Yue Wang, Weishi Wang, Shafiq Joty, and Steven CH Hoi. Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 8696–8708, 2021.

Sean Welleck, Kianté Brantley, Hal Daumé Iii, and Kyunghyun Cho. Non-monotonic sequential text generation. In *International Conference on Machine Learning*, pp. 6716–6726. PMLR, 2019.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pp. 38–45, 2020.