Plansformer: Generating Symbolic Plans using Transformers

Vishal Pallagani
AIISC, University of South Carolina
vishalp@mailbox.sc.edu

Bharath Muppasani
AIISC, University of South Carolina
bharath@email.sc.edu

Keerthiram Murugesan
IBM T.J. Watson Research Center
keerthiram.murugesan@ibm.com

Francesca Rossi
IBM T.J. Watson Research Center
francesca.rossi2@ibm.com

Biplav Srivastava
AIISC, University of South Carolina
biplav.s@sc.edu

Lior Horesh
IBM T.J. Watson Research Center
lhoresh@us.ibm.com

Francesco Fabiano
University of Udine
francesco.fabiano@unipr.it

Andrea Loreggia
University of Brescia
andrea.loreggia@gmail.com

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 focus 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 executed by intelligent agents, autonomous robots, and unmanned vehicles. We introduce Plansformer, an LLM fine-tuned on planning problems and capable of generating plans with favorable behavior in terms of correctness and length with reduced 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 transformer-based (neural) architecture [Vaswani et al., 2017, Devlin et al., 2018, Brown et al., 2020, Scao et al., 2022, Chowdhery et al., 2022], have significantly 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 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

NeurIPS 2023 Workshop on Generalization in Planning (GenPlan 2023).

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 has been limited in the domains that involve symbols. For example, domains 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] suggest that even state-of-the-art LLMs cannot reason with symbolic data and offer a new suite of benchmarks to test their reasoning capabilities. Recently, there has been a lot of interest in LLMs for code generation; for example, CodeT5 [Wang et al., 2021], CodeBERT [Feng et al., 2020], and Codex [Chen et al., 2021]. In this paper, we propose to employ LLMs that are trained to generate code and repurpose them to generate valid plans for automated planning domains.

To advance the research in LLM-based automated planning, we create a training and test set for four planning domains. We use CodeT5 (base), a transformer-based code generation model that achieves state-of-the-art results in CodeXGlue Lu et al. [2021], as the pre-trained LLM. We select CodeT5 due to its ability to generate goal-directed, sequential instructions and semantically meaningful program codes with syntactic and structural constraints [Pallagani et al., 2023]. Then, we present, Plansformer, an LLM trained to generate symbolic plans of high quality in terms of correctness and length. Our experimental results indicate that the syntactic/symbolic knowledge learned from different programming languages in the CodeT5 model can be beneficial for the PDDL-based automated planning task. For example, in the puzzle-solving domain of Towers of Hanoi, our model was able to generate 97% valid plans, out of which 95% are shortest length plans. The results reveal a promising direction to harness LLMs for symbolic tasks such as planning.

Plansformer is not intended to replace an existing automated planner, which is traditionally used to solve planning problems. A traditional planner is capable of searching through an entire state space and certainly generating a valid (or optimal) plan(s). However, a planner often consumes a considerable time to navigate the search space for larger problems or complex domains and also does not possess language semantics. Thus, one of the use cases for Plansformer currently is using it in a neuro-symbolic cognitive architecture as inspired by the Fast and Slow thinking principles as proposed by Daniel [2017]. A Plansformer can play to its benefit as a System 1 solver, which has relaxation in terms of correctness but is supposed to be fast and the traditional planner can be used as a System 2 solver, which is deliberative and can reason with no time constraints in order to always generate a correct output. This work is also a promising exploration in understanding LLMs capabilities in dealing with symbolic language. Valmeekam et al. [2022] show how a pre-trained model, GPT-3 [Brown et al., 2020], is unable to generate valid plans for *blocksworld* domain using prompt conditioning, but our experiments show that a pre-trained model on coding languages with further fine-tuning, can produce valid plans.

In the remainder of the paper, we present preliminaries on automated planning and language models and then propose an LLM repurposed planner called *Plansformer*. Next, we present the experimental results comparing our approach with state-of-the-art planners and other large language models. Furthermore, we demonstrate the ability of Plansformer to adapt to other domains and discuss the relevance to instruction generation. We conclude with a discussion of the results and presentation of ongoing work.

2 Background

2.1 Automated Planning

Given the initial and goal states, alongside a set of legal actions, the objective of a planning agent is to devise a sequence of actions that advance the agent from the initial to the goal state. This paper adopts the Planning Domain Description Language (PDDL) [McDermott et al., 1998, Fox and 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 modify 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 devises plans by searching in the space of states, where a state is a configuration of physical objects or partial plans. There is a single agent in the most basic formulation, called classical planning. 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, and the cost and duration of actions can be non-uniform. At the same time, 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.

2.2 Large Language Models and Symbolic Tasks

LLMs such as BERT [Wolf et al., 2020], RoBERTa [Liu et al., 2019], and GPT3 [Brown et al., 2020] are pre-trained on extensive unstructured knowledge from public data such as Wikipedia, Bookcorpus, and Commoncrawl. They have shown impressive results in several NLP tasks. It demonstrated 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 and 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 and Höcker, 2022], and programming languages [Ahmad et al., 2021]. 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 significant knowledge bases. In protein data [Unsal et al., 2022, Ferruz and 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], and CodeT5 [Wang et al., 2021] have shown significant improvement in transfer from pre-trained models for natural language to structured codes. One of the key contributors to the success of these LLMs in code generation is fine-tuning the models on task-specific data. For instance, CodeXGlue, a benchmark dataset for code understanding and generation with sample codes from several programming languages, is used to fine-tune CodeBERT, CodeT5, and others. In this paper, we harness CodeT5 for further fine-tuning to the classical automated planning domain due to its ability to generate goal-directed, sequential instructions and semantically meaningful program codes with syntactic and structural constraints.

The closest prior art addressing the ability of LLMs to generate symbolic plans are the studies of Hernandez et al. [2021], Valmeekam et al. [2022], Liu et al. [2023], Silver et al. [2023] and 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 discusses encodings to generate plans and verify using a plan validator. Huang et al. [2022] generates step-by-step instructions for a user-defined task using LLM prompting. All these studies 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 generate and test our planner, called Plansformer. The first phase, *modeling*, shows how we fine-tune the CodeT5 to address planning syntax and semantics. The second phase, *evaluation*, deals with assessing the competency of Plansformer as a model and as a planner. The key idea here is to utilize an LLM (CodeT5) pretrained on code generation and further train it on planning problem instances with corresponding valid plans. We evaluate its competence in generating valid plans (or almost valid plans for unseen planning problem instances) using two types of testing: 1) Model testing measures if Plansformer could generate meaningful responses (as in the test dataset), 2) Planner testing measures if the generated plans are valid/optimal (independently of whether they were the same plans as in the test dataset).

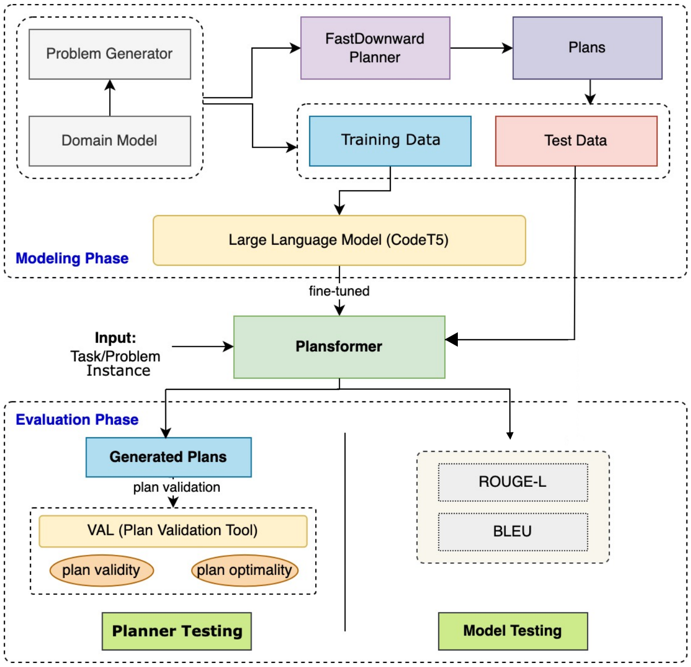


Figure 1: Plansformer Model Architecture showing modeling and evaluation phases. Modeling phase involves finetuning CodeT5 with data from planning domain. Evaluation phase shows both the planner and model testing.

3.1 Modeling Phase

In the modeling phase, we first create a planning-based dataset for finetuning the CodeT5 to generate plans. The modeling phase of Figure 1 depicts the different modules employed.

3.1.1 Planning dataset

We generate a PDDL-based dataset as a benchmark to finetune pretrained CodeT5 and facilitate further research at the intersection of LLMs and automated planning. We use the domain model (in PDDL) to generate corresponding valid problem files with varying complexities automatically. In this paper, we focus on four different classical planning domains, i.e., *Blocksworld, Towers of Hanoi, Grippers, Driverlog.*

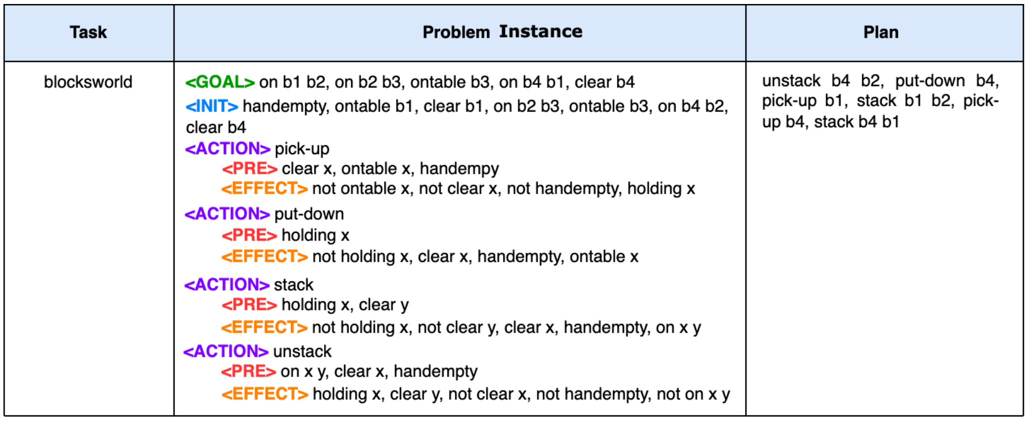
Blocksworld, or **bw**, is a well-studied domain [Gupta and Nau, 1991] with blocks placed on a table or arranged in vertical stacks. Here, one can alter the arrangement of the blocks with the available actions such as pick-up, put-down, stack, and unstack. We generate the problems with 2 to 5 block configurations.

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 and Cull, 1986]. Here, we generate the problems 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.

Each planning domain explained above includes multiple problem instances. We generate the corresponding plans for each problem instance using FastDownward planner [Helmert, 2006]. FastDownward is a classical planning system based on a heuristic search and offers different search algorithms such as causal graph heuristics and A* search. FastDownward can generate optimal plans with A* LM-Cut heuristic [Helmert and Domshlak, 2011]. Hence, FastDownward can be regarded as a potent planner for generating a dataset of optimal plans. We show the snapshot of the generated dataset in Figure 2, with more examples in Section 1 of supplementary material. Unlike the traditional planner which requires two different files - domain and problem (as pddl files, See Figure 2 in supplementary material), Plansformer reduces the knowledge-engineering efforts with a simplified input format that includes the problem instance coupled with a corresponding valid plan. The problem instance captures all the essential information in the domain and problem instance, such as the goal, the initial state, and the possible actions that can be taken in that domain. The generated dataset for each domain consists of 18,000 plans with 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 (number of actions in the generated plan) for blocksworld is 9, gripper is 9, driverlog is 10, and hanoi is 12.



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, handempty <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.

3.1.2 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]`, `[EFFECT]` to represent the goal state, initial state, possible actions with their associated preconditions and effects these actions cause in the environment respectively. We do not re-train a specific tokenizer for this task from scratch following the previous work [Chen et al., 2021], where GPT-3's tokenizer was reused to generate code.

3.1.3 Fine-tuning CodeT5

While there are many LLMs to select as a candidate for this work, we shortlist the models pre-trained on code generation to exploit the syntactic information in the programming languages implicitly captured in their weights. Although Codex [Chen et al., 2021], built using GPT-3, has reported the best performance in solving code-related tasks, its lack of public access led us to choose an equally competitive LLM: 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]. It is capable of performing a wide range of tasks including code generation and understanding tasks. The generation tasks include code summarization, code generation, translation, and refinement. The understanding tasks include code defect detection and clone detection. CodeT5 is pretrained with example codes from eight 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 encoded within CodeT5, we finetune it with 14400 samples (80% of the generated dataset) for each independent domain from the planning dataset. As a result of this finetuning, the weights of CodeT5 are updated to account for the task of plan generation. We give the planning problem instance as input to CodeT5's encoder and generate the intermediate features for the decoder of CodeT5 to output a plan.

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 must test its quality as a model and planner. The evaluation phase is described in the lower part of Figure 1, showing both testing phases.

3.2.1 Planner Testing

Unlike in natural language, symbolic plans have richer information content, inherently captured in their structure. Thus, we have an additional evaluation phase for 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 for a given problem instance. We call a generated plan *cost-optimal* [1] if it is the shortest possible among all other plans. Several metrics exist in the automated planning literature to evaluate a plan generated by Plansformer. In this paper, we consider *validity* and *optimality*. We evaluate the plan generated by Plansformer using 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 and the corresponding generated plan. It applies PDDL-based relaxation conditions to check for validity and optimality.

3.2.2 Model Testing

It is typical to evaluate natural language tasks such as summarization or generation using metrics such as BLEU and ROUGE. Both BLEU and ROUGE are widely used metrics in NLP. 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). On the other hand, 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 to look at the task of plan generation from the perspective of LLMs. The evaluation based on these metrics provides us with an insight into the performance of Plansformer as a language model. In the next section, we evaluate Plansformer as a planner to give conclusive evidence on how well Plansformer generates the plans.

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. We select a test-bed of $3,600$ unique and unseen problem instances (20% of the dataset) for each domain for evaluating Plansformer. All the results reported in this paper are averaged over 5 randomly selected ($80\% - 20\%$) train-test splits. We report the results for the Plansformer variants by evaluating the corresponding test-bed. For example, `Plansformer-bw`'s results are reported based on the performance results obtained on **bw** test-bed. We evaluate Plansformer using both model and planner testing to find its efficiency as a *language model* and a *planner*.

4.1 Is Plansformer a Good Model?

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. Table 1 compares all the Plansformer models with other LLMs using the model evaluation metrics (ROUGE and BLEU). In this experiment, we consider the best-performing models from (bidirectional) masked language models (e.g., T5 [Raffel et al., 2020]) and (unidirectional) causal language models (e.g., GPT-2

[1] We also refer to it as optimality interchangeably

Models	**ROUGE-L$_{\textbf{recall}}$**	**ROUGE-L$_{\textbf{precision}}$**	**ROUGE-L$_{\textbf{fmeasure}}$**	**BLEU**
GPT-2	0.04	0.14	0.06	0.07
T5-base	0.16	0.70	0.26	0.02
Codex	0.72	0.52	0.60	0.36
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).

Models	**Valid Plans** (%)	**Invalid Plans**		**Optimal Plans** (%)	**Avg. Time (sec)**
		Failed (%)	Incomplete/Wrong (%)		
FastDownward (Ground Truth)	100%	-	-	100%	10.28s
GPT-2	0%	0%	100%	0%	0.05s
T5-base	0.25%	17.3%	82.7%	0.25%	0.47s
Codex	0.15%	99.85%	0%	0.15%	1s
CodeT5-base	0.6%	0%	99.4%	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.

[Radford et al., 2019]). We present the actual plan generations from a few of these models in Figure 5 of supplementary material.

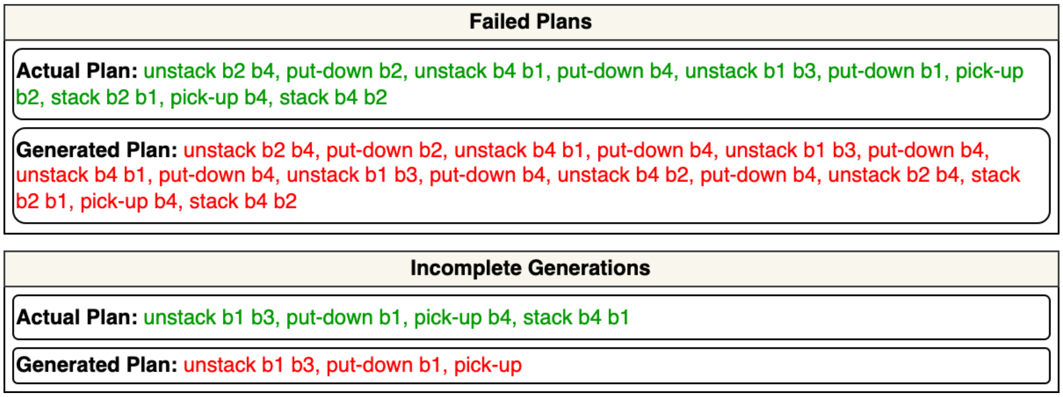
We report the performance of the baseline models averaged over the four planning domains. We also show the performance of Plansformer on individual domains (Plansformer-bw, Plansformer-hn, Plansformer-gr and Plansformer-dl). We observe that Plansformer performs best on all metrics, followed by Codex, with a significant ROUGE-L$_{\text{recall}}$ score. We believe that the performance gain from Codex compared to other baseline models is due to it's ability to relate the natural language understanding (a skill inherited from GPT-3) with code generation. It is interesting to see that CodeT5 performs poorly compared to Codex and Plansformer, demonstrating the advantages of the natural language understanding with code generation task on this evalutation metrics. We conclude that the models pre-trained with code-related tasks have an advantage over other models in plan generation task due to the similarities of PDDL with other programming languages. Despite with the best model testing metrics, We need to test Plansformer for plan validation to see its effectiveness as a planner.

4.2 Is Plansformer a Good Planner?

In this section, we report the results from the planner testing. We evaluate the generated plans for validity and optimality. We also report the average time taken to solve the problem instances. Table 2 shows the plan validation scores obtained by different models. We consider FastDownward [Helmert, 2006] to generate the ground truth plans. FastDownward planner generates a 100% valid and optimal plan for a given input (i.e., a combination of domain description and problem instance) when the landmark-cut heuristic is used within a standard A* search framework [Helmert and Domshlak, 2011].

We can see that Blockworld **bw** domain achieved the highest performance gain via `Plansformer-bw` - generated 90.04%, out of which 88.44% are optimal. This better performance is analogous to the fact that **bw** is the easiest domain among the four domains. Although it is hard to find optimal plans, we can find a valid plan linear in the number of blocks to any problem instances by putting them down and picking from the table [Gupta and Nau, 1991].

On a relatively more complex domain, i.e., **dl**, `Plansformer-dl` achieves 76.56% valid plans, out of which 52.61% are optimal. We notice a ~20% difference between valid and optimal plans for **dl**, with an observation that the model can come up with completely new and valid action sequences, although may not be optimal. We can see that the number of optimal plans generated reduces with the increasing complexity of the domains. We include both incomplete/wrong generations from the models and failed plans when reporting invalid plans. An incomplete/wrong generation is a partially correct ordering of action sequences but with some truncated tokens, whereas a failed plan consists of an incorrect ordering of actions, not leading to a goal state. Figure 3 shows an example of incomplete generation and failed plans. All Plansformer models generate close to 0% incomplete/wrong plans for respective domains. Failed plans can be repaired [Van Der Krogt and De Weerdt, 2005] as they have a complete sequence of actions, some of which are correctly ordered.



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. We notice that the average time taken by Plansformer to completely solve the test-bed of problems is ∼200x faster than the FastDownward, an automated planner that generated ground truth plans. Plansformer may offer an immense advantage in generating approximately correct plans in real-time applications. Interestingly, CodeT5, used to build Plansformer, takes considerable time to solve the same problem instances from the test bed. We believe that the Plansformer is faster since it generates valid and likely optimal plans shorter in length than usually long incoherent sequences generated by CodeT5, which tend to be time-consuming.

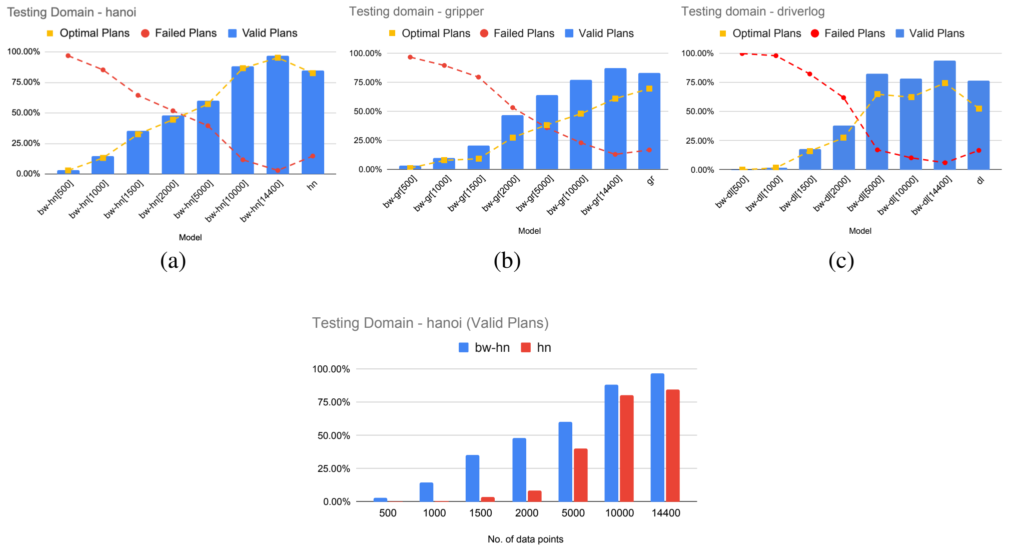
There have been very few relevant works that can be compared with Plansformer. Although some recent works such as [Huang et al., 2022] use LLMs to generate “plans", it is different from automated planning and is not symbolic in nature. These “plans" are step-by-step actions to perform a trivial everyday task such as *“Brush teeth"*. These methods use a user-constructed prompt to enable an LLM to generate appropriate steps from its prior knowledge. We believe the work by [Valmeekam et al., 2022] is similar in spirit to ours, using a PDDL-based natural language prompt to obtain symbolic plans using GPT-3. The significant difference between their dataset and ours is the difference in the object names, for example, a block is named as *b1* in our dataset as opposed to *a* in [Valmeekam et al., 2022]. This difference lets us evaluate Plansformer[2] on the dataset from [Valmeekam et al., 2022] with different object names as opposed to our dataset introduced in Section 3.1.1. On this dataset, Plansformer generated 66% *valid plans*, whereas, GPT3 with PDDL-based natural language prompting generated only 0.6% valid plans. The significant difference in performance enables us to validate the advantage of our approach in generating valid plans despite different object names.

Plansformer, trained and tested on the same domain, displays 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 domain can be re-purposed to solve other related domains. In the next section, we explore how Plansformer trained on one domain can be adapted to another.

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 (i.e., `Plansformer-bw` on **hn, gr, and dl**) since each domain differs from the

[2] We would like to note that Plansformer is trained only on the dataset introduced in Section 3.1.1 and is not trained/finetuned on the dataset in [Valmeekam et al., 2022] for this experiment



(a) (b) (c)

(d)

Figure 4: Plansformer-bw as the base model fine-tuned with and tested on (a) **hanoi** (b) **grippers** (c) **driverlog**, and (d) shows the comparison of valid plans generated by Plansformer-bw-hn derived models with Plansformer-hn trained using similar data points.

others in terms of action space, complexity, and solving. However, LLMs allow us to utilize the model trained in one domain to adapt to another using either making use of prompt conditioning or transfer learning with further fine-tuning on the problem instances from the new domain. We have seen from the previous work on prompt conditioning [Valmeekam et al., 2022] that the performance of the model on an unseen domain is very sensitive to the manually-identified prompt. A small perturbation to the prompt can significantly affect the model's performance, and creating a perfect prompt requires understanding the inner workings of LLM on hand and trial and error. In recent years, researchers have started looking at automatic prompt generation [Shin et al., 2020], which we would like to explore in the future.

Instead of the prompt conditioning, we follow the transfer learning approach by finetuning Plansformer *base models* with problem instances from other domains to check the ability of Plansformer to adapt to new domains. For brevity, we demonstrate variants of `Plansformer-bw` models on three other domains. Figure 4 shows different `Plansformer-bw` models and their plan validation scores on respective test-bed from target domains. We report that the results for transfer learning setup of Plansformer *base models* convey the same insights as `Plansformer-bw` shown here and are presented in Section 4.2 of supplementary material .

We consider different numbers of problem instances for finetuning `Plansformer-bw` on a given domain to see how the performance of the model varies across the sample size. We use the model naming format to convey the details on the amount of problem instances used for finetuning the Plansformer base model, i.e., `bw-hn[500]` implies that we further finetune `Plansformer-bw` using 500 problem instances from **hn** and report the results. In Figure 4, we can see an overall increase in the number of valid plans for every testing domain as we increase the problem instances available for finetuning. We observe that the models fine-tuned with 2000, which is 14% of the training size of *base models*, achieves $\sim 50\%$ of the valid plans recorded by `Plansformer-hn`, `Plansformer-gr`, and `Plansformer-dl`. Despite the complexity of these planning domains, we obtain $> 90\%$ valid plans for all testing domains by increasing the finetuning samples to that of the training size of *base models*. `Plansformer-bw-hn[14400]` obtains the best performance among all models, by achieving 97.05% valid plans, out of which 95.22% are optimal. In Figure 4(d), we compare Plansformer-hn trained with different number of data points from **hn** domain against the Plansformer-bw (base model) finetuned on the same data points. We can see a clear advantage of the transfer learning capability in LLMs, as both the **bw** and **hn** domains have similar plan semantics. Similar trends can be seen for the other domains and the results are reported in Section 4.2 of supplementary material.

We notice that the failed plans decrease with additional problem instances used for finetuning. Using the same amount of problem instances as training $(14,400)$, we observe that the number of failed plans is less than that of the *base models* built for the respective domains. The number of optimal plans consistently increases with the number of problem instances in **hn** domain for finetuning. It is 13% more than `Plansformer-hn`, whereas we can see some variations for the other two domains. We also report the fine-tuning results of all possible Plansformer models and their performance in Figure 6 of the supplementary material. We have also trained a single Plansformer model on all the domains (multi-task setting) and found that the individual model has relatively comparable performance to that of the *base models* as shown in Table 2 (See supplementary material Section 4.2 for more details). It is to be noted that Plansfomer's performance is affected by randomizing object names present in the prompt. We see a decrease in the performance with addition of more alphabets in the object names and has an equivalent performance as reported in Table 2 when the input prompt has object names belonging to the same vocabulary as that of the training set. However, we found that remapping object names in the input prompt to that of the training set lead to a better performance. Finding the right nomenclature for objects is beyond the scope of this paper. Section 4.2.4 in the supplementary section discusses object name randomization in detail.

5 Conclusions and Ongoing Work

In this paper, we have explored using LLMs to generate symbolic plans for multiple domains. We have taken an LLM tailored to code and trained it further over a set of planning problem instances and corresponding valid plans. We then tested the model's capability to generate plans for unseen planning problem instances, evaluating the correctness and length of such plans. Our approach is compared 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. Beyond serving as a plan generator, Plansformer can also be used as a building block for 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 minor degradation in 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.

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*, pages 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.

Kahneman Daniel. *Thinking, fast and slow*. 2017.

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*, pages 1536–1547, 2020.

Noelia Ferruz and Birte Höcker. Controllable protein design with language models. *Nature Machine Intelligence*, pages 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*, pages 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*, pages 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)*, pages 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*, pages 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. In *International Conference on Machine Learning*, pages 9118–9147. PMLR, 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`.

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. *arXiv preprint arXiv:2304.11477*, 2023.

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.

Vishal Pallagani and Biplav Srivastava. A generic dialog agent for information retrieval based on automated planning within a reinforcement learning platform. *Bridging the Gap Between AI Planning and Reinforcement Learning (PRL)*, 2021.

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. *arXiv preprint arXiv:2305.16151*, 2023.

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)*, pages 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.

Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilić, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, Matthias Gallé, et al. Bloom: A 176b-parameter open-access multilingual language model. *arXiv preprint arXiv:2211.05100*, 2022.

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.

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. *arXiv preprint arXiv:2305.11014*, 2023.

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, pages 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*, pages 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*, pages 8696–8708, 2021.

Sean Welleck, Kianté Brantley, Hal Daumé Iii, and Kyunghyun Cho. Non-monotonic sequential text generation. In *International Conference on Machine Learning*, pages 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*, pages 38–45, 2020.

Lili Yao, Nanyun Peng, Ralph Weischedel, Kevin Knight, Dongyan Zhao, and Rui Yan. Plan-and-write: Towards better automatic storytelling. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 7378–7385, 2019.

Supplementary Material

Contents

A Frequently Asked Questions

A.1 What are the main contributions of this paper?

Our main contributions in this paper are as follows:

- We introduce Plansformer; an LLM pre-trained on code and fine-tuned on planning problems is capable of generating plans with favorable behavior in terms of correctness and length with minimal knowledge-engineering efforts for four different planning domains (Hanoi, Gripper, Driverlog, Blocksworld). For one configuration of Plansformer - Towers of Hanoi - a puzzle-solving domain, we achieved 97% valid plans, out of which 95% are optimal. This is in contrast to negative results with GPT-3 reported in literature where they found 0.6% valid plans in blocksworld; we were able to generate 66% valid plans in the best case.
- We also demonstrate the generalization ability of Plansformer in solving different planning domains (Hanoi, Gripper, Driverlog, Blocksworld) with varying complexities, owing to the transfer learning abilities of LLMs.

Additionally, we also show that Plansformer's performance is affected by randomizing the object names. Although, we suggest a way to overcome this in Section 4.2.4 of the supplementary material, our *main focus in this paper is to show the empirical demonstration of how transformers can be repurposed to plan, and the methods that work better.*

A.2 Have you compared Plansformer with ChatGPT's capability in generating plans?

Figure 6 shows ChatGPT's response to a simple problem from blocksworld as input. In Figure 5, we can see the initial state and required goal state as mentioned in the input. Plansformer for the same input gives an optimal plan. It is worthwhile to mention that our claim here is not to test any specific LLM in its ability to generate plans, but rather, with proper adaptation an LLM can be repurposed to effectively solve planning problems. We are not specifically tied to any specific LLM, and that is one major advantage of the framework (though code-based LLMs may be more appropriate than others).

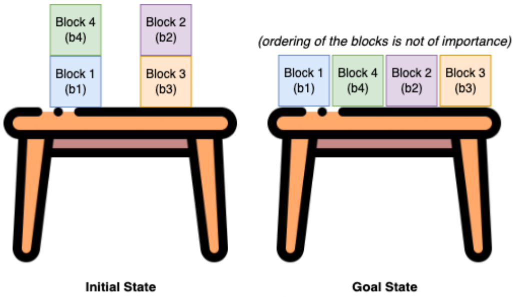
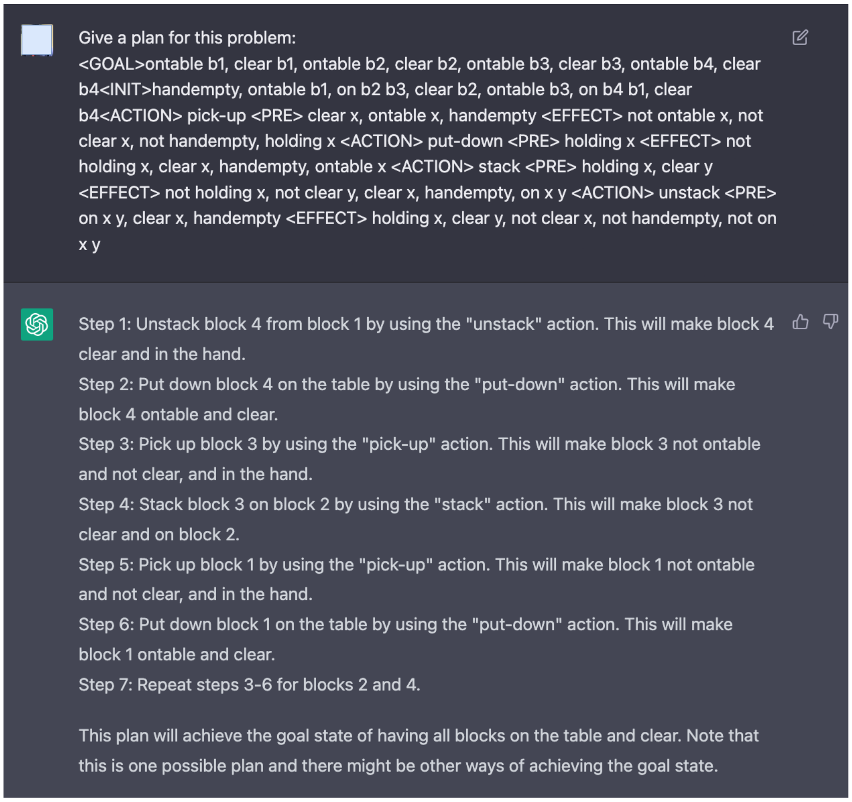


Figure 5: Visualization of blocksworld problem instance posed to ChatGPT



Give a plan for this problem:
<GOAL>ontable b1, clear b1, ontable b2, clear b2, ontable b3, clear b3, ontable b4, clear b4<INIT>handempty, ontable b1, on b2 b3, clear b2, ontable b3, on b4 b1, clear b4<ACTION> pick-up <PRE> clear x, ontable x, handempty <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

Step 1: Unstack block 4 from block 1 by using the "unstack" action. This will make block 4 clear and in the hand.
Step 2: Put down block 4 on the table by using the "put-down" action. This will make block 4 ontable and clear.
Step 3: Pick up block 3 by using the "pick-up" action. This will make block 3 not ontable and not clear, and in the hand.
Step 4: Stack block 3 on block 2 by using the "stack" action. This will make block 3 not clear and on block 2.
Step 5: Pick up block 1 by using the "pick-up" action. This will make block 1 not ontable and not clear, and in the hand.
Step 6: Put down block 1 on the table by using the "put-down" action. This will make block 1 ontable and clear.
Step 7: Repeat steps 3-6 for blocks 2 and 4.

This plan will achieve the goal state of having all blocks on the table and clear. Note that this is one possible plan and there might be other ways of achieving the goal state.

Figure 6: Incorrect plan generated by ChatGPT for a blocksworld problem instance shown in Figure 5. We also later tested natural language description of the PDDL problem as input, the entire PDDL structure as input, and few-shot prompting approaches [Pallagani et al., 2023] but found dismal planning performance from ChatGPT.

A.3 Are transformers memorizing the input (planning problem) and output (plan)?

Our datasets are built to have a wide coverage of combination of problems for each domain. The diversity and complexity of the considered domains can be seen in Section 4.2.1 of the supplementary material. We have created an extensive set of problem instances for each domain (∼200,000 problem instances per domain) that cover a diverse possibility of object configurations. For example, in blocksworld, we consider 2 to 5 block configurations and generate all possible problem instances and corresponding plans (plans are generated using FastDownward). Once the plans are obtained, we use the set function to remove any duplicate plans and then select 18,000 (14,400 for train and 3600 for test sets) problem instances of varying plan lengths with different initial goal conditions and the number of objects to obtain the dataset for Plansformer. We have performed experiments by varying the plan lengths in the train and test sets as reported in Section 4.2 of the supplementary material. We think there is enough evidence to comment that Plansformer is not latching on to correlations present in the dataset and the obtained performance is not due to memorization.

A.4 Are the domains complex enough?

Yes, the considered domains are complex and we support our statements by reporting the results obtained by performing blind search on the considered domain. These results are reported in Section 4.2.1 of the supplementary material.

A.5 What is/are the consequences of this work?

Plansformer is a promising approach to efficiently generate plans without extensive knowledge engineering to represent planning problems and yet getting competent results. Although it is not as reliable as traditional planners yet and has only been shown in this paper for classical planning, we believe that this is just the start. Specifically, the approach can be extended to other types of planning like temporal, metric and epistemic [3], and also made to work for the small cases where it fails currently. Ongoing research is being done on creating a single general model that can scale up to accommodate different planning kinds. General and adaptable planners can be constructed using our model, Plansformer. We are now working on utilizing Plansformer with an existing automated planner (like FastDownward) in the context of a cognitive architecture motivated by the thinking fast and slow theory. In addition, Plansformer may also be used as an instruction-following framework in robot navigation, unmanned vehicles, and embodied artificial intelligence [Huang et al., 2022]. It can also be potentially used as an automated planner in storytelling/dialogue generation [Yao et al., 2019, Pallagani and Srivastava, 2021]. The ability to solve difficult problems in constant time is a key benefit of utilizing Plansformer in these domains (perhaps in conjunction with an automated planner), while current automated planners (like FastDownward) are unable to do so without running out of time or memory.

A.6 What is the significance of using FastDownward planner to generate the dataset?

FastDownward is a traditional planning system that searches the space of world states associated with a planning task in the forward direction using heuristics. In the 4th International Planning Competition at ICAPS 2004, FastDownward secured first place in the "traditional (i.e. propositional, non-optimizing) track". FastDownward comes equipped with a variety of search algorithms by default. We utilize the A* LM-Cut heuristic since it can produce the best plans [Helmert and Domshlak, 2011]. Thus, we use FastDownward to generate a planning dataset consisting of optimal plans.

B Dataset

In this section, we provide examples from the Planning dataset for each of the considered domain - **bw**, **hn**, **gr**, and **dl**. Figure 11 captures the different problem instances.

C Planning vs Plansformer Input

We have talked about how a Plansformer brings about reduced knowledge engineering effort. In Figure 12 and Figure 13, we show the input requirement for an Automated Planner for a **driverlog** problem configuration and Figure 15 shows corresponding input required by Plansformer for the same problem. An automated planner requires two files - (a) **domain.pddl**, and (b) **problem.pddl**. We reduce the knowledge engineering efforts in Plansformer by not requiring:

- explicit mention of *predicates* which are present in **domain.pddl** file.
- explicit mention of *objects* which are present in **problem.pddl** file.

We also have a conversion mechanism for Plansformer and Planning inputs, i.e., given a **domain.pddl** and **problem.pddl** files, we can convert them automatically to input required by Plansformer and vice versa.

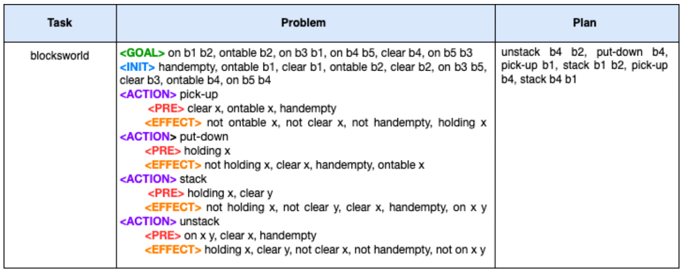
D Training Phase

In this section, we describe the hardware used for computation, training parameters and time taken by different models for training.

D.1 Hardware

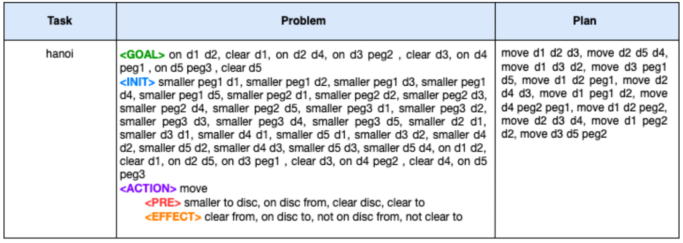
We have used 9 (Dual P-100) 44 (Dual V100) GPU nodes for running our experiments. For training all models, we have made use of 24 cores of CPU run on 1 GPU node. Compute and GPU nodes have 128 GB of RAM and Big Data nodes have 1.5 TB RAM. All nodes have EDR infiniband (100 Gb/s) interconnects, and access to 1.4 PB of GPFS storage. The processor speed is 2.8 GHz.

[3] https://en.wikipedia.org/wiki/Automated_planning_and_scheduling



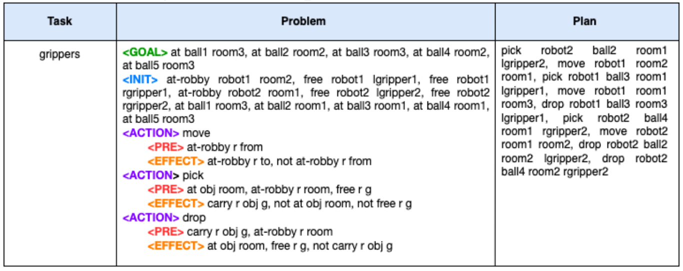
Task	Problem	Plan
blocksworld	<GOAL> on b1 b2, ontable b2, on b3 b1, on b4 b5, clear b4, on b5 b3 <INIT> handempty, ontable b1, clear b1, ontable b2, clear b2, on b3 b5, clear b3, ontable b4, on b5 b4 <ACTION> pick-up <PRE> clear x, ontable x, handempty <EFFECT> not ontable x, not clear x, not handempty, 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 7: Example from **blocksworld** dataset



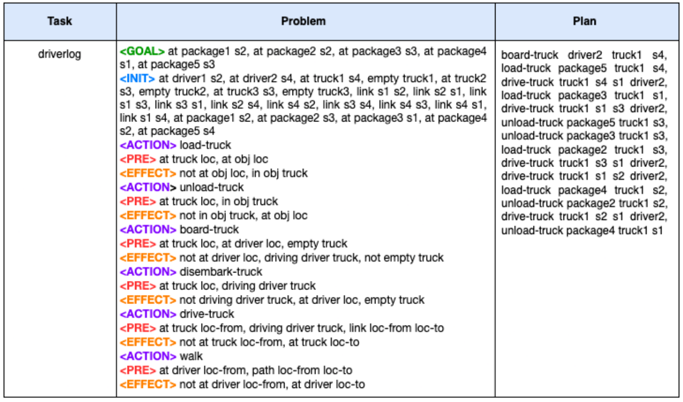
Task	Problem	Plan
hanoi	<GOAL> on d1 d2, clear d1, on d2 d4, on d3 peg2 , clear d3, on d4 peg1 , on d5 peg3 , clear d5 <INIT> smaller peg1 d1, smaller peg1 d2, smaller peg1 d3, smaller peg1 d4, smaller peg1 d5, smaller peg2 d1, smaller peg2 d2, smaller peg2 d3, smaller peg2 d4, smaller peg2 d5, smaller peg3 d1, smaller peg3 d2, smaller peg3 d3, smaller peg3 d4, smaller peg3 d5, smaller d2 d1, smaller d3 d1, smaller d4 d1, smaller d5 d1, smaller d3 d2, smaller d4 d2, smaller d5 d2, smaller d4 d3, smaller d5 d3, smaller d5 d4, on d1 d2, clear d1, on d2 d5, on d3 peg1 , clear d3, on d4 peg2 , clear d4, on d5 peg3 <ACTION> move <PRE> smaller to disc, on disc from, clear disc, clear to <EFFECT> clear from, on disc to, not on disc from, not clear to	move d1 d2 d3, move d2 d5 d4, move d1 d3 d2, move d3 peg1 d5, move d1 d2 peg1, move d2 d4 d3, move d1 peg1 d2, move d4 peg2 peg1, move d1 d2 peg2, move d2 d3 d4, move d1 peg2 d2, move d3 d5 peg2

Figure 8: Example from **hanoi** dataset



Task	Problem	Plan
grippers	<GOAL> at ball1 room3, at ball2 room2, at ball3 room3, at ball4 room2, at ball5 room3 <INIT> at-robby robot1 room2, free robot1 lgripper1, free robot1 rgripper1, at-robby robot2 room1, free robot2 lgripper2, free robot2 rgripper2, at ball1 room3, at ball2 room1, at ball3 room1, at ball4 room1, at ball5 room3 <ACTION> move <PRE> at-robby r from <EFFECT> at-robby r to, not at-robby r from <ACTION> pick <PRE> at obj room, at-robby r room, free r g <EFFECT> carry r obj g, not at obj room, not free r g <ACTION> drop <PRE> carry r obj g, at-robby r room <EFFECT> at obj room, free r g, not carry r obj g	pick robot2 ball2 room1 lgripper2, move robot1 room2 room1, pick robot1 ball3 room1 lgripper1, move robot1 room1 room3, drop robot1 ball3 room3 lgripper1, pick robot2 ball4 room1 rgripper2, move robot2 room1 room2, drop robot2 ball2 room2 lgripper2, drop robot2 ball4 room2 rgripper2

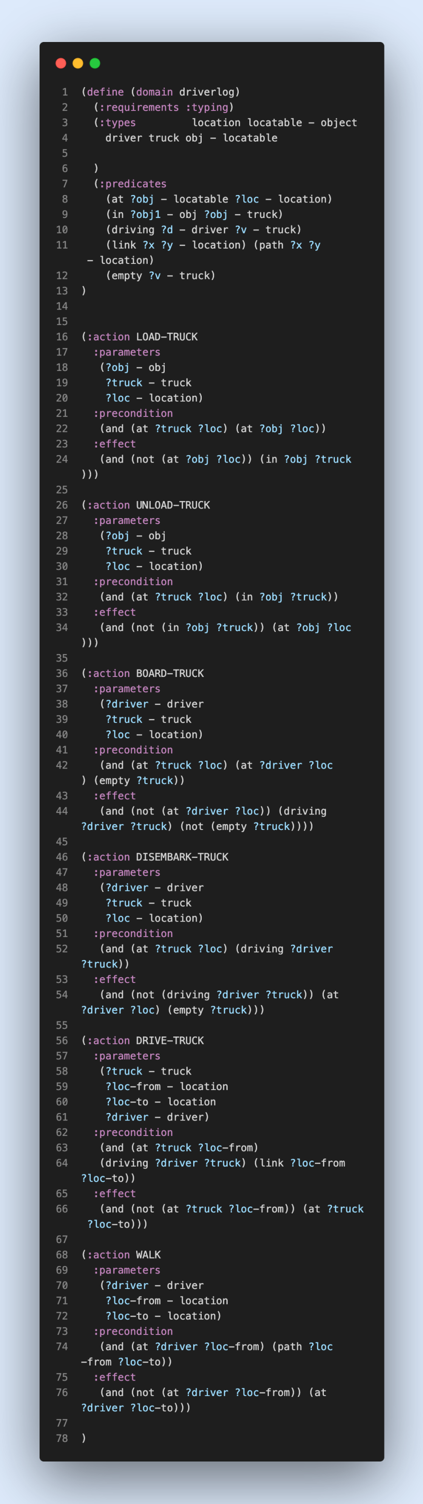
Figure 9: Example from **grippers** dataset



Task	Problem	Plan
driverlog	<GOAL> at package1 s2, at package2 s2, at package3 s3, at package4 s1, at package5 s3 <INIT> at driver1 s2, at driver2 s4, at truck1 s4, empty truck1, at truck2 s3, empty truck2, at truck3 s3, empty truck3, link s1 s2, link s2 s1, link s1 s3, link s3 s1, link s2 s4, link s4 s2, link s3 s4, link s4 s3, link s4 s1, link s1 s4, at package1 s2, at package2 s3, at package3 s1, at package4 s2, at package5 s4 <ACTION> load-truck <PRE> at truck loc, at obj loc <EFFECT> not at obj loc, in obj truck <ACTION> unload-truck <PRE> at truck loc, in obj truck <EFFECT> not in obj truck, at obj loc <ACTION> board-truck <PRE> at truck loc, at driver loc, empty truck <EFFECT> not at driver loc, driving driver truck, not empty truck <ACTION> disembark-truck <PRE> at truck loc, driving driver truck <EFFECT> not driving driver truck, at driver loc, empty truck <ACTION> drive-truck <PRE> at truck loc-from, driving driver truck, link loc-from loc-to <EFFECT> not at truck loc-from, at truck loc-to <ACTION> walk <PRE> at driver loc-from, path loc-from loc-to <EFFECT> not at driver loc-from, at driver loc-to	board-truck driver2 truck1 s4, load-truck package5 truck1 s4, drive-truck truck1 s4 s1 driver2, load-truck package3 truck1 s1, drive-truck truck1 s1 s3 driver2, unload-truck package5 truck1 s3, unload-truck package3 truck1 s3, load-truck package2 truck1 s3, drive-truck truck1 s3 s1 driver2, drive-truck truck1 s1 s2 driver2, load-truck package4 truck1 s2, unload-truck package2 truck1 s2, drive-truck truck1 s2 s1 driver2, unload-truck package4 truck1 s1

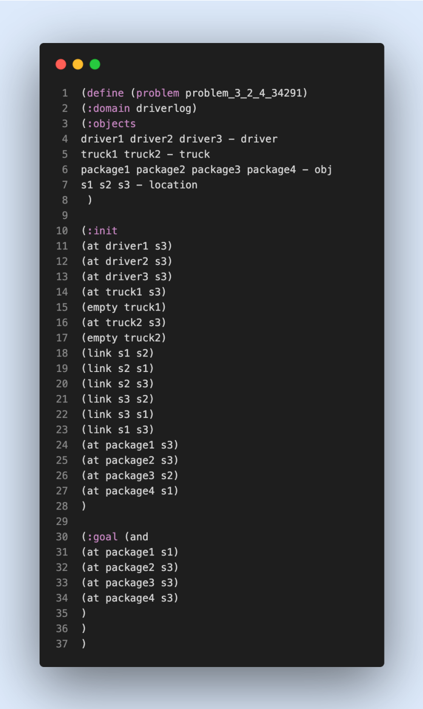
Figure 10: Example from **driverlog** dataset

Figure 11: Problem instances from four different planning domains



```
(define (domain driverlog)
  (:requirements :typing)
  (:types         location locatable - object
    driver truck obj - locatable

  )
  (:predicates
    (at ?obj - locatable ?loc - location)
    (in ?obj1 - obj ?obj - truck)
    (driving ?d - driver ?v - truck)
    (link ?x ?y - location) (path ?x ?y
 - location)
    (empty ?v - truck)
)


(:action LOAD-TRUCK
  :parameters
   (?obj - obj
    ?truck - truck
    ?loc - location)
  :precondition
   (and (at ?truck ?loc) (at ?obj ?loc))
  :effect
   (and (not (at ?obj ?loc)) (in ?obj ?truck
)))

(:action UNLOAD-TRUCK
  :parameters
   (?obj - obj
    ?truck - truck
    ?loc - location)
  :precondition
   (and (at ?truck ?loc) (in ?obj ?truck))
  :effect
   (and (not (in ?obj ?truck)) (at ?obj ?loc
)))

(:action BOARD-TRUCK
  :parameters
   (?driver - driver
    ?truck - truck
    ?loc - location)
  :precondition
   (and (at ?truck ?loc) (at ?driver ?loc
) (empty ?truck))
  :effect
   (and (not (at ?driver ?loc)) (driving
?driver ?truck) (not (empty ?truck))))

(:action DISEMBARK-TRUCK
  :parameters
   (?driver - driver
    ?truck - truck
    ?loc - location)
  :precondition
   (and (at ?truck ?loc) (driving ?driver
?truck))
  :effect
   (and (not (driving ?driver ?truck)) (at
?driver ?loc) (empty ?truck)))

(:action DRIVE-TRUCK
  :parameters
   (?truck - truck
    ?loc-from - location
    ?loc-to - location
    ?driver - driver)
  :precondition
   (and (at ?truck ?loc-from)
   (driving ?driver ?truck) (link ?loc-from
?loc-to))
  :effect
   (and (not (at ?truck ?loc-from)) (at ?truck
 ?loc-to)))

(:action WALK
  :parameters
   (?driver - driver
    ?loc-from - location
    ?loc-to - location)
  :precondition
   (and (at ?driver ?loc-from) (path ?loc
-from ?loc-to))
  :effect
   (and (not (at ?driver ?loc-from)) (at
?driver ?loc-to)))

)
```

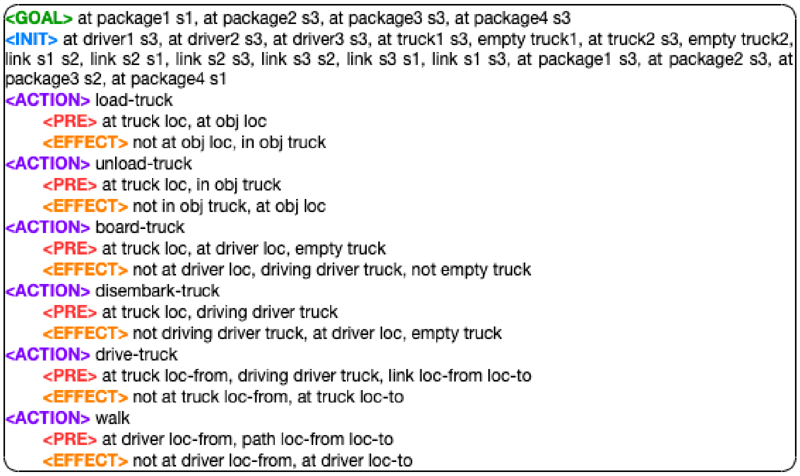
Figure 12: Capturing **driverlog's** environment in `domain.pddl`



```
(define (problem problem_3_2_4_34291)
(:domain driverlog)
(:objects
driver1 driver2 driver3 - driver
truck1 truck2 - truck
package1 package2 package3 package4 - obj
s1 s2 s3 - location
 )

(:init
(at driver1 s3)
(at driver2 s3)
(at driver3 s3)
(at truck1 s3)
(empty truck1)
(at truck2 s3)
(empty truck2)
(link s1 s2)
(link s2 s1)
(link s2 s3)
(link s3 s2)
(link s3 s1)
(link s1 s3)
(at package1 s3)
(at package2 s3)
(at package3 s2)
(at package4 s1)
)

(:goal (and
(at package1 s1)
(at package2 s3)
(at package3 s3)
(at package4 s3)
)
)
)
```

Figure 13: Capturing the current state and desired goal of an object in **driverlog's** environment in `problem.pddl`

Figure 14: Files required to model a problem from **driverlog** in PDDL for execution by an Automated Planner



```
<GOAL> at package1 s1, at package2 s3, at package3 s3, at package4 s3
<INIT> at driver1 s3, at driver2 s3, at driver3 s3, at truck1 s3, empty truck1, at truck2 s3, empty truck2,
link s1 s2, link s2 s1, link s2 s3, link s3 s2, link s3 s1, link s1 s3, at package1 s3, at package2 s3, at
package3 s2, at package4 s1
<ACTION> load-truck
    <PRE> at truck loc, at obj loc
    <EFFECT> not at obj loc, in obj truck
<ACTION> unload-truck
    <PRE> at truck loc, in obj truck
    <EFFECT> not in obj truck, at obj loc
<ACTION> board-truck
    <PRE> at truck loc, at driver loc, empty truck
    <EFFECT> not at driver loc, driving driver truck, not empty truck
<ACTION> disembark-truck
    <PRE> at truck loc, driving driver truck
    <EFFECT> not driving driver truck, at driver loc, empty truck
<ACTION> drive-truck
    <PRE> at truck loc-from, driving driver truck, link loc-from loc-to
    <EFFECT> not at truck loc-from, at truck loc-to
<ACTION> walk
    <PRE> at driver loc-from, path loc-from loc-to
    <EFFECT> not at driver loc-from, at driver loc-to
```

Figure 15: Plansformer's input for the same problem defined in Figure 12

Hyperparameter	Value
Train Batch Size	8
Validation Batch Size	8
Train Epochs	3
Validation Epochs	1
Learning Rate	1e-4
Max Source Text Length	512
Max Target Text Length	150

Table 3: Hyperparameters used for Training

D.2 Training Hyperparameters

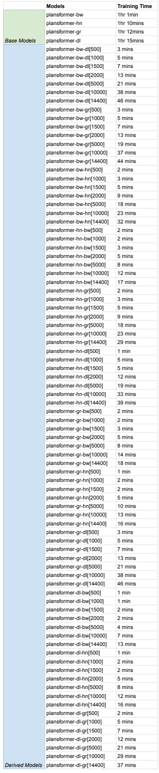
Table 3 captures the hyperparameters set for training our models. For plan generation by all models apart from Codex, we have used beam search with number of beams set to 2, repetition penalty of 2.5, and length penalty set to 1.0. Codex doesn't have the functionality to change the parameters, thus, we have used it in the default setting. On the parameters constituting the considered models, Codex has 12 billion parameters, GPT-2 has 1.2 billion parameters, T5-base has 220 million parameters, and CodeT5-base has 8.35 million parameters.

D.3 Training Time

Figure 16 presents the training time taken by different Plansformer variations. Base models are Plansformer variants directly trained on each of the planning domains with CodeT5 as base. Derived models use a Plansformer base model as a starting point, and further pretrain on other domains. We can see a considerable drop in training time taken by derived models. It is also to be noted that these derived models outperform the base models when entire training data points are used.

E Extended Results

This section adds additional qualitative and quantitative results that cover all the domains and model configurations tried and tested during our experimentation with Plansformer. In our initial testing phase, we have fine-tuned both T5 and CodeT5 with the same blocksworld dataset and hyperparameters and found that fine-tuned T5 gave 32% valid plans, whereas fine-tuned CodeT5 generated 90% valid plans. This is because CodeT5 has syntactically meaningful sequences for code-like structured inputs well defined as opposed to T5, which only deals with natural language. With this intuition that models pre-trained on code have an advantage for plan generation, we proceeded with the choice of using CodeT5 for all our experiments.



	Models	Training Time
Base Models	plansformer-bw	1hr 1min
	plansformer-hn	1hr 10mins
	plansformer-gr	1hr 12mins
	plansformer-dl	1hr 15mins
Derived Models	plansformer-bw-dl[500]	3 mins
	plansformer-bw-dl[1000]	5 mins
	plansformer-bw-dl[1500]	7 mins
	plansformer-bw-dl[2000]	13 mins
	plansformer-bw-dl[5000]	21 mins
	plansformer-bw-dl[10000]	38 mins
	plansformer-bw-dl[14400]	46 mins
	plansformer-bw-gr[500]	3 mins
	plansformer-bw-gr[1000]	5 mins
	plansformer-bw-gr[1500]	7 mins
	plansformer-bw-gr[2000]	13 mins
	plansformer-bw-gr[5000]	19 mins
	plansformer-bw-gr[10000]	37 mins
	plansformer-bw-gr[14400]	44 mins
	plansformer-bw-hn[500]	2 mins
	plansformer-bw-hn[1000]	3 mins
	plansformer-bw-hn[1500]	5 mins
	plansformer-bw-hn[2000]	9 mins
	plansformer-bw-hn[5000]	18 mins
	plansformer-bw-hn[10000]	23 mins
	plansformer-bw-hn[14400]	32 mins
	plansformer-hn-bw[500]	2 mins
	plansformer-hn-bw[1000]	2 mins
	plansformer-hn-bw[1500]	3 mins
	plansformer-hn-bw[2000]	5 mins
	plansformer-hn-bw[5000]	8 mins
	plansformer-hn-bw[10000]	12 mins
	plansformer-hn-bw[14400]	17 mins
	plansformer-hn-gr[500]	2 mins
	plansformer-hn-gr[1000]	3 mins
	plansformer-hn-gr[1500]	5 mins
	plansformer-hn-gr[2000]	9 mins
	plansformer-hn-gr[5000]	18 mins
	plansformer-hn-gr[10000]	23 mins
	plansformer-hn-gr[14400]	29 mins
	plansformer-hn-dl[500]	1 min
	plansformer-hn-dl[1000]	5 mins
	plansformer-hn-dl[1500]	5 mins
	plansformer-hn-dl[2000]	12 mins
	plansformer-hn-dl[5000]	19 mins
	plansformer-hn-dl[10000]	33 mins
	plansformer-hn-dl[14400]	39 mins
	plansformer-gr-bw[500]	2 mins
	plansformer-gr-bw[1000]	2 mins
	plansformer-gr-bw[1500]	3 mins
	plansformer-gr-bw[2000]	5 mins
	plansformer-gr-bw[5000]	8 mins
	plansformer-gr-bw[10000]	14 mins
	plansformer-gr-bw[14400]	18 mins
	plansformer-gr-hn[500]	1 min
	plansformer-gr-hn[1000]	2 mins
	plansformer-gr-hn[1500]	2 mins
	plansformer-gr-hn[2000]	5 mins
	plansformer-gr-hn[5000]	10 mins
	plansformer-gr-hn[10000]	13 mins
	plansformer-gr-hn[14400]	16 mins
	plansformer-gr-dl[500]	3 mins
	plansformer-gr-dl[1000]	5 mins
	plansformer-gr-dl[1500]	7 mins
	plansformer-gr-dl[2000]	13 mins
	plansformer-gr-dl[5000]	21 mins
	plansformer-gr-dl[10000]	38 mins
	plansformer-gr-dl[14400]	46 mins
	plansformer-dl-bw[500]	1 min
	plansformer-dl-bw[1000]	1 min
	plansformer-dl-bw[1500]	2 mins
	plansformer-dl-bw[2000]	2 mins
	plansformer-dl-bw[5000]	4 mins
	plansformer-dl-bw[10000]	7 mins
	plansformer-dl-bw[14400]	13 mins
	plansformer-dl-hn[500]	1 min
	plansformer-dl-hn[1000]	2 mins
	plansformer-dl-hn[1500]	2 mins
	plansformer-dl-hn[2000]	5 mins
	plansformer-dl-hn[5000]	8 mins
	plansformer-dl-hn[10000]	12 mins
	plansformer-dl-hn[14400]	16 mins
	plansformer-dl-gr[500]	2 mins
	plansformer-dl-gr[1000]	5 mins
	plansformer-dl-gr[1500]	7 mins
	plansformer-dl-gr[2000]	12 mins
	plansformer-dl-gr[5000]	21 mins
	plansformer-dl-gr[10000]	29 mins
	plansformer-dl-gr[14400]	37 mins

Figure 16: Training time of different Plansformer variants

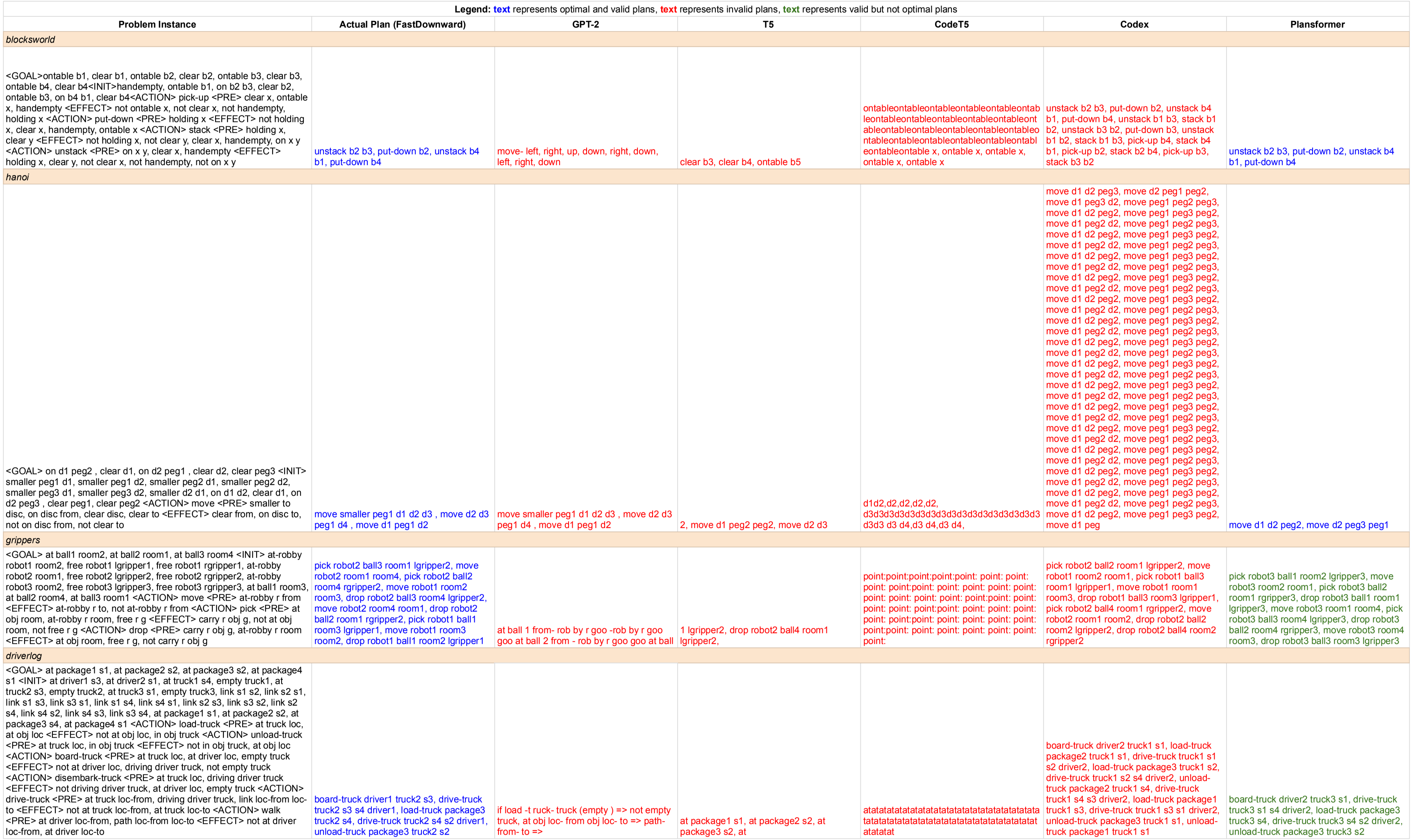
E.1 Qualitative Analysis

Figure 17 shows the output generations obtained by different models under study for a problem instance from each of the planning domains. When reporting Plansformer results, we take the *base models* corresponding to the domain being tested.

E.2 Quantitative Analysis

Figure 19 captures the performance of all models in their ability to generate plans for multiple domains.

Figures 20 to 28 represent the performance of different base models fine-tuned and tested on other domains in graphical manner. Additionally, we also wanted to check Plansformer's capability in plan generation when the plan length of test set is considerably larger than that of the train set. For this purpose, we have created a train set for blocksworld consisting of 2,3 block configurations and a test set with 4,5 block configurations. The test set consists of 100 total instances, 50 from 4 block configuration and 50 from 5 block configuration. The average plan length for the train set and test set is 4 and 10 respectively. Plansformer trained on 2,3 block configurations was able to generate 64% valid plans on problem instances from the test set, showing that our approach can generate valid plans even if the plan length for test set >>> train set. We were able to achieve 64% valid plans in this experimentation as the train set consists of only 162 data points (all possible 2,3 block configurations) as compared to 87% obtained by Plansformer-bw (trained on 14,400 data points) on the same test set.



Legend: **text** represents optimal and valid plans, **text** represents invalid plans, **text** represents valid but not optimal plans

Problem Instance	Actual Plan (FastDownward)	GPT-2	T5	CodeT5	Codex	Plansformer
blocksworld						
<GOAL>ontable b1, clear b1, ontable b2, clear b2, ontable b3, clear b3, ontable b4, clear b4<INIT>handempty, ontable b1, on b2 b3, clear b2, ontable b3, on b4 b1, clear b4<ACTION> pick-up <PRE> clear x, ontable x, handempty <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 b2 b3, put-down b2, unstack b4 b1, put-down b4	move- left, right, up, down, right, down, left, right, down	clear b3, clear b4, ontable b5	ontable x, ontable x, ontable x, ontable x, ontable x	unstack b2 b3, put-down b2, unstack b4 b1, put-down b4, unstack b1 b3, stack b1 b2, unstack b3 b2, put-down b3, unstack b1 b2, stack b1 b3, pick-up b4, stack b4 b1, pick-up b2, stack b2 b4, pick-up b3, stack b3 b2	unstack b2 b3, put-down b2, unstack b4 b1, put-down b4
hanoi						
<GOAL> on d1 peg2 , clear d1, on d2 peg1 , clear d2, clear peg3 <INIT> smaller peg1 d1, smaller peg1 d2, smaller peg2 d1, smaller peg2 d2, smaller peg3 d1, smaller peg3 d2, smaller d2 d1, on d1 d2, clear d1, on d2 peg3 , clear peg1, clear peg2 <ACTION> move <PRE> smaller to disc, on disc from, clear disc, clear to <EFFECT> clear from, on disc to, not on disc from, not clear to	move smaller peg1 d1 d2 d3 , move d2 d3 peg1 d4 , move d1 peg1 d2	move smaller peg1 d1 d2 d3 , move d2 d3 peg1 d4 , move d1 peg1 d2	2, move d1 peg2 peg2, move d2 d3	d1d2,d2,d2,d2,d2, d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3 d3d3 d3 d4,d3 d4,d3 d4,	move d1 d2 peg3, move d2 peg1 peg2, move d1 peg3 d2, move peg1 peg2 peg3, move d1 d2 peg2, move peg1 peg3 peg2, move d1 peg2 d2, move peg1 peg2 peg3, move d1 d2 peg2, move peg1 peg3 peg2, move d1 peg2 d2, move peg1 peg2 peg3, move d1 d2 peg2, move peg1 peg3 peg2, move d1 peg2 d2, move peg1 peg2 peg3, move d1 d2 peg2, move peg1 peg3 peg2, move d1 peg2 d2, move peg1 peg2 peg3, move d1 d2 peg2, move peg1 peg3 peg2, move d1 peg2 d2, move peg1 peg2 peg3, move d1 d2 peg2, move peg1 peg3 peg2, move d1 peg2 d2, move peg1 peg2 peg3, move d1 d2 peg2, move peg1 peg3 peg2, move d1 peg2 d2, move peg1 peg2 peg3, move d1 d2 peg2, move peg1 peg3 peg2, move d1 peg2 d2, move peg1 peg2 peg3, move d1 d2 peg2, move peg1 peg3 peg2, move d1 peg2 d2, move peg1 peg2 peg3, move d1 d2 peg2, move peg1 peg3 peg2, move d1 peg2 d2, move peg1 peg2 peg3, move d1 d2 peg2, move peg1 peg3 peg2, move d1 peg2 d2, move peg1 peg2 peg3, move d1 d2 peg2, move peg1 peg3 peg2, move d1 peg2 d2, move peg1 peg2 peg3, move d1 d2 peg2, move peg1 peg3 peg2, move d1 peg2 d2, move peg1 peg2 peg3, move d1 d2 peg2, move peg1 peg3 peg2, move d1 peg2 d2, move peg1 peg2 peg3, move d1 d2 peg2, move peg1 peg3 peg2, move d1 peg	move d1 d2 peg2, move d2 peg3 peg1
grippers						
<GOAL> at ball1 room2, at ball2 room1, at ball3 room4 <INIT> at-robby robot1 room2, free robot1 lgripper1, free robot1 rgripper1, at-robby robot2 room1, free robot2 lgripper2, free robot2 rgripper2, at-robby robot3 room2, free robot3 lgripper3, free robot3 rgripper3, at ball1 room3, at ball2 room4, at ball3 room1 <ACTION> move <PRE> at-robby r from <EFFECT> at-robby r to, not at-robby r from <ACTION> pick <PRE> at obj room, at-robby r room, free r g <EFFECT> carry r obj g, not at obj room, not free r g <ACTION> drop <PRE> carry r obj g, at-robby r room <EFFECT> at obj room, free r g, not carry r obj g	pick robot2 ball3 room1 lgripper2, move robot2 room1 room4, pick robot2 ball2 room4 rgripper2, move robot1 room2 room3, drop robot2 ball3 room4 lgripper2, move robot2 room4 room1, drop robot2 ball2 room1 rgripper2, pick robot1 ball1 room3 lgripper1, move robot1 room3 room2, drop robot1 ball1 room2 lgripper1	at ball 1 from- rob by r goo -rob by r goo goo at ball 2 from - rob by r goo goo at ball	1 lgripper2, drop robot2 ball4 room1 lgripper2,	point:point:point:point:point: point: point: point: point:point: point: point: point: point: point: point: point: point: point:point: point: point: point: point:point: point: point: point: point: point: point:point: point: point: point: point: point: point: point: point:point: point: point: point: point: point: point:	pick robot2 ball2 room1 lgripper2, move robot1 room2 room1, pick robot1 ball3 room1 lgripper1, move robot1 room1 room3, drop robot1 ball3 room3 lgripper1, pick robot2 ball4 room1 rgripper2, move robot2 room1 room2, drop robot2 ball2 room2 lgripper2, drop robot2 ball4 room2 rgripper2	pick robot3 ball1 room2 lgripper3, move robot3 room2 room1, pick robot3 ball2 room1 rgripper3, drop robot3 ball1 room1 lgripper3, move robot3 room1 room4, pick robot3 ball3 room4 lgripper3, drop robot3 ball2 room4 rgripper3, move robot3 room4 room3, drop robot3 ball3 room3 lgripper3
driverlog						
<GOAL> at package1 s1, at package2 s2, at package3 s2, at package4 s1 <INIT> at driver1 s3, at driver2 s1, at truck1 s4, empty truck1, at truck2 s3, empty truck2, at truck3 s1, empty truck3, link s1 s2, link s2 s1, link s1 s3, link s3 s1, link s1 s4, link s4 s1, link s2 s3, link s3 s2, link s2 s4, link s4 s2, link s4 s3, link s3 s4, at package1 s1, at package2 s2, at package3 s4, at package4 s1 <ACTION> load-truck <PRE> at truck loc, at obj loc <EFFECT> not at obj loc, in obj truck <ACTION> unload-truck <PRE> at truck loc, in obj truck <EFFECT> not in obj truck, at obj loc <ACTION> board-truck <PRE> at truck loc, at driver loc, empty truck <EFFECT> not at driver loc, driving driver truck, not empty truck <ACTION> disembark-truck <PRE> at truck loc, driving driver truck <EFFECT> not driving driver truck, at driver loc, empty truck <ACTION> drive-truck <PRE> at truck loc-from, driving driver truck, link loc-from loc-to <EFFECT> not at truck loc-from, at truck loc-to <ACTION> walk <PRE> at driver loc-from, path loc-from loc-to <EFFECT> not at driver loc-from, at driver loc-to	board-truck driver1 truck2 s3, drive-truck truck2 s3 s4 driver1, load-truck package3 truck2 s4, drive-truck truck2 s4 s2 driver1, unload-truck package3 truck2 s2	if load -t ruck- truck (empty) => not empty truck, at obj loc- from obj loc- to => path- from- to =>	at package1 s1, at package2 s2, at package3 s2, at	atatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatat	board-truck driver2 truck1 s1, load-truck package2 truck1 s1, drive-truck truck1 s1 s2 driver2, load-truck package3 truck1 s2, drive-truck truck1 s2 s4 driver2, unload-truck package2 truck1 s4, drive-truck truck1 s4 s3 driver2, load-truck package1 truck1 s3, drive-truck truck1 s3 s1 driver2, unload-truck package3 truck1 s1, unload-truck package1 truck1 s1	board-truck driver2 truck3 s1, drive-truck truck3 s1 s4 driver2, load-truck package3 truck3 s4, drive-truck truck3 s4 s2 driver2, unload-truck package3 truck3 s2

Figure 17: Output generations from different models for planning problem instances

Domains	**A* + LM-Cut** (Generated States & Evaluated States)	**Blind Search** (Generated States & Evaluated States)
bw	51 & 35	707 & 334
hn	141 & 55	201 & 76
gr	33520 & 1347	3795846 & 381627
dl	188 & 131	568337 & 110498

Table 4: An analysis of the complexity of the planning domains

E.2.1 Complexity and Diversity of Planning Domains

We report the complexity of the considered planning domains by implementing a blind search on the non-randomized test sets for each domain (3600 problem instances per domain). As blind search works with no information about the search space, it gives us an idea about the difficulty of navigating a domain. Table 4 captures the results obtained by A* + LM-Cut (informed search) and Breadth-First Search (uninformed/blind search). Here, generated states refer to the total number of states obtained for the given problem instance, whereas, the evaluated states refer to the number of traversed generated states to arrive at the goal state. The results obtained by blind search provide an insight about the complexity of the domains (with hn and bw on the easier side vs dl and gr on the harder side). Both the search strategies solved 3600 problems, but blind search took an average of 214 seconds to solve the test bed, whereas A* + LM-Cut took 14.72 seconds.

All the considered planning domains have a diverse set of state-action space. Because of the diversity of these domains, we observed that in "multi-domain plan generation using a single model" setup from Section 4.2.2 in supplementary material, single model trained on problem instances from all four domains has an unfair advantage for easier domains such as bw (5.79% increase in terms of valid plans on Plansformer-bw vs 90.04% in Table 2) compared to performance drop for harder domains such as gr and dl (around a 4.5% to 13.5% decrease in valid plans).

E.2.2 Transfer Learning

We have reported the advantages of transfer learning when using Plansformer-bw as the base model and further fine-tuning it on **hn** for all the varying data points in the main paper. Similarly, Figures 29 and 30 show a homogeneous trend for the domains **dl** and **gr** on further fine-tuning of the Plansformer-bw base model.

E.2.3 Multi-domain Plan Generation using a Single Model

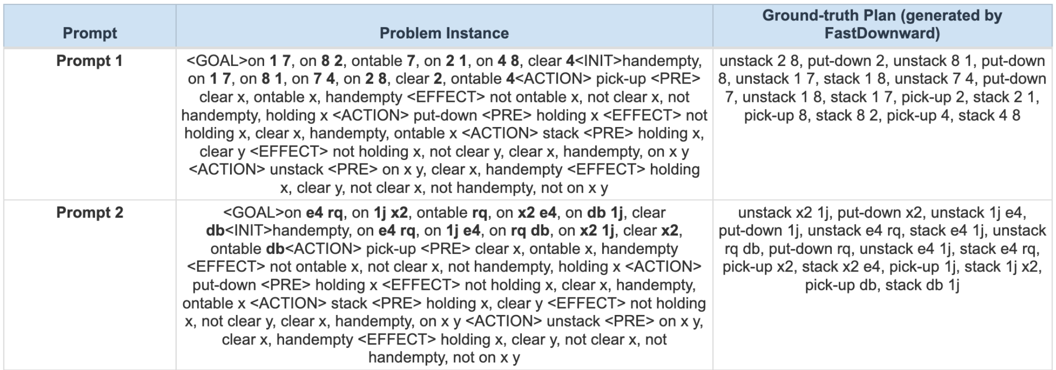
We have additionally trained a single Plansformer model on all the training data points belonging to all the four domains - **bw, hn, dl,** and **gr**. Each domain consists of 14,400 training data points, thus, the single Plansformer model is trained on a total of 57,600 data points. The obtained single Plansformer model is tested on the validation datasets corresponding to all the four domains. Each dataset consists of 3000 problem instances. Table 8 reports the performance obtained by the single Plansformer model on all four domains in terms of plan validation. We observe that the single model is able to perform relatively comparable to the *base models*. The single Plansformer trained on all models outperforms Plansformer-bw by 5.79% (in terms of valid plans) and has around a 4.5% to 13.5% decrease in valid plans for the other three domains.

E.2.4 Object Name Randomization

We wanted to measure the role of object name randomization in Plansformer's performance. For the experimental setup, we have randomized the object names for every instance present in each of the four datasets (**bw**, **hn**,**gr**, and **dl**). We have considered two different types of prompts, varying the object names:

- **Prompt 1:** The datasets generated using Prompt 1 consist of only single-digit numbers as the object names.
- **Prompt 2:** The objects names in the datasets generated using Prompt 2 consist of an alphanumeric string of length 2.

Figure 18 shows examples of two prompts generated for the same problem instance from **bw** taken from the original dataset.



Prompt	Problem Instance	Ground-truth Plan (generated by FastDownward)
Prompt 1	<GOAL>on **1 7**, on **8 2**, ontable **7**, on **2 1**, on **4 8**, clear **4**<INIT>handempty, on **1 7**, on **8 1**, on **7 4**, on **2 8**, clear **2**, ontable **4**<ACTION> pick-up <PRE> clear x, ontable x, handempty <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 2 8, put-down 2, unstack 8 1, put-down 8, unstack 1 7, stack 1 8, unstack 7 4, put-down 7, unstack 1 8, stack 1 7, pick-up 2, stack 2 1, pick-up 8, stack 8 2, pick-up 4, stack 4 8
Prompt 2	<GOAL>on **e4 rq**, on **1j x2**, ontable **rq**, on **x2 e4**, on **db 1j**, clear **db**<INIT>handempty, on **e4 rq**, on **1j e4**, on **rq db**, on **x2 1j**, clear **x2**, ontable **db**<ACTION> pick-up <PRE> clear x, ontable x, handempty <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 x2 1j, put-down x2, unstack 1j e4, put-down 1j, unstack e4 rq, stack e4 1j, unstack rq db, put-down rq, unstack e4 1j, stack e4 rq, pick-up x2, stack x2 e4, pick-up 1j, stack 1j x2, pick-up db, stack db 1j

Figure 18: Examples for Prompt 1 and Prompt 2 for a single instance in blocksworld

Model	**Valid Plans (out of 500)**
GPT-3 + Prompt Conditioning [Valmeekam et al., 2022]	0.60%
Plansformer-bw	66.00%
Plansformer-bw(random)-Prompt1	67.40%
Plansformer-bw(random)-Prompt2	27.52%

Table 5: Comparison of different Plansformer models built on *blocksworld* and tested on the dataset released in Valmeekam et al. [2022]

We trained Plansformer models for every domain using the randomized train sets. As the randomized train and test sets are built by mapping object names in the original dataset following Prompt 1 and Prompt 2 nomenclature, we have 14,400 and 3600 instances in the new randomized datasets. From now on, we will be referring to the randomized Plansformer models as Plansformer-[*domain*](random)-Prompt[*x*], where, *domain* and *x* are variables specifying the domain on which Plansformer is trained on and the prompt number followed for object names. We then tested the randomized models on the dataset released in Valmeekam et al. [2022]. The object names in Valmeekam et al. [2022] are single lettered alphabets. The performance of different models on 500 **bw** problem instances from Valmeekam et al. [2022] are shown in Table 5.

From Table 5, it can be observed that the model trained on a randomized **bw** dataset according to Prompt 1 achieves 67.40% valid plans, followed by Plansformer-bw (original model trained on the non-randomized dataset with objects named b1, b2, and so on). It is observed that a drop in performance is encountered with increasing addition of alphabets in the object names. For example, an object name "abcd12" would be treated as an out-of-vocabulary token by LLMs (and hence, Plansformer) and eventually be tokenized in no way that is meaningful for the application in study. This experimentation provides additional insights that all variants of Plansformer perform better when the object names look different than the action names. On the other hand, if we follow a simple technique of mapping object names [Huang et al., 2022] from the new test set to the object names similar to the train set and generate a plan with this mapped problem instance as input to any Plansformer-bw model (randomized or otherwise), we were able to achieve around 80% valid plans (Plansformer-bw(random)-Prompt1 gave 81.97% valid plans, Plansformer-bw(random)-Prompt2 gave 80.29% valid plans, and Plansformer-bw gave 82.56% valid plans).

Rest of the experiments on object randomization are carried out with Prompt 2 in order to understand the worst-case performance of Plansformer possible for plan generation. We have tested the randomized models and base models on both randomized and original validation datasets, as shown in Table 6. It can be seen that the randomized Plansformer models generate lesser valid plans than the base models, with a maximum drop in performance of 15.41% in the Hanoi domain (Plansformer-hn produces 84.97% valid plans on hn vs 69.56% valid plans with randomized train and test domains). The results show that randomization of variables can have a performance impact ranging from maintaining the performance to a degradation based on the difficulty/complexity of the domain. Since an LLM learns over its inputs, such a performance is not surprising. One easy way to retain performance is by mapping the unique parameter names in test problems to internal naming scheme (used during training) that does not degrade the performance.

E.2.5 Augmented Dataset

With additional copies of the same problem instance but consisting of different objects names using nomenclatures Prompt 1 and 2, we wanted to created an augmented dataset for each domain. Thus, the augmented dataset consists a total of 43,200 problem instances for each domain. A single Plansformer model is trained using this augmented dataset and Table 7 shows the results obtained by such a Plansformer model trained on augmented **bw** dataset.

Plansformer model trained on augmenented data outperforms other existing Plansformer-bw variants by a fair margin. The original Plansformer-bw base model generated 90.04% valid plans (tested on non randomized bw test-bed), whereas the new model trained on the larger dataset obtained 97.77% on the same test bed (second row in the table above). The result is quite interesting as the new model trained on 3 different variants of the same problem instance gave a performance boost of 7%, providing us with alternate ways for the language model on planning domain to learn better.

F Plansformer Architecture

Figure 31 shows the layer-wise architecture that makes up Plansformer. All these layers are updated during the fine-tuning process. We inherit this architecture from CodeT5, and do not freeze any layers during our fine-tuning process involved for constructing Plansformer.

Model	Train Domain	Test Domain	Valid Plans (out of 3600)	Cost-Optimal Plans (out of 3600)	Invalid Plans (out of 3600)
Plansformer-bw(random)-Prompt2	Randomized bw-Prompt2	Randomized bw-Prompt2-test	89.83%	86.42%	Failed = 10.17%, Incomplete = 0%
Plansformer-hn(random)-Prompt2	Randomized hn-Prompt2	Randomized hn-Prompt2-test	69.56%	66.78%	Failed = 24.97%, Incomplete = 5.47%
Plansformer-gr(random)-Prompt2	Randomized gr-Prompt2	Randomized gr-Prompt2-test	75.89%	70.56%	Failed = 22.47%, Incomplete = 1.64%
Plansformer-dl(random)-Prompt2	Randomized dl-Prompt2	Randomized dl-Prompt2-test	67.86%	63.08%	Failed = 19.25%, Incomplete = 12.89%
Plansformer-bw(random)-Prompt2	Randomized bw-Prompt2	bw-test	48.22%	45.22%	Failed = 45.17%, Incomplete = 6.61%
Plansformer-hn(random)-Prompt2	Randomized hn-Prompt2	hn-test	10.42%	8.72%	Failed = 69.33%, Incomplete = 20.25%
Plansformer-gr(random)-Prompt2	Randomized gr-Prompt2	gr-test	3.06%	2.86%	Failed = 53.86%, Incomplete = 43.08%
Plansformer-dl(random)-Prompt2	Randomized dl-Prompt2	dl-test	2.69%	1.04%	Failed = 33.94%, Incomplete = 63.37%
Plansformer-bw	bw	Randomized bw-Prompt2-test	70.69%	68.14%	Failed = 29.25%, Incomplete = 0.06%
Plansformer-hn	hn	Randomized hn-Prompt2-test	34.58%	34.03%	Failed = 43.72%, Incomplete = 21.7%
Plansformer-gr	gr	Randomized gr-Prompt2-test	28.39%	27.31%	Failed = 2.08%, Incomplete = 69.53%
Plansformer-dl	dl	Randomized dl-Prompt2-test	20.64%	20.64%	Failed = 0.06%, Incomplete = 79.30%

Table 6: Results of object name randomization using Prompt 2

		Test Domain - *(tested using 3600 problems for each domain)*											
		blocksworld (bw)			hanoi (hn)			gripper (gr)			driverlog (dl)		
		Valid Plans	Invalid Plans	Cost-Optimal Plans	Valid Plans	Invalid Plans	Cost-Optimal Plans	Valid Plans	Invalid Plans	Cost-Optimal Plans	Valid Plans	Invalid Plans	Cost-Optimal Plans
Trained using **14,400** problem instances for each domain	plansformer-bw	90.04%	Failed = 9.94%, Incomplete = 0.02%	88.44%	0.00%	Failed = 100%, Incomplete = 0%	0.00%	0.00%	Failed = 100%, Incomplete = 0%	0.00%	0.00%	Failed = 0%, Incomplete = 100%	0.00%
	plansformer-hn	0.00%	Failed = 2.58%, Incomplete = 97.42%	0.00%	84.97%	Failed = 14.72%, Incomplete = 0.31%	82.58%	0.00%	Failed = 1.14%, Incomplete = 98.86%	0.00%	0.00%	Failed = 0%, Incomplete = 100%	0.00%
	plansformer-gr	0.00%	Failed = 1.75%, Incomplete = 98.25%	0.00%	0.00%	Failed = 8.83%, Incomplete = 91.16%	0.00%	82.97%	Failed = 16.61%, Incomplete = 0.42%	69.47%	0.00%	Failed = 0%, Incomplete = 100%	0.00%
	plansformer-dl	0.00%	Failed = 0%, Incomplete = 100%	0.00%	0.00%	Failed = 0%, Incomplete = 100%	0.00%	0.00%	Failed = 100%, Incomplete = 0%	0.00%	76.56%	Failed = 23.44%, Incomplete = 0%	52.61%
	plansformer-bw-dl[500]	68.28%	Failed = 31.72%, Incomplete = 0%	66.56%	0.00%	Failed = 0%, Incomplete = 100%	0.00%	0.00%	Failed = 23.17%, Incomplete = 76.83%	0.00%	0.00%	Failed = 99.81%, Incomplete = 0.19%	0.00%
	plansformer-bw-dl[1000]	59.22%	Failed = 40.78%, Incomplete = 0%	58.44%	0.00%	Failed = 0.03%, Incomplete = 99.97%	0.00%	0.00%	Failed = 0%, Incomplete = 100%	0.00%	1.92%	Failed = 97.97%, Incomplete = 0.11%	1.58%
	plansformer-bw-dl[1500]	61.67%	Failed = 38.33%, Incomplete = 0%	60.86%	0.00%	Failed = 0.17%, Incomplete = 99.83%	0.00%	0.00%	Failed = 83.94%, Incomplete = 16.06%	0.00%	17.57%	Failed = 82.26%, Incomplete = 0.17%	15.99%
	plansformer-bw-dl[2000]	49.57%	Failed = 50.43%, Incomplete = 0%	48.66%	0.00%	Failed = 0.17%, Incomplete = 99.83%	0.00%	0.00%	Failed = 83.94%, Incomplete = 16.06%	0.00%	37.81%	Failed = 61.94%, Incomplete = 0.25%	27.39%
	plansformer-bw-dl[5000]	33.44%	Failed = 56.69%, Incomplete = 9.86%	32.50%	0.00%	Failed = 0.14%, Incomplete = 99.86%	0.00%	0.00%	Failed = 94.69%, Incomplete = 5.31%	0.00%	82.72%	Failed = 17.06%, Incomplete = 0.22%	64.81%
	plansformer-bw-dl[10000]	3.25%	Failed = 10.64%, Incomplete = 86.11%	3.25%	0.00%	Failed = 0%, Incomplete = 100%	0.00%	0.00%	Failed = 100%, Incomplete = 0%	0.00%	78.56%	Failed = 10.22%, Incomplete = 11.22%	62.33%
	plansformer-bw-dl[14400]	0.00%	Failed = 1%, Incomplete = 99%	0.00%	0.00%	Failed = 2.56%, Incomplete = 97.44%	0.00%	0.00%	Failed = 100%, Incomplete = 0%	0.00%	93.75%	Failed = 6%, Incomplete = 0.25%	74.56%
	plansformer-bw-gr[500]	32.33%	Failed = 40.03%, Incomplete = 27.64%	32.14%	0.00%	Failed = 4.89%, Incomplete = 95.11%	0.00%	2.86%	Failed = 96.64%, Incomplete = 0.5%	0.97%	0.00%	Failed = 0%, Incomplete = 100%	0.00%
	plansformer-bw-gr[1000]	17.03%	Failed = 13.81%, Incomplete = 69.17%	17.00%	0.00%	Failed = 0.67%, Incomplete = 99.33%	0.00%	9.42%	Failed = 89.47%, Incomplete = 1.11%	7.61%	0.00%	Failed = 0%, Incomplete = 100%	0.00%
	plansformer-bw-gr[1500]	6.39%	Failed = 10.78%, Incomplete = 82.83%	6.28%	0.00%	Failed = 9.72%, Incomplete = 90.28%	0.00%	20.53%	Failed = 79.42%, Incomplete = 0.05	9.03%	0.00%	Failed = 0%, Incomplete = 100%	0.00%

Figure 19: Plan Validation metrics for all Plansformer variants

	plansformer-bw-gr[2000]	9.31%	Failed = 18.42%, Incomplete = 72.28%	9.31%	0.00%	Failed = 0.92%, Incomplete = 99.08%	0.00%	26.56%	Failed = 73.11%, Incomplete = 0.33%	11.31%	0.00%	Failed = 0%, Incomplete = 100%	0.00%
	plansformer-bw-gr[5000]	1.28%	Failed = 3.64%, Incomplete = 95.08%	1.28%	0.00%	Failed = 39.03%, Incomplete = 60.97%	0.00%	64.22%	Failed = 35.78%, Incomplete = 0%	38.22%	0.00%	Failed = 0%, Incomplete = 100%	0.00%
	plansformer-bw-gr[10000]	0.00%	Failed = 1.56%, Incomplete = 98.44%	0.00%	0.00%	Failed = 6.06%, Incomplete = 93.94%	0.00%	77.22%	Failed = 22.75%, Incomplete = 0.03%	47.83%	0.00%	Failed = 0%, Incomplete = 100%	0.00%
	plansformer-bw-gr[14400]	0.00%	Failed = 0%, Incomplete = 100%	0.00%	0.00%	Failed = 2.03%, Incomplete = 97.97%	0.00%	87.17%	Failed = 12.80%, Incomplete = 0.03%	60.86%	0.00%	Failed = 0%, Incomplete = 100%	0.00%
	plansformer-bw-hn[500]	40.86%	Failed = 58.33%, Incomplete = 0.81%	37.42%	3.11%	Failed = 96.89%, Incomplete = 0%	2.92%	0.00%	Failed = 4.92%, Incomplete = 95.08%	0.00%	0.00%	Failed = 0%, Incomplete = 100%	0.00%
	plansformer-bw-hn[1000]	37.81%	Failed = 57.81%, Incomplete = 4.39%	35.61%	14.72%	Failed = 85.28%, Incomplete = 0%	13.18%	0.00%	Failed = 4.92%, Incomplete = 95.08%	0.00%	0.00%	Failed = 0%, Incomplete = 100%	0.00%
	plansformer-bw-hn[1500]	31.64%	Failed = 60.53%, Incomplete = 7.83%	27.70%	35.28%	Failed = 64.44%, Incomplete = 0.28%	32.58%	0.00%	Failed = 14.58%, Incomplete = 85.42%	0.00%	0.00%	Failed = 0%, Incomplete = 100%	0.00%
	plansformer-bw-hn[2000]	17.97%	Failed = 56.22%, Incomplete = 25.81%	16.56%	48.19%	Failed = 51.78%, Incomplete = 0.03%	44.50%	0.00%	Failed = 9.06%, Incomplete = 90.94%	0.00%	0.00%	Failed = 0%, Incomplete = 100%	0.00%
	plansformer-bw-hn[5000]	1.18%	Failed = 2.74%, Incomplete = 96.08%	1.01%	60.25%	Failed = 39.52%, Incomplete = 0.22%	57.27%	0.00%	Failed = 0%, Incomplete = 100%	0.00%	0.00%	Failed = 0%, Incomplete = 100%	0.00%
	plansformer-bw-hn[10000]	0.00%	Failed = 1.22%, Incomplete = 98.78%	0.00%	88.28%	Failed = 11.66%, Incomplete = 0.06%	86.64%	0.00%	Failed = 0%, Incomplete = 100%	0.00%	0.00%	Failed = 0%, Incomplete = 100%	0.00%
	plansformer-bw-hn[14400]	0.00%	Failed = 1.55%, Incomplete = 98.44%	0.00%	97.05%	Failed = 2.94%, Incomplete = 0%	95.22%	0.00%	Failed = 1.44%, Incomplete = 98.56%	0.00%	0.00%	Failed = 100%, Incomplete = 0%	0.00%
	plansformer-hn-bw[500]	11.75%	Failed = 88.14%, Incomplete = 0.11%	11.06%	0.00%	Failed = 2.34%, Incomplete = 97.61%	0.00%	0.00%	Failed = 100%, Incomplete = 0%	0.00%	0.00%	Failed = 0%, Incomplete = 100%	0.00%
	plansformer-hn-bw[1000]	20.61%	Failed = 78.97%, Incomplete = 0.42%	18.28%	0.00%	Failed = 0.28%, Incomplete = 99.72%	0.00%	0.00%	Failed = 100%, Incomplete = 0%	0.00%	0.00%	Failed = 0%, Incomplete = 100%	0.00%
	plansformer-hn-bw[1500]	27.58%	Failed = 70.36%, Incomplete = 2.06%	22.00%	0.00%	Failed = 0.14%, Incomplete = 99.86%	0.00%	0.00%	Failed = 100%, Incomplete = 0%	0.00%	0.00%	Failed = 0%, Incomplete = 100%	0.00%
	plansformer-hn-bw[2000]	46.53%	Failed = 53.31%, Incomplete = 0.17%	37.89%	0.00%	Failed = 0.56%, Incomplete = 99.44%	0.00%	0.00%	Failed = 100%, Incomplete = 0%	0.00%	0.00%	Failed = 0%, Incomplete = 100%	0.00%

	plansformer-hn-bw[5000]	69.64%	Failed = 29.56%, Incomplete = 0.81%	64.72%	0.00%	Failed = 0.22%, Incomplete = 99.78%	0.00%	0.00%	Failed = 100%, Incomplete = 0%	0.00%	0.00%	Failed = 0%, Incomplete = 100%	0.00%
	plansformer-hn-bw[10000]	90.36%	Failed = 9.58%, Incomplete = 0.06%	87.97%	0.00%	Failed = 4.06%, Incomplete = 95.94%	0.00%	0.00%	Failed = 100%, Incomplete = 0%	0.00%	0.00%	Failed = 0%, Incomplete = 100%	0.00%
	plansformer-hn-bw[14400]	95.44%	Failed = 4.56%, Incomplete = 0%	93.03%	0.00%	Failed = 2.28%, Incomplete = 97.72%	0.00%	0.00%	Failed = 100%, Incomplete = 0%	0.00%	0.00%	Failed = 0%, Incomplete = 100%	0.00%
	plansformer-hn-gr[500]	0.00%	Failed = 1.56%, Incomplete = 98.44%	0.00%	6.92%	Failed = 75.81%, Incomplete = 17.28%	6.56%	3.17%	Failed = 95.58%, Incomplete = 1.25%	1.78%	0.00%	Failed = 1.56%, Incomplete = 98.44%	0.00%
	plansformer-hn-gr[1000]	0.00%	Failed = 1.06%, Incomplete = 98.94%	0.00%	12.89%	Failed = 31.42%, Incomplete = 55.69%	12.33%	10.64%	Failed = 89.22%, Incomplete = 0.14%	5.92%	0.00%	Failed = 0%, Incomplete = 100%	0.00%
	plansformer-hn-gr[1500]	0.00%	Failed = 0.5%, Incomplete = 99.5%	0.00%	8.08%	Failed = 10.31%, Incomplete = 81.61%	7.94%	15.61%	Failed = 84.36%, Incomplete = 0.03%	10.08%	0.00%	Failed = 100.0%, Incomplete = 0.0%	0.00%
	plansformer-hn-gr[2000]	0.00%	Failed = 1.97%, Incomplete = 98.03%	0.00%	6.72%	Failed = 1.5%, Incomplete = 91.78%	6.72%	22.36%	Failed = 77.36%, Incomplete = 0.28%	16.06%	0.00%	Failed = 100.0%, Incomplete = 0.0%	0.00%
	plansformer-hn-gr[5000]	0.00%	Failed = 27.69%, Incomplete = 72.31%	0.00%	2.64%	Failed = 1.08%, Incomplete = 96.28%	2.64%	49.31%	Failed = 50.33%, Incomplete = 0.36%	33.72%	0.00%	Failed = 100.0%, Incomplete = 0.0%	0.00%
	plansformer-hn-gr[10000]	0.00%	Failed = 41.14%, Incomplete = 58.86%	0.00%	0.00%	Failed = 0.0%, Incomplete = 100.0%	0.00%	75.58%	Failed = 24.25%, Incomplete = 0.17%	48.56%	0.00%	Failed = 100.0%, Incomplete = 0.0%	0.00%
	plansformer-hn-gr[14400]	0.00%	Failed = 5.28%, Incomplete = 94.72%	0.00%	0.00%	Failed = 0.11%, Incomplete = 99.89%	0.00%	65.50%	Failed = 34.31%, Incomplete = 0.19%	39.44%	0.00%	Failed = 100.0%, Incomplete = 0.0%	0.00%
	plansformer-hn-dl[500]	0.00%	Failed = 0.97%, Incomplete = 99.03%	0.00%	9.19%	Failed = 87.44%, Incomplete = 3.36%	8.86%	0.00%	Failed = 100.0%, Incomplete = 0.0%	0.00%	0.11%	Failed = 99.44%, Incomplete = 0.44%	0.08%
	plansformer-hn-dl[1000]	0.00%	Failed = 1.11%, Incomplete = 98.89%	0.00%	11.25%	Failed = 84.86%, Incomplete = 3.89%	10.83%	0.00%	Failed = 100.0%, Incomplete = 0.0%	0.00%	6.97%	Failed = 92.47%, Incomplete = 0.56%	4.64%
	plansformer-hn-dl[1500]	0.00%	Failed = 1.67%, Incomplete = 98.33%	0.00%	10.19%	Failed = 56.89%, Incomplete = 32.92%	10.11%	0.00%	Failed = 100.0%, Incomplete = 0.0%	0.00%	27.11%	Failed = 72.72%, Incomplete = 0.17%	18.19%
	plansformer-hn-dl[2000]	0.00%	Failed = 1.92%, Incomplete = 98.08%	0.00%	11.56%	Failed = 50.0%, Incomplete = 38.44%	11.33%	0.00%	Failed = 100.0%, Incomplete = 0.0%	0.00%	40.69%	Failed = 59.14%, Incomplete = 0.17%	30.06%
	plansformer-hn-dl[5000]	0.00%	Failed = 37.17%, Incomplete = 62.83%	0.00%	0.75%	Failed = 0.14%, Incomplete = 99.11%	0.75%	0.00%	Failed = 100.0%, Incomplete = 0.0%	0.00%	72.75%	Failed = 26.72%, Incomplete = 0.53%	55.83%

	plansformer-hn-dl[10000]	0.00%	Failed = 0.03%, Incomplete = 99.97%	0.00%	0.03%	Failed = 0.11%, Incomplete = 99.86%	0.03%	0.00%	Failed = 100.0%, Incomplete = 0.0%	0.00%	86.00%	Failed = 13.92%, Incomplete = 0.08%	67.89%
	plansformer-hn-dl[14400]	0.00%	Failed = 0.31%, Incomplete = 99.69%	0.00%	0.00%	Failed = 0.92%, Incomplete = 99.08%	0.00%	0.00%	Failed = 100.0%, Incomplete = 0.0%	0.00%	90.06%	Failed = 9.64%, Incomplete = 0.31%	72.61%
	plansformer-gr-bw[500]	22.33%	Failed = 77.61%, Incomplete = 0.06%	17.69%	0.00%	Failed = 0.0%, Incomplete = 100.0%	0.00%	1.92%	Failed = 97.86%, Incomplete = 0.22%	0.75%	0.00%	Failed = 100.0%, Incomplete = 0.0%	0.00%
	plansformer-gr-bw[1000]	48.11%	Failed = 51.89%, Incomplete = 0.0%	35.75%	0.00%	Failed = 0.03%, Incomplete = 99.97%	0.00%	1.50%	Failed = 98.06%, Incomplete = 0.44%	0.75%	0.00%	Failed = 100.0%, Incomplete = 0.0%	0.00%
	plansformer-gr-bw[1500]	75.19%	Failed = 23.06%, Incomplete = 1.75%	60.56%	0.00%	Failed = 0.08%, Incomplete = 99.92%	0.00%	3.39%	Failed = 96.25%, Incomplete = 0.36%	2.56%	0.00%	Failed = 100.0%, Incomplete = 0.0%	0.00%
	plansformer-gr-bw[2000]	58.33%	Failed = 40.53%, Incomplete = 1.14%	51.89%	0.00%	Failed = 0.14%, Incomplete = 99.86%	0.00%	0.81%	Failed = 98.36%, Incomplete = 0.83%	0.42%	0.00%	Failed = 100.0%, Incomplete = 0.0%	0.00%
	plansformer-gr-bw[5000]	79.00%	Failed = 20.89%, Incomplete = 0.11%	69.00%	0.00%	Failed = 0.03%, Incomplete = 99.97%	0.00%	0.00%	Failed = 100.0%, Incomplete = 0.0%	0.00%	0.00%	Failed = 100.0%, Incomplete = 0.0%	0.00%
	plansformer-gr-bw[10000]	93.89%	Failed = 6.11%, Incomplete = 0.0%	91.39%	0.00%	Failed = 0.0%, Incomplete = 100.0%	0.00%	0.00%	Failed = 100.0%, Incomplete = 0.0%	0.00%	0.00%	Failed = 100.0%, Incomplete = 0.0%	0.00%
	plansformer-gr-bw[14400]	95.94%	Failed = 4.06%, Incomplete = 0.0%	93.72%	0.00%	Failed = 0.0%, Incomplete = 100.0%	0.00%	0.00%	Failed = 100.0%, Incomplete = 0.0%	0.00%	0.00%	Failed = 100.0%, Incomplete = 0.0%	0.00%
	plansformer-gr-hn[500]	0.00%	Failed = 0.0%, Incomplete = 100.0%	0.00%	3.61%	Failed = 96.33%, Incomplete = 0.06%	3.39%	3.50%	Failed = 96.03%, Incomplete = 0.47%	1.39%	0.00%	Failed = 100.0%, Incomplete = 0.0%	0.00%
	plansformer-gr-hn[1000]	0.00%	Failed = 0.0%, Incomplete = 100.0%	0.00%	9.33%	Failed = 90.64%, Incomplete = 0.03%	8.81%	3.36%	Failed = 96.06%, Incomplete = 0.58%	2.06%	0.00%	Failed = 100.0%, Incomplete = 0.0%	0.00%
	plansformer-gr-hn[1500]	0.00%	Failed = 0.0%, Incomplete = 100.0%	0.00%	24.78%	Failed = 75.19%, Incomplete = 0.03%	23.94%	2.92%	Failed = 96.22%, Incomplete = 0.86%	1.03%	0.00%	Failed = 100.0%, Incomplete = 0.0%	0.00%
	plansformer-gr-hn[2000]	0.00%	Failed = 0.03%, Incomplete = 99.97%	0.00%	34.64%	Failed = 65.36%, Incomplete = 0.0%	34.08%	1.58%	Failed = 97.58%, Incomplete = 0.83%	1.06%	0.00%	Failed = 100.0%, Incomplete = 0.0%	0.00%
	plansformer-gr-hn[5000]	0.00%	Failed = 0.03%, Incomplete = 99.97%	0.00%	71.36%	Failed = 28.56%, Incomplete = 0.08%	68.83%	0.00%	Failed = 97.89%, Incomplete = 2.11%	0.00%	0.00%	Failed = 100.0%, Incomplete = 0.0%	0.00%
	plansformer-gr-hn[10000]	0.00%	Failed = 0.75%, Incomplete = 99.25%	0.00%	88.47%	Failed = 11.53%, Incomplete = 0.0%	86.56%	0.00%	Failed = 63.72%, Incomplete = 36.28%	0.00%	0.00%	Failed = 100.0%, Incomplete = 0.0%	0.00%

	plansformer-gr-hn[14400]	0.00%	Failed = 0.08%, Incomplete = 99.92%	0.00%	91.06%	Failed = 8.94%, Incomplete = 0.0%	89.86%	0.00%	Failed = 6.0%, Incomplete = 94.0%	0.00%	0.00%	Failed = 100.0%, Incomplete = 0.0%	0.00%
	plansformer-gr-dl[500]	0.00%	Failed = 0.06%, Incomplete = 99.94%	0.00%	0.00%	Failed = 0.11%, Incomplete = 99.89%	0.00%	74.72%	Failed = 25.28%, Incomplete = 0.0%	43.36%	31.08%	Failed = 64.28%, Incomplete = 4.64%	18.44%
	plansformer-gr-dl[1000]	0.00%	Failed = 0.39%, Incomplete = 99.61%	0.00%	0.00%	Failed = 0.0%, Incomplete = 100.0%	0.00%	74.75%	Failed = 25.22%, Incomplete = 0.03%	40.14%	50.81%	Failed = 48.92%, Incomplete = 0.28%	37.72%
	plansformer-gr-dl[1500]	0.00%	Failed = 0.06%, Incomplete = 99.94%	0.00%	0.00%	Failed = 0.03%, Incomplete = 99.97%	0.00%	66.31%	Failed = 33.67%, Incomplete = 0.03%	31.39%	61.42%	Failed = 38.47%, Incomplete = 0.11%	40.19%
	plansformer-gr-dl[2000]	0.00%	Failed = 0.06%, Incomplete = 99.94%	0.00%	0.00%	Failed = 0.06%, Incomplete = 99.94%	0.00%	53.69%	Failed = 43.14%, Incomplete = 3.17%	24.25%	65.03%	Failed = 34.64%, Incomplete = 0.33%	45.86%
	plansformer-gr-dl[5000]	0.00%	Failed = 0.08%, Incomplete = 99.92%	0.00%	0.00%	Failed = 0.03%, Incomplete = 99.97%	0.00%	2.58%	Failed = 96.0%, Incomplete = 1.42%	1.14%	76.36%	Failed = 11.08%, Incomplete = 12.56%	60.72%
	plansformer-gr-dl[10000]	0.00%	Failed = 1.03%, Incomplete = 98.97%	0.00%	0.00%	Failed = 1.03%, Incomplete = 98.97%	0.00%	0.00%	Failed = 99.94%, Incomplete = 0.06%	0.00%	90.44%	Failed = 9.31%, Incomplete = 0.25%	74.06%
	plansformer-gr-dl[14400]	0.00%	Failed = 0.86%, Incomplete = 99.14%	0.00%	0.00%	Failed = 3.22%, Incomplete = 96.78%	0.00%	0.00%	Failed = 98.5%, Incomplete = 1.5%	0.00%	91.97%	Failed = 7.94%, Incomplete = 0.08%	76.14%
	plansformer-dl-bw[500]	20.83%	Failed = 79.17%, Incomplete = 0.0%	15.89%	0.00%	Failed = 0.0%, Incomplete = 100.0%	0.00%	0.00%	Failed = 67.89%, Incomplete = 32.11%	0.00%	0.00%	Failed = 100.0%, Incomplete = 0.0%	0.00%
	plansformer-dl-bw[1000]	40.36%	Failed = 59.36%, Incomplete = 0.28%	32.50%	0.00%	Failed = 0.0%, Incomplete = 100.0%	0.00%	0.00%	Failed = 98.36%, Incomplete = 1.64%	0.00%	0.03%	Failed = 99.53%, Incomplete = 0.44%	0.00%
	plansformer-dl-bw[1500]	39.86%	Failed = 60.08%, Incomplete = 0.06%	32.47%	0.00%	Failed = 0.0%, Incomplete = 100.0%	0.00%	0.00%	Failed = 100.0%, Incomplete = 0.0%	0.00%	0.00%	Failed = 100.0%, Incomplete = 0.0%	0.00%
	plansformer-dl-bw[2000]	68.00%	Failed = 31.22%, Incomplete = 0.78%	62.42%	0.00%	Failed = 0.0%, Incomplete = 100.0%	0.00%	0.00%	Failed = 100.0%, Incomplete = 0.0%	0.00%	0.00%	Failed = 100.0%, Incomplete = 0.0%	0.00%
	plansformer-dl-bw[5000]	90.97%	Failed = 9.03%, Incomplete = 0.0%	82.69%	0.00%	Failed = 0.47%, Incomplete = 99.53%	0.00%	0.00%	Failed = 100.0%, Incomplete = 0.0%	0.00%	0.00%	Failed = 100.0%, Incomplete = 0.0%	0.00%
	plansformer-dl-bw[10000]	90.17%	Failed = 9.81%, Incomplete = 0.03%	88.22%	0.00%	Failed = 0.06%, Incomplete = 99.94%	0.00%	0.00%	Failed = 100.0%, Incomplete = 0.0%	0.00%	0.00%	Failed = 100.0%, Incomplete = 0.0%	0.00%
	plansformer-dl-bw[14400]	93.19%	Failed = 6.81%, Incomplete = 0.0%	91.28%	0.00%	Failed = 0.0%, Incomplete = 100.0%	0.00%	0.00%	Failed = 100.0%, Incomplete = 0.0%	0.00%	0.00%	Failed = 100.0%, Incomplete = 0.0%	0.00%

Further fine-tuned models	plansformer-dl-hn[500]	0.00%	Failed = 1.31%, Incomplete = 98.69%	0.00%	1.00%	Failed = 98.94%, Incomplete = 0.06%	0.61%	0.00%	Failed = 99.89%, Incomplete = 0.11%	0.00%	33.25%	Failed = 48.72%, Incomplete = 18.03%	28.97%
	plansformer-dl-hn[1000]	0.00%	Failed = 1.69%, Incomplete = 98.31%	0.00%	3.31%	Failed = 96.69%, Incomplete = 0.0%	3.28%	0.00%	Failed = 3.83%, Incomplete = 96.17%	0.00%	10.28%	Failed = 49.78%, Incomplete = 39.94%	9.39%
	plansformer-dl-hn[1500]	0.00%	Failed = 0.0%, Incomplete = 100.0%	0.00%	16.28%	Failed = 83.61%, Incomplete = 0.11%	15.69%	0.00%	Failed = 0.03%, Incomplete = 99.97%	0.00%	11.72%	Failed = 41.44%, Incomplete = 46.83%	11.03%
	plansformer-dl-hn[2000]	0.00%	Failed = 0.17%, Incomplete = 99.83%	0.00%	13.03%	Failed = 86.69%, Incomplete = 0.28%	12.83%	0.00%	Failed = 4.97%, Incomplete = 95.03%	0.00%	5.50%	Failed = 54.81%, Incomplete = 39.69%	5.17%
	plansformer-dl-hn[5000]	0.00%	Failed = 2.19%, Incomplete = 97.81%	0.00%	42.97%	Failed = 56.97%, Incomplete = 0.06%	38.75%	0.00%	Failed = 2.69%, Incomplete = 97.31%	0.00%	0.00%	Failed = 99.22%, Incomplete = 0.78%	0.00%
	plansformer-dl-hn[10000]	0.00%	Failed = 1.28%, Incomplete = 98.72%	0.00%	89.67%	Failed = 10.31%, Incomplete = 0.03%	87.72%	0.00%	Failed = 0.22%, Incomplete = 99.78%	0.00%	0.00%	Failed = 100.0%, Incomplete = 0.0%	0.00%
	plansformer-dl-hn[14400]	0.00%	Failed = 0.53%, Incomplete = 99.47%	0.00%	81.86%	Failed = 18.03%, Incomplete = 0.11%	79.81%	0.00%	Failed = 1.06%, Incomplete = 98.94%	0.00%	0.00%	Failed = 100.0%, Incomplete = 0.0%	0.00%
	plansformer-dl-gr[500]	0.00%	Failed = 3.5%, Incomplete = 96.5%	0.00%	0.00%	Failed = 0.0%, Incomplete = 100.0%	0.00%	18.39%	Failed = 81.36%, Incomplete = 0.25%	14.25%	29.11%	Failed = 39.14%, Incomplete = 31.75%	24.89%
	plansformer-dl-gr[1000]	0.00%	Failed = 0.0%, Incomplete = 100.0%	0.00%	0.00%	Failed = 0.0%, Incomplete = 100.0%	0.00%	48.72%	Failed = 51.22%, Incomplete = 0.06%	27.56%	45.92%	Failed = 45.03%, Incomplete = 9.06%	38.14%
	plansformer-dl-gr[1500]	0.00%	Failed = 0.22%, Incomplete = 99.78%	0.00%	0.00%	Failed = 0.0%, Incomplete = 100.0%	0.00%	61.47%	Failed = 37.56%, Incomplete = 0.97%	34.42%	20.22%	Failed = 68.03%, Incomplete = 11.75%	15.97%
	plansformer-dl-gr[2000]	0.00%	Failed = 1.72%, Incomplete = 98.28%	0.00%	0.00%	Failed = 0.0%, Incomplete = 100.0%	0.00%	62.08%	Failed = 37.83%, Incomplete = 0.08%	33.69%	4.50%	Failed = 91.81%, Incomplete = 3.69%	3.67%
	plansformer-dl-gr[5000]	0.00%	Failed = 0.0%, Incomplete = 100.0%	0.00%	0.00%	Failed = 0.0%, Incomplete = 100.0%	0.00%	77.72%	Failed = 22.25%, Incomplete = 0.03%	42.44%	0.00%	Failed = 99.97%, Incomplete = 0.03%	0.00%
	plansformer-dl-gr[10000]	0.00%	Failed = 0.0%, Incomplete = 100.0%	0.00%	0.00%	Failed = 0.0%, Incomplete = 100.0%	0.00%	85.00%	Failed = 15.0%, Incomplete = 0.0%	51.42%	0.00%	Failed = 100.0%, Incomplete = 0.0%	0.00%
	plansformer-dl-gr[14400]	0.00%	Failed = 0.06%, Incomplete = 99.94%	0.00%	0.00%	Failed = 0.0%, Incomplete = 100.0%	0.00%	85.47%	Failed = 14.5%, Incomplete = 0.03%	51.47%	0.00%	Failed = 100.0%, Incomplete = 0.0%	0.00%

Test Domain	Valid Plans
bw-test + Randomized bw-[Prompt1 & Prompt2]-test	99.06%
bw-test	97.77%
Randomized bw-Prompt1-test	98.41%
Randomized bw-Prompt2-test	96.25%

Table 7: Results obtained by Plansformer trained on augmented blocksworld data and tested using various sets

Test Domain	Valid Plans	Cost-Optimal Plans	Invalid Plans
bw	95.83%	93.75%	Failed = 4.17%, Incomplete = 0%
hn	79.25%	76.72%	Failed = 2.34%, Incomplete = 18.41%
gr	78.44%	50.61%	Failed = 15.03%, Incomplete = 6.53%
dl	63.03%	55.25%	Failed = 2.81%, Incomplete = 34.16%

Table 8: Plan Validation results obtained by a single Plansformer model (trained on all domains) and tested on individual test sets of the domains

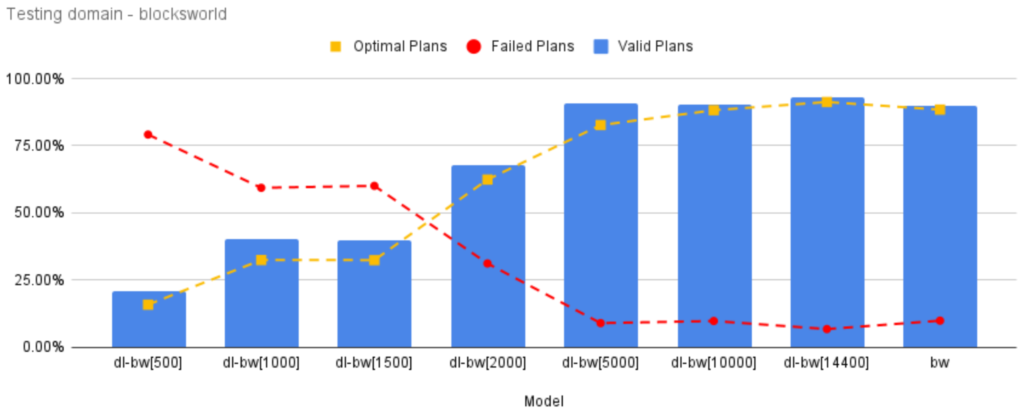


Figure 20: Plansformer-dl variants performance on blocksworld at various stages of fine-tuning

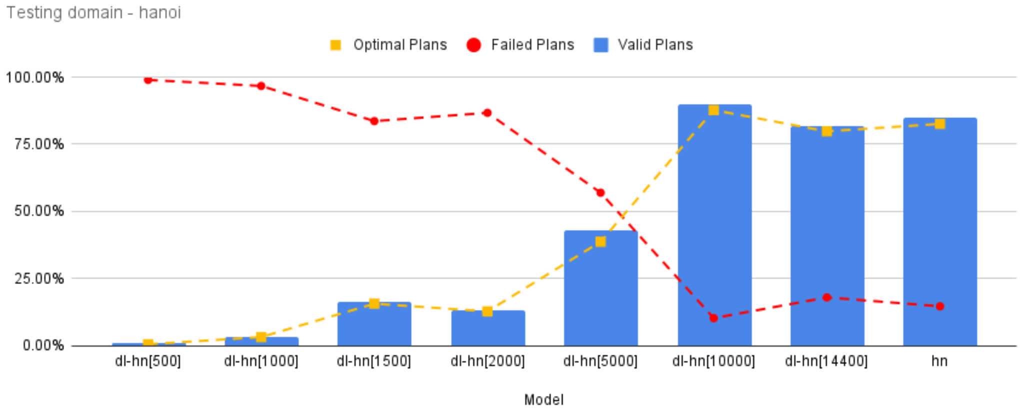


Figure 21: Plansformer-dl variants performance on hanoi at various stages of fine-tuning

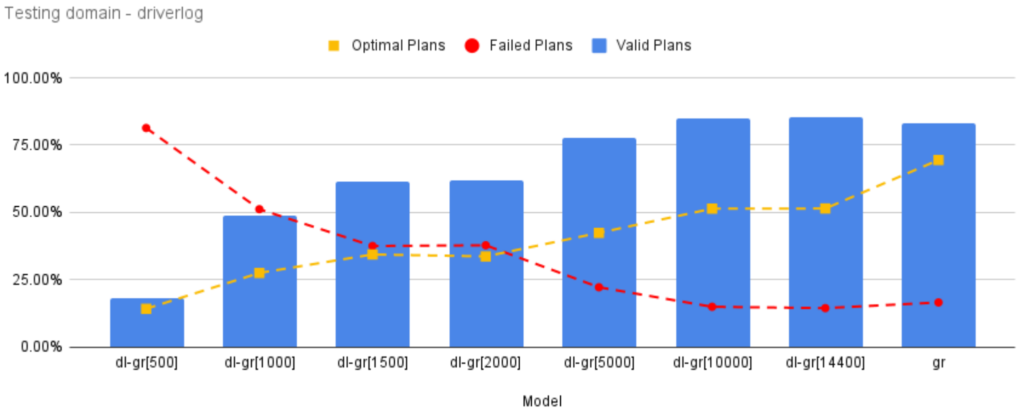


Figure 22: Plansformer-dl variants performance on grippers at various stages of fine-tuning

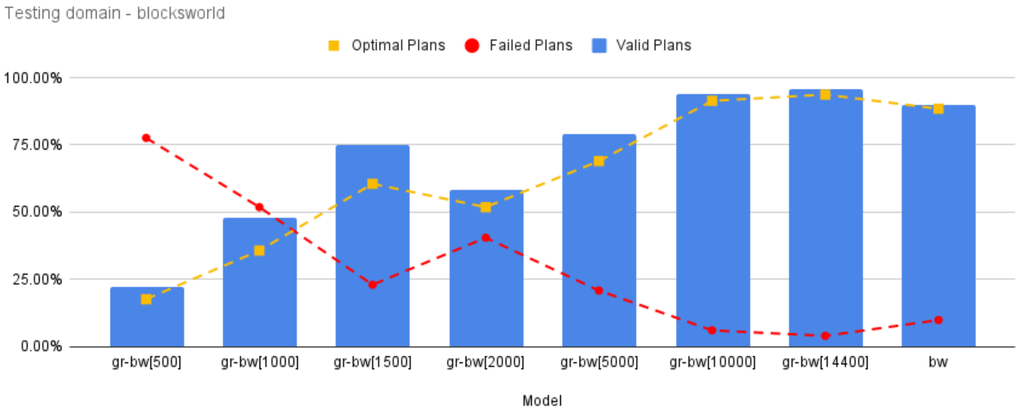


Figure 23: Plansformer-gr variants performance on blocksworld at various stages of fine-tuning

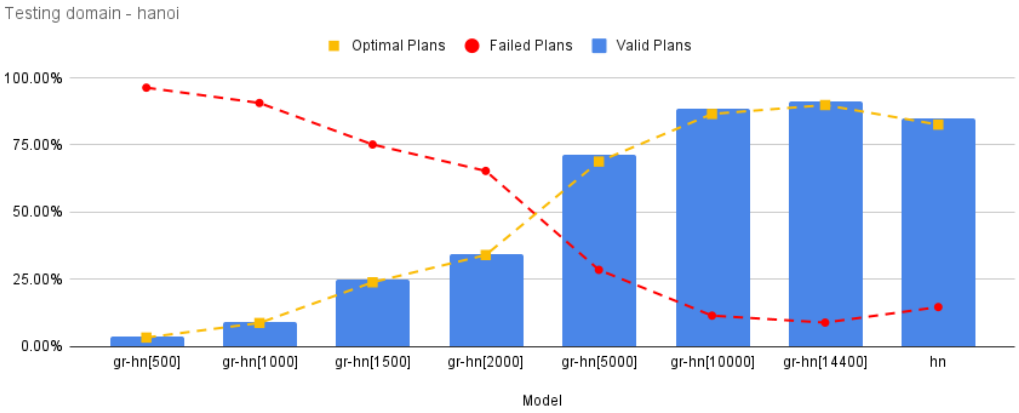


Figure 24: Plansformer-gr variants performance on hanoi at various stages of fine-tuning

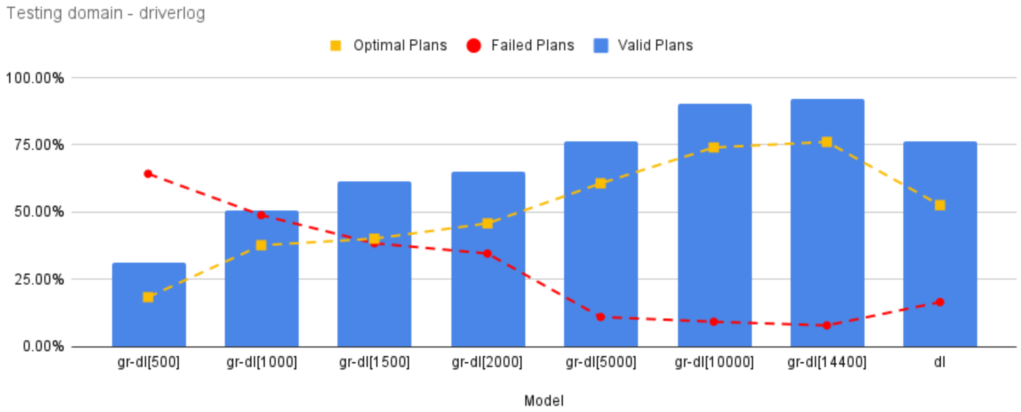


Figure 25: Plansformer-gr variants performance on driverlog at various stages of fine-tuning

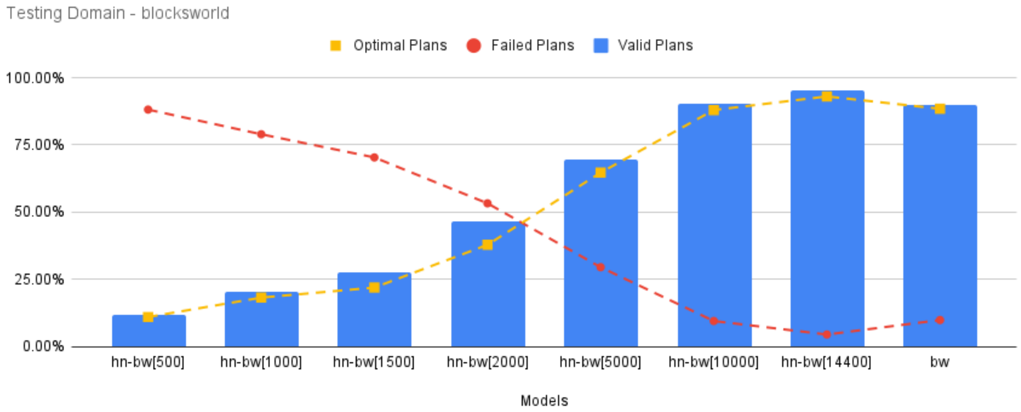


Figure 26: Plansformer-hn variants performance on blocksworld at various stages of fine-tuning

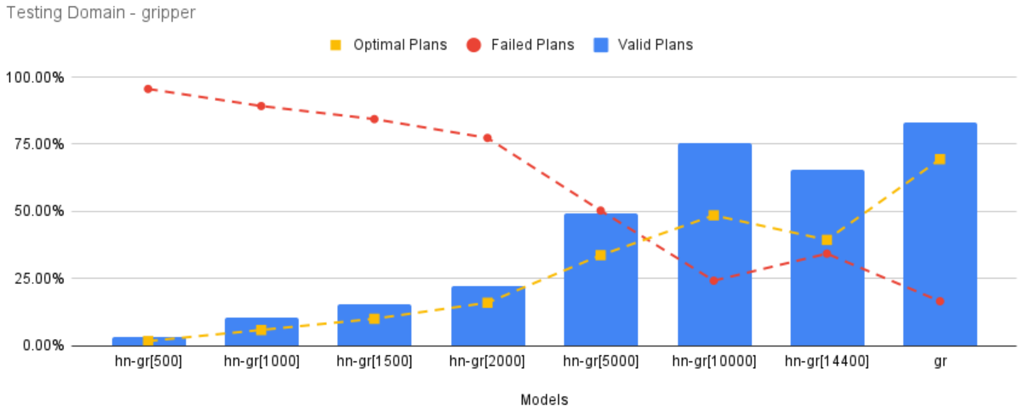


Figure 27: Plansformer-hn variants performance on gripper at various stages of fine-tuning

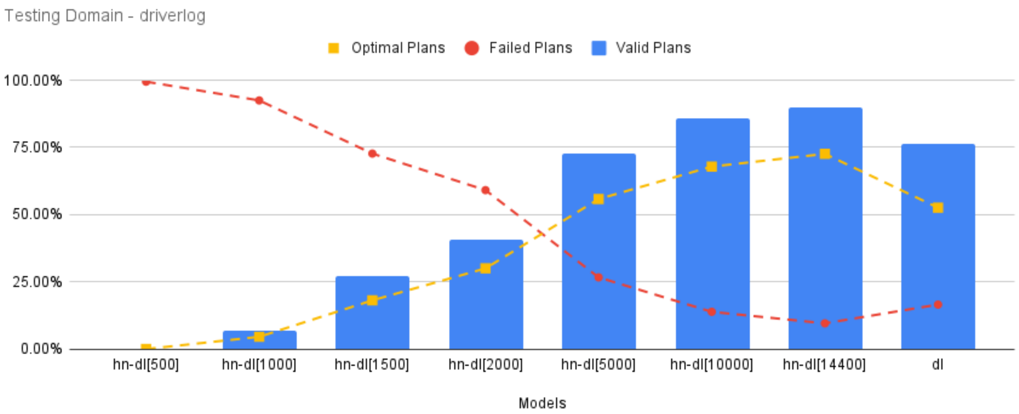


Figure 28: Plansformer-dl variants performance on driverlog at various stages of fine-tuning

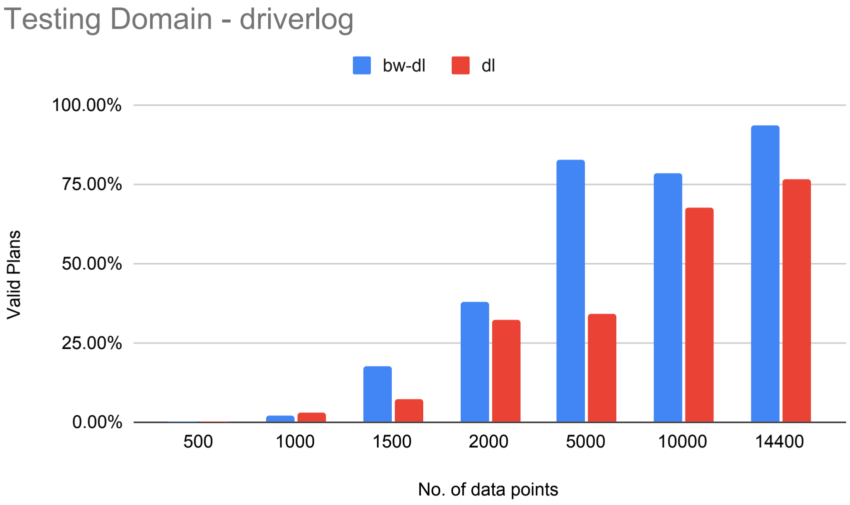


Figure 29: Comparison of valid plans generated by Plansformer-bw-dl derived models with Plansformer-dl trained using similar data points.

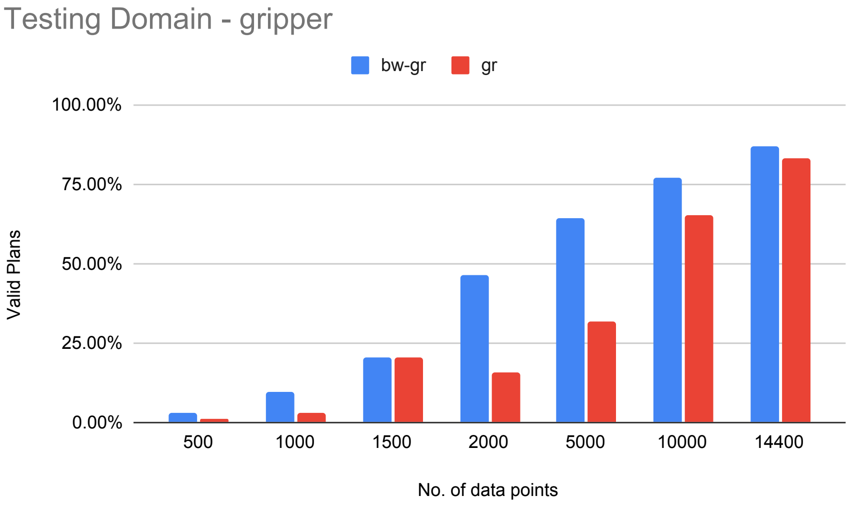


Figure 30: Comparison of valid plans generated by Plansformer-bw-gr derived models with Plansformer-gr trained using similar data points.

```
T5ForConditionalGeneration(
  (shared): Embedding(32100, 768)
  (encoder): T5Stack(
    (embed_tokens): Embedding(32100, 768)
    (block): ModuleList(
      (0): T5Block(
        (layer): ModuleList(
          (0): T5LayerSelfAttention(
            (SelfAttention): T5Attention(
              (q): Linear(in_features=768, out_features=768, bias=False)
              (k): Linear(in_features=768, out_features=768, bias=False)
              (v): Linear(in_features=768, out_features=768, bias=False)
              (o): Linear(in_features=768, out_features=768, bias=False)
              (relative_attention_bias): Embedding(32, 12)
            )
            (layer_norm): T5LayerNorm()
            (dropout): Dropout(p=0.1, inplace=False)
          )
          (1): T5LayerFF(
            (DenseReluDense): T5DenseActDense(
              (wi): Linear(in_features=768, out_features=3072, bias=False)
              (wo): Linear(in_features=3072, out_features=768, bias=False)
              (dropout): Dropout(p=0.1, inplace=False)
              (act): ReLU()
            )
            (layer_norm): T5LayerNorm()
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
      (1): T5Block(
        (layer): ModuleList(
          (0): T5LayerSelfAttention(
            (SelfAttention): T5Attention(
              (q): Linear(in_features=768, out_features=768, bias=False)
              (k): Linear(in_features=768, out_features=768, bias=False)
              (v): Linear(in_features=768, out_features=768, bias=False)
              (o): Linear(in_features=768, out_features=768, bias=False)
            )
            (layer_norm): T5LayerNorm()
            (dropout): Dropout(p=0.1, inplace=False)
          )
          (1): T5LayerFF(
            (DenseReluDense): T5DenseActDense(
              (wi): Linear(in_features=768, out_features=3072, bias=False)
              (wo): Linear(in_features=3072, out_features=768, bias=False)
              (dropout): Dropout(p=0.1, inplace=False)
              (act): ReLU()
            )
            (layer_norm): T5LayerNorm()
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
      (2): T5Block(
        (layer): ModuleList(
          (0): T5LayerSelfAttention(
            (SelfAttention): T5Attention(
              (q): Linear(in_features=768, out_features=768, bias=False)
              (k): Linear(in_features=768, out_features=768, bias=False)
              (v): Linear(in_features=768, out_features=768, bias=False)
              (o): Linear(in_features=768, out_features=768, bias=False)
```

Figure 31: Plansformer Layers

```
      )
      (layer_norm): T5LayerNorm()
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (1): T5LayerFF(
      (DenseReluDense): T5DenseActDense(
        (wi): Linear(in_features=768, out_features=3072, bias=False)
        (wo): Linear(in_features=3072, out_features=768, bias=False)
        (dropout): Dropout(p=0.1, inplace=False)
        (act): ReLU()
      )
      (layer_norm): T5LayerNorm()
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
)
(3): T5Block(
  (layer): ModuleList(
    (0): T5LayerSelfAttention(
      (SelfAttention): T5Attention(
        (q): Linear(in_features=768, out_features=768, bias=False)
        (k): Linear(in_features=768, out_features=768, bias=False)
        (v): Linear(in_features=768, out_features=768, bias=False)
        (o): Linear(in_features=768, out_features=768, bias=False)
      )
      (layer_norm): T5LayerNorm()
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (1): T5LayerFF(
      (DenseReluDense): T5DenseActDense(
        (wi): Linear(in_features=768, out_features=3072, bias=False)
        (wo): Linear(in_features=3072, out_features=768, bias=False)
        (dropout): Dropout(p=0.1, inplace=False)
        (act): ReLU()
      )
      (layer_norm): T5LayerNorm()
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
)
(4): T5Block(
  (layer): ModuleList(
    (0): T5LayerSelfAttention(
      (SelfAttention): T5Attention(
        (q): Linear(in_features=768, out_features=768, bias=False)
        (k): Linear(in_features=768, out_features=768, bias=False)
        (v): Linear(in_features=768, out_features=768, bias=False)
        (o): Linear(in_features=768, out_features=768, bias=False)
      )
      (layer_norm): T5LayerNorm()
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (1): T5LayerFF(
      (DenseReluDense): T5DenseActDense(
        (wi): Linear(in_features=768, out_features=3072, bias=False)
        (wo): Linear(in_features=3072, out_features=768, bias=False)
        (dropout): Dropout(p=0.1, inplace=False)
        (act): ReLU()
      )
      (layer_norm): T5LayerNorm()
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
)
(5): T5Block(
  (layer): ModuleList(
    (0): T5LayerSelfAttention(
      (SelfAttention): T5Attention(
        (q): Linear(in_features=768, out_features=768, bias=False)
        (k): Linear(in_features=768, out_features=768, bias=False)
        (v): Linear(in_features=768, out_features=768, bias=False)
        (o): Linear(in_features=768, out_features=768, bias=False)
      )
      (layer_norm): T5LayerNorm()
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (1): T5LayerFF(
      (DenseReluDense): T5DenseActDense(
        (wi): Linear(in_features=768, out_features=3072, bias=False)
        (wo): Linear(in_features=3072, out_features=768, bias=False)
        (dropout): Dropout(p=0.1, inplace=False)
        (act): ReLU()
      )
      (layer_norm): T5LayerNorm()
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
)
(6): T5Block(
  (layer): ModuleList(
    (0): T5LayerSelfAttention(
      (SelfAttention): T5Attention(
        (q): Linear(in_features=768, out_features=768, bias=False)
        (k): Linear(in_features=768, out_features=768, bias=False)
        (v): Linear(in_features=768, out_features=768, bias=False)
        (o): Linear(in_features=768, out_features=768, bias=False)
      )
      (layer_norm): T5LayerNorm()
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (1): T5LayerFF(
      (DenseReluDense): T5DenseActDense(
        (wi): Linear(in_features=768, out_features=3072, bias=False)
        (wo): Linear(in_features=3072, out_features=768, bias=False)
        (dropout): Dropout(p=0.1, inplace=False)
        (act): ReLU()
      )
      (layer_norm): T5LayerNorm()
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
)
(7): T5Block(
  (layer): ModuleList(
    (0): T5LayerSelfAttention(
      (SelfAttention): T5Attention(
        (q): Linear(in_features=768, out_features=768, bias=False)
        (k): Linear(in_features=768, out_features=768, bias=False)
        (v): Linear(in_features=768, out_features=768, bias=False)
        (o): Linear(in_features=768, out_features=768, bias=False)
      )
      (layer_norm): T5LayerNorm()
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (1): T5LayerFF(
      (DenseReluDense): T5DenseActDense(
        (wi): Linear(in_features=768, out_features=3072, bias=False)
        (wo): Linear(in_features=3072, out_features=768, bias=False)
        (dropout): Dropout(p=0.1, inplace=False)
        (act): ReLU()
      )
      (layer_norm): T5LayerNorm()
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
)
(8): T5Block(
  (layer): ModuleList(
    (0): T5LayerSelfAttention(
      (SelfAttention): T5Attention(
        (q): Linear(in_features=768, out_features=768, bias=False)
        (k): Linear(in_features=768, out_features=768, bias=False)
        (v): Linear(in_features=768, out_features=768, bias=False)
        (o): Linear(in_features=768, out_features=768, bias=False)
      )
      (layer_norm): T5LayerNorm()
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (1): T5LayerFF(
      (DenseReluDense): T5DenseActDense(
        (wi): Linear(in_features=768, out_features=3072, bias=False)
        (wo): Linear(in_features=3072, out_features=768, bias=False)
        (dropout): Dropout(p=0.1, inplace=False)
        (act): ReLU()
      )
      (layer_norm): T5LayerNorm()
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
)
(9): T5Block(
  (layer): ModuleList(
    (0): T5LayerSelfAttention(
      (SelfAttention): T5Attention(
        (q): Linear(in_features=768, out_features=768, bias=False)
        (k): Linear(in_features=768, out_features=768, bias=False)
        (v): Linear(in_features=768, out_features=768, bias=False)
        (o): Linear(in_features=768, out_features=768, bias=False)
      )
      (layer_norm): T5LayerNorm()
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (1): T5LayerFF(
      (DenseReluDense): T5DenseActDense(
        (wi): Linear(in_features=768, out_features=3072, bias=False)
        (wo): Linear(in_features=3072, out_features=768, bias=False)
        (dropout): Dropout(p=0.1, inplace=False)
        (act): ReLU()
      )
      (layer_norm): T5LayerNorm()
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
)
(10): T5Block(
  (layer): ModuleList(
        (0): T5LayerSelfAttention(
          (SelfAttention): T5Attention(
            (q): Linear(in_features=768, out_features=768, bias=False)
            (k): Linear(in_features=768, out_features=768, bias=False)
            (v): Linear(in_features=768, out_features=768, bias=False)
            (o): Linear(in_features=768, out_features=768, bias=False)
          )
          (layer_norm): T5LayerNorm()
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (1): T5LayerFF(
          (DenseReluDense): T5DenseActDense(
            (wi): Linear(in_features=768, out_features=3072, bias=False)
            (wo): Linear(in_features=3072, out_features=768, bias=False)
            (dropout): Dropout(p=0.1, inplace=False)
            (act): ReLU()
          )
          (layer_norm): T5LayerNorm()
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
    )
    (11): T5Block(
      (layer): ModuleList(
        (0): T5LayerSelfAttention(
          (SelfAttention): T5Attention(
            (q): Linear(in_features=768, out_features=768, bias=False)
            (k): Linear(in_features=768, out_features=768, bias=False)
            (v): Linear(in_features=768, out_features=768, bias=False)
            (o): Linear(in_features=768, out_features=768, bias=False)
          )
          (layer_norm): T5LayerNorm()
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (1): T5LayerFF(
          (DenseReluDense): T5DenseActDense(
            (wi): Linear(in_features=768, out_features=3072, bias=False)
            (wo): Linear(in_features=3072, out_features=768, bias=False)
            (dropout): Dropout(p=0.1, inplace=False)
            (act): ReLU()
          )
          (layer_norm): T5LayerNorm()
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
    )
  )
  (final_layer_norm): T5LayerNorm()
  (dropout): Dropout(p=0.1, inplace=False)
)
(decoder): T5Stack(
  (embed_tokens): Embedding(32100, 768)
  (block): ModuleList(
    (0): T5Block(
      (layer): ModuleList(
        (0): T5LayerSelfAttention(
          (SelfAttention): T5Attention(
            (q): Linear(in_features=768, out_features=768, bias=False)
            (k): Linear(in_features=768, out_features=768, bias=False)
            (v): Linear(in_features=768, out_features=768, bias=False)
            (o): Linear(in_features=768, out_features=768, bias=False)
            (relative_attention_bias): Embedding(32, 12)
          )
          (layer_norm): T5LayerNorm()
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (1): T5LayerCrossAttention(
          (EncDecAttention): T5Attention(
            (q): Linear(in_features=768, out_features=768, bias=False)
            (k): Linear(in_features=768, out_features=768, bias=False)
            (v): Linear(in_features=768, out_features=768, bias=False)
            (o): Linear(in_features=768, out_features=768, bias=False)
          )
          (layer_norm): T5LayerNorm()
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (2): T5LayerFF(
          (DenseReluDense): T5DenseActDense(
            (wi): Linear(in_features=768, out_features=3072, bias=False)
            (wo): Linear(in_features=3072, out_features=768, bias=False)
            (dropout): Dropout(p=0.1, inplace=False)
            (act): ReLU()
          )
          (layer_norm): T5LayerNorm()
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
    )
    (1): T5Block(
      (layer): ModuleList(
        (0): T5LayerSelfAttention(
          (SelfAttention): T5Attention(
            (q): Linear(in_features=768, out_features=768, bias=False)
            (k): Linear(in_features=768, out_features=768, bias=False)
            (v): Linear(in_features=768, out_features=768, bias=False)
            (o): Linear(in_features=768, out_features=768, bias=False)
          )
          (layer_norm): T5LayerNorm()
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (1): T5LayerCrossAttention(
          (EncDecAttention): T5Attention(
            (q): Linear(in_features=768, out_features=768, bias=False)
            (k): Linear(in_features=768, out_features=768, bias=False)
            (v): Linear(in_features=768, out_features=768, bias=False)
            (o): Linear(in_features=768, out_features=768, bias=False)
          )
          (layer_norm): T5LayerNorm()
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (2): T5LayerFF(
          (DenseReluDense): T5DenseActDense(
            (wi): Linear(in_features=768, out_features=3072, bias=False)
            (wo): Linear(in_features=3072, out_features=768, bias=False)
            (dropout): Dropout(p=0.1, inplace=False)
            (act): ReLU()
          )
          (layer_norm): T5LayerNorm()
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
    )
    (2): T5Block(
      (layer): ModuleList(
        (0): T5LayerSelfAttention(
          (SelfAttention): T5Attention(
            (q): Linear(in_features=768, out_features=768, bias=False)
            (k): Linear(in_features=768, out_features=768, bias=False)
            (v): Linear(in_features=768, out_features=768, bias=False)
            (o): Linear(in_features=768, out_features=768, bias=False)
          )
          (layer_norm): T5LayerNorm()
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (1): T5LayerCrossAttention(
          (EncDecAttention): T5Attention(
            (q): Linear(in_features=768, out_features=768, bias=False)
            (k): Linear(in_features=768, out_features=768, bias=False)
            (v): Linear(in_features=768, out_features=768, bias=False)
            (o): Linear(in_features=768, out_features=768, bias=False)
          )
          (layer_norm): T5LayerNorm()
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (2): T5LayerFF(
          (DenseReluDense): T5DenseActDense(
            (wi): Linear(in_features=768, out_features=3072, bias=False)
            (wo): Linear(in_features=3072, out_features=768, bias=False)
            (dropout): Dropout(p=0.1, inplace=False)
            (act): ReLU()
          )
          (layer_norm): T5LayerNorm()
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
    )
    (3): T5Block(
      (layer): ModuleList(
        (0): T5LayerSelfAttention(
          (SelfAttention): T5Attention(
            (q): Linear(in_features=768, out_features=768, bias=False)
            (k): Linear(in_features=768, out_features=768, bias=False)
            (v): Linear(in_features=768, out_features=768, bias=False)
            (o): Linear(in_features=768, out_features=768, bias=False)
          )
          (layer_norm): T5LayerNorm()
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (1): T5LayerCrossAttention(
          (EncDecAttention): T5Attention(
            (q): Linear(in_features=768, out_features=768, bias=False)
            (k): Linear(in_features=768, out_features=768, bias=False)
            (v): Linear(in_features=768, out_features=768, bias=False)
            (o): Linear(in_features=768, out_features=768, bias=False)
          )
          (layer_norm): T5LayerNorm()
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (2): T5LayerFF(
          (DenseReluDense): T5DenseActDense(
            (wi): Linear(in_features=768, out_features=3072, bias=False)
            (wo): Linear(in_features=3072, out_features=768, bias=False)
            (dropout): Dropout(p=0.1, inplace=False)
            (act): ReLU()
          )
          (layer_norm): T5LayerNorm()
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
    )
    (4): T5Block(
      (layer): ModuleList(
        (0): T5LayerSelfAttention(
          (SelfAttention): T5Attention(
            (q): Linear(in_features=768, out_features=768, bias=False)
            (k): Linear(in_features=768, out_features=768, bias=False)
            (v): Linear(in_features=768, out_features=768, bias=False)
            (o): Linear(in_features=768, out_features=768, bias=False)
          )
          (layer_norm): T5LayerNorm()
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (1): T5LayerCrossAttention(
          (EncDecAttention): T5Attention(
            (q): Linear(in_features=768, out_features=768, bias=False)
            (k): Linear(in_features=768, out_features=768, bias=False)
            (v): Linear(in_features=768, out_features=768, bias=False)
            (o): Linear(in_features=768, out_features=768, bias=False)
          )
          (layer_norm): T5LayerNorm()
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (2): T5LayerFF(
          (DenseReluDense): T5DenseActDense(
            (wi): Linear(in_features=768, out_features=3072, bias=False)
            (wo): Linear(in_features=3072, out_features=768, bias=False)
            (dropout): Dropout(p=0.1, inplace=False)
            (act): ReLU()
          )
          (layer_norm): T5LayerNorm()
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
    )
    (5): T5Block(
      (layer): ModuleList(
        (0): T5LayerSelfAttention(
          (SelfAttention): T5Attention(
            (q): Linear(in_features=768, out_features=768, bias=False)
            (k): Linear(in_features=768, out_features=768, bias=False)
            (v): Linear(in_features=768, out_features=768, bias=False)
            (o): Linear(in_features=768, out_features=768, bias=False)
          )
          (layer_norm): T5LayerNorm()
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (1): T5LayerCrossAttention(
          (EncDecAttention): T5Attention(
            (q): Linear(in_features=768, out_features=768, bias=False)
            (k): Linear(in_features=768, out_features=768, bias=False)
            (v): Linear(in_features=768, out_features=768, bias=False)
            (o): Linear(in_features=768, out_features=768, bias=False)
          )
          (layer_norm): T5LayerNorm()
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (2): T5LayerFF(
          (DenseReluDense): T5DenseActDense(
            (wi): Linear(in_features=768, out_features=3072, bias=False)
            (wo): Linear(in_features=3072, out_features=768, bias=False)
            (dropout): Dropout(p=0.1, inplace=False)
            (act): ReLU()
          )
          (layer_norm): T5LayerNorm()
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
    )
    (6): T5Block(
      (layer): ModuleList(
        (0): T5LayerSelfAttention(
          (SelfAttention): T5Attention(
            (q): Linear(in_features=768, out_features=768, bias=False)
            (k): Linear(in_features=768, out_features=768, bias=False)
            (v): Linear(in_features=768, out_features=768, bias=False)
            (o): Linear(in_features=768, out_features=768, bias=False)
          )
          (layer_norm): T5LayerNorm()
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (1): T5LayerCrossAttention(
          (EncDecAttention): T5Attention(
            (q): Linear(in_features=768, out_features=768, bias=False)
            (k): Linear(in_features=768, out_features=768, bias=False)
            (v): Linear(in_features=768, out_features=768, bias=False)
            (o): Linear(in_features=768, out_features=768, bias=False)
          )
          (layer_norm): T5LayerNorm()
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (2): T5LayerFF(
          (DenseReluDense): T5DenseActDense(
            (wi): Linear(in_features=768, out_features=3072, bias=False)
            (wo): Linear(in_features=3072, out_features=768, bias=False)
            (dropout): Dropout(p=0.1, inplace=False)
            (act): ReLU()
          )
          (layer_norm): T5LayerNorm()
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
    )
    (7): T5Block(
      (layer): ModuleList(
        (0): T5LayerSelfAttention(
          (SelfAttention): T5Attention(
            (q): Linear(in_features=768, out_features=768, bias=False)
            (k): Linear(in_features=768, out_features=768, bias=False)
            (v): Linear(in_features=768, out_features=768, bias=False)
            (o): Linear(in_features=768, out_features=768, bias=False)
          )
          (layer_norm): T5LayerNorm()
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (1): T5LayerCrossAttention(
          (EncDecAttention): T5Attention(
            (q): Linear(in_features=768, out_features=768, bias=False)
            (k): Linear(in_features=768, out_features=768, bias=False)
            (v): Linear(in_features=768, out_features=768, bias=False)
            (o): Linear(in_features=768, out_features=768, bias=False)
          )
          (layer_norm): T5LayerNorm()
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (2): T5LayerFF(
          (DenseReluDense): T5DenseActDense(
            (wi): Linear(in_features=768, out_features=3072, bias=False)
            (wo): Linear(in_features=3072, out_features=768, bias=False)
            (dropout): Dropout(p=0.1, inplace=False)
            (act): ReLU()
          )
          (layer_norm): T5LayerNorm()
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
    )
    (8): T5Block(
      (layer): ModuleList(
        (0): T5LayerSelfAttention(
          (SelfAttention): T5Attention(
            (q): Linear(in_features=768, out_features=768, bias=False)
            (k): Linear(in_features=768, out_features=768, bias=False)
            (v): Linear(in_features=768, out_features=768, bias=False)
            (o): Linear(in_features=768, out_features=768, bias=False)
          )
          (layer_norm): T5LayerNorm()
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (1): T5LayerCrossAttention(
          (EncDecAttention): T5Attention(
            (q): Linear(in_features=768, out_features=768, bias=False)
            (k): Linear(in_features=768, out_features=768, bias=False)
            (v): Linear(in_features=768, out_features=768, bias=False)
            (o): Linear(in_features=768, out_features=768, bias=False)
          )
          (layer_norm): T5LayerNorm()
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (2): T5LayerFF(
          (DenseReluDense): T5DenseActDense(
            (wi): Linear(in_features=768, out_features=3072, bias=False)
            (wo): Linear(in_features=3072, out_features=768, bias=False)
            (dropout): Dropout(p=0.1, inplace=False)
            (act): ReLU()
          )
          (layer_norm): T5LayerNorm()
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
    )
    (9): T5Block(
      (layer): ModuleList(
        (0): T5LayerSelfAttention(
          (SelfAttention): T5Attention(
            (q): Linear(in_features=768, out_features=768, bias=False)
            (k): Linear(in_features=768, out_features=768, bias=False)
            (v): Linear(in_features=768, out_features=768, bias=False)
            (o): Linear(in_features=768, out_features=768, bias=False)
          )
          (layer_norm): T5LayerNorm()
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (1): T5LayerCrossAttention(
          (EncDecAttention): T5Attention(
            (q): Linear(in_features=768, out_features=768, bias=False)
            (k): Linear(in_features=768, out_features=768, bias=False)
            (v): Linear(in_features=768, out_features=768, bias=False)
            (o): Linear(in_features=768, out_features=768, bias=False)
          )
          (layer_norm): T5LayerNorm()
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (2): T5LayerFF(
          (DenseReluDense): T5DenseActDense(
            (wi): Linear(in_features=768, out_features=3072, bias=False)
            (wo): Linear(in_features=3072, out_features=768, bias=False)
            (dropout): Dropout(p=0.1, inplace=False)
            (act): ReLU()
          )
          (layer_norm): T5LayerNorm()
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
    )
    (10): T5Block(
      (layer): ModuleList(
        (0): T5LayerSelfAttention(
          (SelfAttention): T5Attention(
            (q): Linear(in_features=768, out_features=768, bias=False)
            (k): Linear(in_features=768, out_features=768, bias=False)
            (v): Linear(in_features=768, out_features=768, bias=False)
            (o): Linear(in_features=768, out_features=768, bias=False)
          )
          (layer_norm): T5LayerNorm()
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (1): T5LayerCrossAttention(
          (EncDecAttention): T5Attention(
            (q): Linear(in_features=768, out_features=768, bias=False)
            (k): Linear(in_features=768, out_features=768, bias=False)
            (v): Linear(in_features=768, out_features=768, bias=False)
            (o): Linear(in_features=768, out_features=768, bias=False)
          )
          (layer_norm): T5LayerNorm()
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (2): T5LayerFF(
          (DenseReluDense): T5DenseActDense(
            (wi): Linear(in_features=768, out_features=3072, bias=False)
            (wo): Linear(in_features=3072, out_features=768, bias=False)
            (dropout): Dropout(p=0.1, inplace=False)
            (act): ReLU()
          )
          (layer_norm): T5LayerNorm()
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
    )
    (11): T5Block(
      (layer): ModuleList(
        (0): T5LayerSelfAttention(
          (SelfAttention): T5Attention(
            (q): Linear(in_features=768, out_features=768, bias=False)
            (k): Linear(in_features=768, out_features=768, bias=False)
```

```
              (v): Linear(in_features=768, out_features=768, bias=False)
              (o): Linear(in_features=768, out_features=768, bias=False)
            )
            (layer_norm): T5LayerNorm()
            (dropout): Dropout(p=0.1, inplace=False)
          )
          (1): T5LayerCrossAttention(
            (EncDecAttention): T5Attention(
              (q): Linear(in_features=768, out_features=768, bias=False)
              (k): Linear(in_features=768, out_features=768, bias=False)
              (v): Linear(in_features=768, out_features=768, bias=False)
              (o): Linear(in_features=768, out_features=768, bias=False)
            )
            (layer_norm): T5LayerNorm()
            (dropout): Dropout(p=0.1, inplace=False)
          )
          (2): T5LayerFF(
            (DenseReluDense): T5DenseActDense(
              (wi): Linear(in_features=768, out_features=3072, bias=False)
              (wo): Linear(in_features=3072, out_features=768, bias=False)
              (dropout): Dropout(p=0.1, inplace=False)
              (act): ReLU()
            )
            (layer_norm): T5LayerNorm()
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
    )
    (final_layer_norm): T5LayerNorm()
    (dropout): Dropout(p=0.1, inplace=False)
  )
  (lm_head): Linear(in_features=768, out_features=32100, bias=False)
)
```