Language Models For Generalised PDDL Planning: Synthesising Sound and Programmatic Policies

Dillon Z. Chen[1,2,3] **Johannes Zenn**[1] **Tristan Cinquin**[1] **Sheila A. McIlraith**[1,3]
[1]Vector Institute [2]LAAS-CNRS, University of Toulouse [3]University of Toronto

Abstract

We study the usage of language models (LMs) for planning over world models specified in the Planning Domain Definition Language (PDDL). We prompt LMs to generate Python programs that serve as generalised policies for solving PDDL problems from a given domain. Notably, our approach synthesises policies that are provably sound relative to the PDDL domain without reliance on external verifiers. We conduct experiments on competition benchmarks which show that our policies can solve more PDDL problems than PDDL planners and recent LM approaches within a fixed time and memory constraint. Our approach manifests in the LMPLAN planner which can solve planning problems with several hundreds of relevant objects. Surprisingly, we observe that LMs used in our framework sometimes plan more effectively over PDDL problems written in meaningless symbols in place of natural language; e.g. rewriting `(at dog kitchen)` as `(p2 o1 o3)`. This finding challenges hypotheses that LMs reason over word semantics and memorise solutions from its training corpus, and is worth further exploration.

1 Introduction

AI automated planning (AP) [GNT04, GB13] refers to the class of sequential decision-making problems formally specified by symbolic models in specification languages such as the Planning Domain Definition Language (PDDL) [MGH+98, HLMM19], and typically solved using heuristic search techniques. Although AP is intractable [Cha87, Byl94, ENS95], agents are often tasked with tractable families of planning problems—problems that share a common set of actions, transition system, and typed objects, and whose structural properties can be exploited to construct plans or policies that solve these families of problems. For example, interplanetary rovers must continuously explore and collect information about planets with minimal communication and time for activity planning [BJMR05], while logistics companies such as UPS™ ship and deliver packages and scale to over 20 million packages every day across 200 countries and territories in 2024 [UPS25]. In such applications, there is much need for *planning efficiently at scale* to solve time-intensive problems, and *automated plan synthesis* to minimise dependency on human intervention. Generalised planning (GP) exactly encapsulates this problem of automatically generating plans as programs that address families of related planning problems [Lev05, SIZ08, SIZ11, BPG09, HD11, BFG19, IM19, CSJ19, CLL21, FBG21, DSG22].

Recently, immense progress in the development of language models (LMs) [BMR+20, CND+23] has revealed strong emergent capabilities in reasoning and problem solving [KGR+22, WWS+22]. This progress has given rise to significant advances across various areas of artificial intelligence and problem solving, including algorithmic discovery [RBN+24, NVE+25], code generation [CTJ+21, NPH+23, LTY+24], competitive programming [Ope25], and embodied intelligence [DXS+23, IBB+25]. However, several studies have suggested that current LMs are incapable of solving long-horizon planning problems [VMSK23, VSK24].

18th European Workshop on Reinforcement Learning (EWRL 2025).

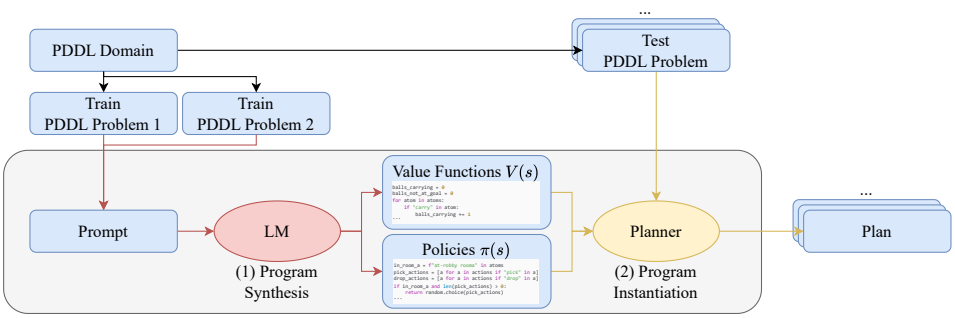


Figure 1: Pipeline for planning with LM-generated value functions and policies. The architecture is enclosed in gray. (1) A domain, 2 example PDDL problems, and a prompt is input into the LM which outputs a Python program representing a value function or policy. (2) The program is then used to help plan for PDDL problems from the same domain. See Section 3 for more details.

In this work, we leverage LMs to synthesise Python programs as solutions to GP problems. Similarly to prior work in LMs for GP [SDS+24], we encode GP problems in PDDL, the de facto specification for symbolic AP problems. We assume that the PDDL models are given but they can be synthesised with minimal human intervention from unstructured data such as natural language [CWF+22, LAM+23, GVSK23, OSK+24, HLC25, TZM25], images [XGT24, AKS+25], and environment interactions [VMS21, VMS22, SCK+23, SK23, LKT+25].

We study LM-generated programs as (1) value functions following [CPS25], and (2) policies for solving GP problems represented in PDDL. We use LMs to generate Python programs implementing value functions to guide heuristic search, and policies as reactive controllers that specify an action to take from a set of applicable actions in a given state. Notably, we guarantee that the synthesised policies are sound (any returned solution is correct in relation to the PDDL domain theory) by restricting them to predict actions that are only applicable at the input state, and furthermore to improve search performance. Our approach manifests in an LM planner that outperforms state-of-the-art planners on total number of problems solved within a given time and memory limit from the recent International Planning Competition Learning Track [TAE+24]. We further study the effect of symbolic and semantic representations of planning problems for LMs used in our framework. We observe that surprisingly and in contrast to observations in previous works [VMH+23, SDS+24], LMs used in our framework can sometimes match or even perform better on planning problems encoded using meaningless symbols (e.g., `o1` for `dog` or `p2` for `at`). This is a provocative finding worth further exploration since it could suggest that the LM has learned to do some form of symbolic planning or reasoning. Our contributions are summarised as follows.

- We use LMs to generate code implementing *sound* policies and compare its performance to generating value functions used in heuristic search for planning, with results generally favouring the former approach. We further use LM-generated policies to improve heuristic search performance.
- We conduct experiments demonstrating that LMs used in our framework can sometimes plan better on PDDL problems written with only symbols in place of natural language. This observation challenges hypotheses from previous works that LMs can only plan with languages with semantic meaning and memorise solutions from their training corpus.

2 Preliminaries: Classical Planning, PDDL, and Generalised Planning

In this section we introduce the necessary background on AP, followed by the problem we tackle in this paper: generalised planning. AP refers to the class of sequential decision-making problems using models represented in formal, symbolic languages and the methodologies for solving them. We begin by introducing the abstract classical planning problem model, a special case of AP problems where the objective is to find a sequence of actions that transitions an agent from an initial state to a goal state under a fully-observable, deterministic transition system. Although probabilistic extensions of planning exist [BT91, San10, MK12], we focus our attention on classical planning for ease of presentation. Following this, we provide an informal introduction to PDDL for encoding planning problems. We then introduce heuristic search as a state-of-the-art method for solving

Table 1: Analogies between RL and AP methodologies.

Methodology	Reinforcement Learning (RL)	AI Planning (AP)
Action space	Policy approximation	Policy synthesis (3.2.a)
State space	Value function approximation	Heuristic search (3.2.b)
State & action	Actor-critic algorithms	Heuristic search w. preferred operators (3.2.c)

planning problems and how they can be viewed analogously to value functions in reinforcement learning (RL). For those familiar with RL, Table 1 draws analogies between common RL and AP methodologies. We conclude this section by formalising the GP problem which is concerned with synthesising programmatic plans for solving families of related planning problems.

Planning problem A classical planning problem is a deterministic state transition model concerned with driving a given initial state into a goal state via a sequence of actions. Following the notation in [GB13], a *(classical) planning problem* is a tuple $P = \langle \mathcal{S}, \mathcal{A}, f, \mathcal{G}, s_I, c \rangle$ where $\mathcal{S}$ is a set of states, $\mathcal{A}$ is a set of actions, $f : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S} \cup \{\bot\}$ is a *deterministic transition function* where $f(s, a) = \bot$ represents that the action a is not applicable in the state s, $\mathcal{G} \subseteq \mathcal{S}$ is a non-empty set of goal states, $s_I \in \mathcal{S}$ is the initial state, and $c : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}_{\geq 0}$ is a cost function. The set of *applicable actions* of a state s is defined by $A(s) = \{a \mid f(s, a) \neq \bot\} \subseteq A$ and the *successors* of s is defined by $\{f(s, a) \mid a \in A(s)\} \subseteq S$. A solution or *plan* α for a planning problem is a finite sequence of actions $a_0, \ldots, a_n$ such that $f(s_i, a_i) = s_{i+1} \neq \bot$ for $i = 0, \ldots, n$ where $s_0 = s_I$ and $s_{n+1} \in \mathcal{G}$. In this paper, we focus on *satisficing planning* which refers to the problem of finding satisficing solutions for P, i.e. any plan for the problem.

Planning representations: PDDL Planning problems are often represented in a first-order formal language, the most common being PDDL. Details of the PDDL syntax are not necessary for the understanding of the paper but one should note that fragments of PDDL planning range from being EXPSPACE-hard [ENS95] to undecidable [Hel02]. This is because PDDL provides compact encodings of transition models that can be exponential in the size of the input files or greater. A planning problem is represented by a PDDL *domain*, providing a compact encoding of the transition function in terms of a set of object types, predicates, numeric functions, and actions, and a PDDL *problem*, specifying a finite set of objects, an initial state, and a goal condition. For example, a package delivery **domain** can contain

- the types `object`, `vehicle`, `location`, `package`,
- the predicates `(at ?o - object ?l - location)`, `(in ?p - package ?l - location)`,
- the numeric functions `(capacity ?v - vehicle)`, `(weight ?p - package)`, and
- the following action schema, among others, for loading a package into a vehicle

```
(:action pick-up
 :parameters (?v - vehicle ?l - location ?p - package)
 :precondition (and (at ?v ?l) (at ?p ?l) (>= (capacity ?v) (weight ?p)))
 :effect (and (not (at ?p ?l)) (in ?p ?v) (decrease (capacity ?v) (weight ?p))))
```

Next, a package delivery **problem** can consist of a truck `truck2 - vehicle` that is at the depot `(at truck2 depot)`, has a specified capacity `(= (capacity truck2) 5)`, and is carrying some packages `(in truck2 package3) & (in truck2 package1)`. A PDDL goal condition consists of a conjunction of ground atoms and inequalities of expressions of numeric functions, such as `(at package1 office) & (>= (capacity truck2) 7)`. A goal condition induces a set of goal states as a state that satisfies a goal condition is considered a goal state.

Heuristic search Heuristic search is the main driver of current state-of-the-art planners [RW10, SKH20], with roots dating back to early planning systems [FN71, LG95, McD96, BLG97]. A heuristic function is a function $h : \mathcal{S} \rightarrow \mathbb{R} \cup \{\infty\}$ which estimates the cost to go from a state s to a goal state in $\mathcal{G}$, and returns the value ∞ estimating that there is no plan from s to a goal. A heuristic is safe if $h(s) = \infty$ only if there is no plan from s to a goal. The optimal heuristic h^* returns the **minimal** plan cost from a state, if a plan exists, and ∞ otherwise. A heuristic function is analogous to an approximate value function in RL which estimates the **maximal** reward from a state where one views the cost of a plan as a negative reward. Thus, h^* is the optimal value function and induces a policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ that executes a successor state with the lowest value, breaking ties arbitrarily. However, h^* is intractable to compute so heuristics are instead used to guide search. The *Greedy*

Best First Search (GBFS) algorithm searches for a path over the graph induced by the transition system of a planning problem from an initial state to a goal state, guided by a heuristic function. It consists of a priority queue initialised with the initial state as the only element, and a main loop that performs the following steps while the queue is non-empty:

(1) pop a state s with the lowest heuristic value (breaking ties arbitrarily) from the queue,
(2) generate the successors of s via all applicable actions, and
(3) check if a successor s' is a goal, in which case terminate with the plan to s', and otherwise add s' to the queue if it has not been seen before.

The algorithm determines a problem is unsolvable if the main loop terminates which only occurs if there are finitely many states. GBFS is *sound* (any returned solution is correct) and *complete* (a solution is returned if it exists) for planning problems with finite state spaces. A* search [HNR68] is an optimal search algorithm when used with an admissible heuristic. We do not use it in our experiments because LM-generated heuristics are not guaranteed to be admissible.

Problem statement: generalised planning Recall that planning problems are specified by a PDDL domain and problem. Let $\mathcal{D}$ denote a PDDL domain, and P a problem associated with $\mathcal{D}$, where we say that the problem P *belongs* to $\mathcal{D}$. A *generalised planning (GP) problem* is a tuple $\langle \mathcal{P}_{train}, \mathcal{P}_{test} \rangle$ where $\mathcal{P}_{train}$ is a finite set of training problems and $\mathcal{P}_{test}$ a (possibly infinite) set of testing problems belonging to the same domain. A solution to a GP problem is a policy (here a program) that is synthesised from $\mathcal{P}_{train}$ and can be instantiated on and solve each problem $P \in \mathcal{P}_{test}$. A key attribute of GP problems is that problems in $\mathcal{P}_{test}$ are larger in terms of number of objects and more difficult than to solve than problems in $\mathcal{P}_{train}$. Thus, GP is as an out-of-distribution learning task.

3 LM-Generated Python Programs for Generalised Planning

In this section we describe our approach for GP which employs LMs to generate code as programs representing value functions and policies for use in planning. Notably, all approaches to be described next are sound algorithms and are furthermore complete when used with complete search.

LMs for GP Our approach consists of two modules for solving a GP problem $\langle \mathcal{P}_{train}, \mathcal{P}_{test} \rangle$ as illustrated in Figure 1: **(1)** a program synthesis module (Section 3.1) and **(2)** a program instantiation module (Section 3.2). The *program synthesis* module (red in Figure 1) takes as input the training problems $\mathcal{P}_{train}$, corresponding domain $\mathcal{D}$, and a natural language prompt, and outputs a program implementing a value function or policy. The *program instantiation* module (yellow in Figure 1), takes as input a problem $P \in \mathcal{P}_{test}$ and a program from the previous module and outputs a plan α for P. Note that the LM is queried for a program once per domain $\mathcal{D}$ associated with the GP problem while the planner module is called for every problem P in $\mathcal{P}_{test}$.

3.1 LMs for program generation

Both the value function and policy programs are prompted to be generated as Python classes which extend a class containing the domain $\mathcal{D}$ associated with the GP problem to be solved. We follow a setup similar to [CPS25] for generating code as programs extended to both value functions and policies. More specifically, for any given domain, we prompt an LM for a program as a value function or policy with the following content:

(i) instructions for generating code for a program as a value function or policy,
(ii) the domain $\mathcal{D}$ corresponding to the GP problem and two problems in $\mathcal{P}_{train}$ in PDDL,
(iii) an example PDDL file for the Gripper domain [McD00] and a Gripper problem,
(iv) an example Python class encoding a value function or policy for Gripper.

Differently to [CPS25] we only provide example files for a single domain instead of two domains (Gripper and Logistics). Gripper is a simple PDDL domain consisting of two rooms and a set of balls located in one room. The objective is for a robot to move all balls in one room to the other, subject to capacity constraints. Logistics is a more complex domain than Gripper which commands a fleet of planes and trucks for delivering packages across various cities and locations. We emphasise that the example Gripper files and Python class (iii-iv) are used to provide in-context learning [DLD+24] for the LM to understand the syntax of PDDL and the Python class structure, but does not provide any information about solving the GP problem associated with $\mathcal{D}$.

3.2 Sound planning with LM-generated programs

Next, we describe how to **(a)** ensure that LM-generated policies are sound when used as reactive controllers, **(b)** use value functions for sound and complete planning and **(c)** combine both value functions and policies together for sound and complete planning and boosting performance over **(b)**.

Algorithm 1: Greedy Best First Search (GBFS) with a value function and policy (3.2.c)

Input: Planning problem $P = \langle \mathcal{S}, \mathcal{A}, f, \mathcal{G}, s_I, c \rangle$, value function h, and policy π.
Output: A plan α or failure if no plan exists.

1 **if** $s_I \in \mathcal{G}$ **then return** $\emptyset$
2 $q_H \leftarrow [s_I]$; $q_P \leftarrow []$; $visited \leftarrow \{s_I\}$; $popH \leftarrow \top$
3 **while** q_H *or* q_P *is not empty* **do**
4 **if** $popH = \top$ **then** $s \leftarrow \arg\min_{s \in q_H} h(s)$
5 **else** $s \leftarrow \arg\min_{s \in q_P} h(s)$
6 $popH \leftarrow !popH$; $q_P.push(\pi(s))$
7 **for** $a \in A(s)$ **do**
8   $s' \leftarrow f(s, a)$
9   **if** $s' \in visited$ **then continue**
10   **if** $s' \in \mathcal{G}$ **then return** *extract plan to* s'
11   $visited.insert(s')$; $q.push(s')$
12 **return** *failure*

(3.2.a) Sound policies as reactive controllers Given a planning problem $P = \langle \mathcal{S}, \mathcal{A}, f, \mathcal{G}, s_I, c \rangle$, the policy program π^{LM} is instantiated on P to represent a policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ that takes as input a state s and its applicable actions $A(s)$ and outputs an action $\pi(s) \in A(s)$. We furthermore have wrapper code around π^{LM} such that if due to an error or mistake in the generated code and $\pi(s) \notin A(s)$, we choose a random action from $A(s)$ instead. We use the policy program in the usual way for a policy via rollout. Specifically, we repeatedly apply the operation $s = f(s, \pi(s))$ starting from the initial state $s = s_I$ of a planning problem until either a goal is reached (i.e. $s \in \mathcal{G}$) or no applicable actions exist. Notably, this approach is sound, meaning that any plan returned by this procedure is valid.

Theorem 1. *Approach (3.2.a) is sound with respect to an input planning problem P.*

Proof sketch. This is because all predicted actions are applicable at their current state. Thus any sequence of actions generated by the policy is applicable from the initial state. Furthermore, a plan is only returned when the goal is reached so any returned action sequence reaches the goal. □

(3.2.b) Sound and complete value functions in search Equivalently to [CPS25], the value function program V^{LM} consists of a method representing a heuristic function $h : \mathcal{S} \rightarrow \mathbb{R} \cup \{\infty\}$ that takes as input a state s from P and outputs a value $h(s)$ used in GBFS described in Section 2. To ensure the output heuristic is safe, we have wrapper code that converts ∞ outputs to a large constant value. Thus, we have the following property that this approach is sound and complete.

Theorem 2. *Approach (3.2.b) is sound with respect to an input planning problem P and complete if the state space of P is finite.*

Proof sketch. This follows from the fact that GBFS is sound and complete for finite state spaces when used with a safe heuristic, and that the generated value function program used as a heuristic is ensured to be safe by disallowing ∞ values. □

(3.2.c) Sound and complete value functions and policies in search We now describe how to combine LM-generated policies and value functions with search. The main idea is that GBFS can be extended to two queues from which nodes are popped in a round robin fashion: one each for a value function V^{LM} and policy π^{LM}. Indeed, the usage of multiple queues have been explored in previous work with multiple heuristics [RH10] or *preferred operators*, actions that are deemed useful for achieving the goal within the computation of a heuristic function [HN01]. The algorithm is summarised in Algorithm 1, which begins by checking if the problem is trivially solvable (Line 1) before initialising the two queues q_H and q_P representing a queue containing successors of any expanded states, and a state predicted by the policy π, respectively (Line 2). The main loop alternates between popping states from the two queues, followed by pushing the state predicted by the policy into the q_P queue (Lines 3 to 6), and the remainder of the original GBFS algorithm for the q_H queue (Lines 7 to 11) described in Section 2. This approach is also sound and complete.

Theorem 3. *Approach (3.2.c) is sound with respect to an input planning problem P and complete if the state space of P is finite.*

Proof sketch. Extending GBFS with multiple queues preserves the same soundness and completeness properties of GBFS as the search is still exhaustive for finite state spaces. □

4 Experiments

We conduct experiments to address the following questions.

(**1**) **Q:** Which of LM-generated value functions used for search or policies used as is solve more planning problems within a fixed time limit, and how do they compare to PDDL planners?
A: Policies are faster for simpler problems, while search with value functions solve more complex problems. LM programs are competitive with PDDL planners on easy domains.

(**2**) **Q:** How important is soundness and completeness for planning performance?
A: Soundness is important but completeness is not always necessary.

(**3**) **Q:** Are LMs planning over word semantics or logical symbols?
A: LMs are shown to be capable to reason over PDDL planning problems represented in either semantic or symbolic text, but more experimentation is required to draw conclusive results.

Domains We evaluate the effectiveness of LM-generated value functions and policies on 10 standard PDDL planning domains (Blocksworld, Childsnack, Ferry, Floortile, Miconic, Rovers, Satellite, Sokoban, Spanner, Transport) with validation and testing problems (of increasing difficulty in terms of number of objects) taken from the Learning Track of the 2023 International Planning Competition [TAE+24]. Figure 2 summarises the ranges of problem sizes across the domains, noting that the testing sizes are up to two orders of magnitude larger than validation problems.

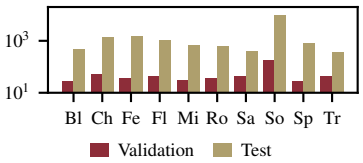


Figure 2: Number of objects (log y-axis) of validation and test problems across domains.

Implementation For our experiments, we implement a planner from scratch, namely LMPLAN, which prompts LMs for programs and both (a) uses value functions for heuristic search and (b) executes policies as reactive controllers. LMPLAN also implements the search using both value functions and policies introduced in Algorithm 1. The implementation consists of a combination of Python for heuristic evaluation and C++ for data structure representations of planning components. We use the SQLite library [Hip20] to compute actions applicable for a state in Line 7, as planning states can be viewed as databases and applicable action generation as database queries [CPHF20].

Validation for selecting LM-generated programs With regards to LMs, we experiment with DeepSeek-R1 [DGY+25], Gemini 2.0 Flash, and Gemini 2.5 Flash Preview 04-17 [GTGBW+23]. Similarly to [CPS25], we perform a validation procedure to select the best value function and policy for each domain out of several generated programs. Each LM is called 10 times to generate a value function and policy program for each domain. For each domain, the best value function (resp. policy) is then selected by taking the best average score of the program when used for search (resp. rollout) on 10 small, training problems. We choose a scoring function that favours models that solve the training problems quickly, as the time to generate a solution is a major performance metric for satisficing planning. Specifically, the *validation score* for each problem and program is given by $(1 + \log(t + 1))^{-1}$ if the program used for search or rollout solves a given problem in t seconds for $t < 60$ seconds, and 0 otherwise. Appendix A lists prompting costs, Appendix B shows LMs that generated the selected program, and Appendix C reports the corresponding generation times.

Approaches We evaluate our proposed approaches using LM-generated programs in the LMPLAN planner (•) and compare them to traditional PDDL planners (○) listed as follows:

- ○ h^{FF}: GBFS with the FF heuristic [HN01],
- ○ CPS25: reported results for LM-generated programs for heuristic search by [CPS25],
- ○ WLGOOSE: GBFS with state-of-the-art value functions learned with the Weisfeiler-Leman graph kernel for generating features from planning tasks [CTT24b] and Gaussian process regression for computing linear model weights,
- ○ LAMA: a state-of-the-art, general-purpose planner that uses multiple heuristics, queues and several optimisation techniques [RW10].
- • V^{LM}: GBFS using an LM-generated program as a value function,
- • π^{LM}: rollout of an LM-generated program as a policy,
- • $\pi^{\mathrm{LM}} \oplus V^{\mathrm{LM}}$: GBFS with two queues associated with V^{LM} and π^{LM},
- • $\pi^{\mathrm{LM}} \otimes V^{\mathrm{LM}}$: a choice between π^{LM} and $\pi^{\mathrm{LM}} \oplus V^{\mathrm{LM}}$ depending on whether π^{LM} or V^{LM} by themselves respectively achieves the higher average validation score.

Table 2: Coverage (↑) of existing planners (top) and LMPLAN approaches introduced in this paper (bottom), see Approaches for details. The S/C column represents if an approach is sound/complete. Best values in each column are highlighted, where 90 is the best achievable score per domain.

	S	C	Bl	Ch	Fe	Fl	Mi	Ro	Sa	So	Sp	Tr	Σ
h^{FF}	✓	✓	28	26	68	**12**	**90**	34	65	36	30	41	430
CPS25	✓	✓	66	22	–	4	**90**	32	–	30	70	59	*373
WLGOOSE	✓	✓	75	29	76	2	**90**	37	53	38	73	29	502
LAMA	✓	✓	61	**35**	68	11	**90**	**67**	89	**40**	30	66	557
V^{LM}	✓	✓	33	15	59	2	63	32	60	32	46	55	397
π^{LM}	✓	✗	**90**	11	**90**	0	**90**	12	**90**	0	**90**	**90**	563
$\pi^{\mathrm{LM}} \oplus V^{\mathrm{LM}}$	✓	✓	36	19	66	2	72	35	62	34	47	51	424
$\pi^{\mathrm{LM}} \otimes V^{\mathrm{LM}}$	✓	✗	**90**	19	**90**	2	**90**	35	**90**	34	**90**	**90**	**630**

* Note that CPS25 does not support the Ch and Sa domains, indicated by –, such that $\sum$ is not representative of its best performance.

There are 90 testing problems in each of the 10 domains for a total of 900 testing problems. Each approach is run on Intel Xeon Platinum 8268 cores for 1800 seconds and with a memory budget of 8GB for each problem. An exception is CPS25 where we report results from the original paper using AMD EPYC 7742 cores as their code is not yet publicly available. We report the coverage metric in Table 2, the number of problems solved within the given computational budgets.

(1) Do value functions for search or policies generated by LMs solve planning problems faster? We note from Table 2 that policies π^{LM} vastly outperform value functions V^{LM} for more domains but otherwise both are complementary overall. More specifically execution of the policy π^{LM} solves all problems for 6 domains (Blocksworld, Ferry, Miconic, Satellite, Spanner and Transport), but struggles to solve problems from the remaining domains. Regardless, π^{LM} already outperforms a state-of-the-art planner, LAMA, in overall coverage (563 against 557) as well as in number of domains (6 against 4). On the other hand, search with the value function V^{LM} is complete and hence provides more well-rounded performance across domains by solving at least 30 problems from each domain except Childsnack and Floortile, but has a lower overall cumulative coverage. A similar statement can be made by the baseline planners which all employ heuristic search with value functions. However, the validation procedure correctly identifies when a policy or search is performs better, resulting in the strong portfolio planner, $\pi^{\mathrm{LM}} \otimes V^{\mathrm{LM}}$, with a total coverage of 630. As to be discussed in the next question, $\pi^{\mathrm{LM}} \otimes V^{\mathrm{LM}}$ takes advantage of the $\pi^{\mathrm{LM}} \oplus V^{\mathrm{LM}}$ setting which improves upon the domains where π^{LM} struggle, and also upon V^{LM} for search.

(2) How important is soundness and completeness for planning performance? Recall that search with value functions is both sound and complete, while execution of our LM-generated policies is sound but not complete. We note that soundness is an important property: previous works show that LM prompting of plans, which is neither sound nor complete, achieve low coverage on planning problems which are trivial to solve for symbolic planning systems [VMSK23, VSK24]. In this work, all approaches are sound so we study the impact of complete algorithms. We recall from the previous question that π^{LM} and $\pi^{\mathrm{LM}} \otimes V^{\mathrm{LM}}$ are incomplete but achieve better overall performance over complete approaches. However, as also noted previously, the performance of policies on some domains are very poor when the LM did not understand how to solve these tasks.

However, we observe that combining the policies into (complete) search with value functions as done in $\pi^{\mathrm{LM}} \oplus V^{\mathrm{LM}}$ always matches or improves upon pure search with V^{LM}. Indeed, adding policy generated states into a queue strictly improves search in 8 out of 10 domains, including domains on which the policy π^{LM} by itself struggles, and improves the overall coverage of V^{LM} from 397 to 424. Furthermore, the validation procedure correctly identifies when to use search or policy execution which suggests that combining the best of complete and incomplete algorithms while maintaining soundness is a promising approach. We refer to Appendix D for a statistical analysis of a positive correlation between validation and test performance that supports this validation procedure.

(3) Are LMs planning over word semantics or logical symbols? In order to address this question, we perform the ablation introduced by [SDS+24] by replacing all type, predicate, function, schema and object names in all PDDL files with nondescriptive symbols; e.g. predicates are renamed to `p1`,

p2, ..., actions to a1, a2, ..., objects to o1, o2, ..., etc. Table 3 illustrates the coverage results of LM-generated value functions for search (V^{LM}) and policies for rollout (π^{LM}) on PDDL input files with and without semantic names.

Surprisingly and contrary to previous works performing similar experiments [VMH+23, SDS+24], we observe that in multiple settings, LM-generated programs perform better without than with semantic names in the PDDL inputs. LM-generated value functions perform better by at least 3 problems without semantic names on 3 domains and worse on 2 (see green and red cells in Table 3). The case for policies is performing better on 1 domain and worse on 2 domains. Specifically, removing semantic names often improves performance for value function generation but decreases for policy generation. However, there are still 3 domains for which policies can solve all problems even without semantic names in PDDL inputs (Fe, Mi, Sp). It is unclear exactly why this may be the case but the results mirror related work studying whether LMs can reason without comprehensible natural language inputs. Indeed, [PMB24] showed that removing word semantics from reasoning tokens in LMs, instead of the input reasoning problem as in our work, has minor impact to reasoning performance, while [SVG+25] showed that intermediate reasoning tokens may not reflect human-like or algorithm-interpretable trace semantics.

Table 3: Coverage (↑) with (sem) and without (sym) semantic names. Green/red cells indicate where sym solves at least 3 problems more/fewer problems than its sem counterpart for a domain.

	Bl	Ch	Fe	Fl	Mi	Ro	Sa	So	Sp	Tr	Σ
V_{sem}	33	15	59	2	63	32	60	32	46	55	397
V_{sym}	33	24	61	1	70	34	48	30	63	42	406
π_{sem}	90	11	90	0	90	12	90	0	90	90	563
π_{sym}	1	12	90	0	90	46	6	0	90	89	424

Limitations

Although our proposed approaches, specifically π^{LM} and $\pi^{\mathrm{LM}} \otimes V^{\mathrm{LM}}$ in Table 2, achieve the highest total coverage, they are not necessarily the best performing planners across all domains and metrics. This fact may change with the improvement of LMs over time as they understand how to solve more complex planning problems which are still tractable to solve satisficingly, such as the Childsnack, Floortile and Rovers domains.

Another limitation of the synthesised policies is that they have no completeness or termination guarantee. Although it is possible to guarantee such properties by combining them with search or by using them as an epsilon-greedy policy in the case of domains with reversible actions, the latter approach comes at a cost of extremely poor plan quality.

Indeed, we have yet to discuss plan quality. Figure 3 compares the solution qualities of various approaches. We direct the reader to the 2nd plot (π^{LM} compared against LAMA) and note that the LM-generated policy returns inferior plans on the Blocksworld and Transport domains, sometimes up to 100 times worse for several Transport problems. When analysing the plans and generated code, the policy sometimes randomly selects unnecessary actions that undo previous actions. On the other hand, LM-generated value functions perform similarly to PDDL planners in terms of plan quality (1st plot: V^{LM} compared against LAMA). Similarly, the effect of input representation on plan quality of LM-generated value functions is neglible (3rd plot: V_{sem} compared against V_{sym}). However, there is more variance for the case of policies, especially Rovers where although π_{sym} solves more problems, it achieves significantly worse plans (4th plot: π_{sem} compared against π_{sym}).

Lastly, although replacing semantic names with arbitrary symbols in our PDDL encodings does not degrade the performance of value function generation, doing so significantly decreases the performance of generated policies on 2 domains (Blocksworld and Satellite). Furthermore, it is not obvious why the performance of LM-generated value functions improves when semantic names are removed which warrants further investigation.

5 Related Work

LMs for PDDL Planning Previous works have shown that querying LMs directly to output plans [VMSK23, VSK24] or using LMs themselves as value functions [KKSS24] result in poor planning performance and LM efficiency. Instead, LMs have shown more success in hybrid systems [KVG+24] such as those which generate or leverage PDDL models from natural language to

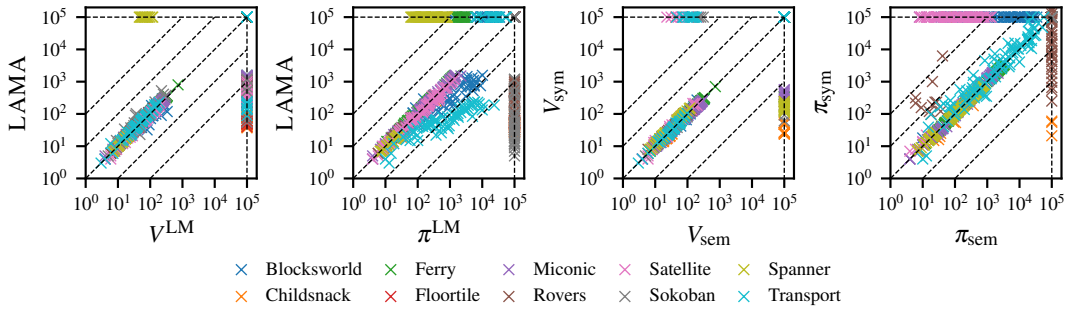


Figure 3: Returned plan cost (↓) of planners labelled in the x and y-axes in log scale. Problems that were not solved by one planner has their respective metric set to the axis limit. Points on the top left triangle favour the x-axis planner while points on the bottom right triangle favour the y-axis planner.

be solved by or with the aid of PDDL planning technology [CWF+22, LAM+23, XYZ+23, LJZ+23, GVSK23, OSK+24, LPL+25]. The closest related works to ours that study generalised planning via LM-generated programs [SDS+24] and via LM-generated value functions [TVS25, CPS25] provide significantly stronger planning performance than standalone LM planners. In the former work by [SDS+24], LMs are queried to generate code which aims to directly synthesise a plan for a PDDL planning problem of a given domain. Although external verifiers are used to improve the code in a validation stage, the approach is not sound for test problems. The latter works by [TVS25] and [CPS25] query LMs to generate a value function for use in heuristic search by leveraging the PDDL input structure and existing planners. [TVS25] generate a value function for each new PDDL problem, whereas we and [CPS25] only use one value function for all PDDL problems within the same PDDL domain. Our work differs from these two works as we also synthesise *sound policies* via LMs as opposed to value functions for search or unsound programs. We further perform an ablation study that shows the LMs used in our framework do not suffer from performance when replacing semantic words in PDDL problem inputs with meaningless symbols.

Learning Reusable Value Functions and Policies AI planning researchers have been studying representations and approaches for learning reusable value functions and policies for PDDL planning which can generalise to unseen problems of arbitrary numbers of objects [CdlRF+12, IM19, CSJ19]. This is analogous to reinforcement learning whose approaches can be categorised into value function learning [WD92, MKS+13, MKS+15, vHGS16], policy gradient methods [Wil92, SLA+15, SWD+17], and actor-critic methods which combine value function and policy learning [KT99, SMSM99, MBM+16, LHP+16]. Recent works in machine learning for PDDL planning involve learning value functions [STT20, KS21, GAC+22, CTT24a, CTT24b, HTT+24, CT24] and policies [TTTX18, TTTX20, SBG22, WT24]. Our work is one of the few in this body of work which learns to evaluate both states and actions [WT25].

Generalised Planning Generalised planning (GP) aims to compute programs that can solve families of related planning problems. GP stemmed from synthesising programs containing conditionals and loops [Lev05, SIZ08, SCJ24] from which researchers found various approaches for representing programs such as with memoryless finite-state controllers [BPG09, BPG10, HD11, AJJ18] or policies derived from lifted rules [Kha99, MG04, SZIG11, IM19, FCGP19, YSC+22, DSG22, HG24]. Key attributes of symbolic GP approaches include guarantees of soundness, completeness, and termination of policies, some or all of which are usually not guaranteed by LM or deep learning architectures. Our work studies the effect of soundness and completeness for LMs for GP.

6 Conclusion

We introduce a language model (LM) planner, LMPLAN, that generates Python programs as value functions and sound policies for PDDL planning. Conducted experiments show that LMPLAN achieves strong planning performance relative to state-of-the-art planners and recent LM approaches. We also identify that, surprisingly, LMPLAN sometimes show better planning performance over purely symbolic representations of planning problems. This observation challenges previous hypotheses that LMs cannot reason over meaningless symbols and is worth much further exploration.

Acknowledgements

This work was carried out when all three authors were research interns at the Vector Institute for AI, Toronto, Canada. We gratefully acknowledge funding from the Natural Sciences and Engineering Research Council of Canada (NSERC) and the Canada CIFAR AI Chairs Program. Resources used in preparing this research were provided, in part, by the Province of Ontario, the Government of Canada through CIFAR, and companies sponsoring the Vector Institute. We thank Felix Dangel for feedback on the figures in the paper. Finally, the first three authors acknowledge the Nando's team on Bay Street for the copious amounts of Extra Hot chicken that helped fuel this work.

References

[AJJ18] Javier Segovia Aguas, Sergio Jiménez, and Anders Jonsson. Computing hierarchical finite state controllers with classical planning. *J. Artif. Intell. Res.*, 62:755–797, 2018.

[AKS+25] Ashay Athalye, Nishanth Kumar, Tom Silver, Yichao Liang, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Predicate invention from pixels via pretrained vision-language models. *CoRR*, abs/2501.00296, 2025.

[BFG19] Blai Bonet, Guillem Francès, and Hector Geffner. Learning features and abstract actions for computing generalized plans. In *AAAI*, 2019.

[BJMR05] John L. Bresina, Ari K. Jónsson, Paul H. Morris, and Kanna Rajan. Activity planning for the mars exploration rovers. In *ICAPS*, 2005.

[BLG97] Blai Bonet, Gábor Loerincs, and Hector Geffner. A robust and fast action selection mechanism for planning. In *AAAI*, 1997.

[BMR+20] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *NeurIPS*, 2020.

[BPG09] Blai Bonet, Héctor Palacios, and Hector Geffner. Automatic derivation of memoryless policies and finite-state controllers using classical planners. In *ICAPS*, 2009.

[BPG10] Blai Bonet, Héctor Palacios, and Hector Geffner. Automatic derivation of finite-state machines for behavior control. In *AAAI*, 2010.

[BT91] Dimitri P. Bertsekas and John N. Tsitsiklis. An analysis of stochastic shortest path problems. *Math. Oper. Res.*, 16:580–595, 1991.

[Byl94] Tom Bylander. The computational complexity of propositional STRIPS planning. *Artif. Intell.*, 69(1-2):165–204, 1994.

[CdlRF+12] Sergio Jiménez Celorrio, Tomás de la Rosa, Susana Fernández, Fernando Fernández, and Daniel Borrajo. A review of machine learning for automated planning. *Knowl. Eng. Rev.*, 27:433–467, 2012.

[Cha87] David Chapman. Planning for conjunctive goals. *Artif. Intell.*, 32(3):333–377, 1987.

[CLL21] Zhenhe Cui, Yongmei Liu, and Kailun Luo. A uniform abstraction framework for generalized planning. In *IJCAI*, 2021.

[CND+23] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani

Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. Palm: Scaling language modeling with pathways. *J. Mach. Learn. Res.*, 24:240:1–240:113, 2023.

[CPHF20] Augusto B. Corrêa, Florian Pommerening, Malte Helmert, and Guillem Francès. Lifted successor generation using query optimization techniques. In *ICAPS*, 2020.

[CPS25] Augusto B. Corrêa, André Grahl Pereira, and Jendrik Seipp. Classical planning with llm-generated heuristics: Challenging the state of the art with python code. *CoRR*, abs/2503.18809, 2025.

[CSJ19] Sergio Jiménez Celorrio, Javier Segovia-Aguas, and Anders Jonsson. A review of generalized planning. *Knowl. Eng. Rev.*, 34:e5, 2019.

[CT24] Dillon Z. Chen and Sylvie Thiébaux. Graph learning for numeric planning. In *NeurIPS*, 2024.

[CTJ+21] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. *CoRR*, abs/2107.03374, 2021.

[CTT24a] Dillon Z. Chen, Sylvie Thiébaux, and Felipe Trevizan. Learning domain-independent heuristics for grounded and lifted planning. In *AAAI*, 2024.

[CTT24b] Dillon Z. Chen, Felipe W. Trevizan, and Sylvie Thiébaux. Return to tradition: Learning reliable heuristics with classical machine learning. In *ICAPS*, 2024.

[CWF+22] Katherine M. Collins, Catherine Wong, Jiahai Feng, Megan Wei, and Josh Tenenbaum. Structured, flexible, and robust: benchmarking and improving large language models towards more human-like behavior in out-of-distribution reasoning tasks. In *CogSci*, 2022.

[DGY+25] DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, and S. S. Li. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *CoRR*, abs/2501.12948, 2025.

[DLD+24] Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Jingyuan Ma, Rui Li, Heming Xia, Jingjing Xu, Zhiyong Wu, Baobao Chang, Xu Sun, Lei Li, and Zhifang Sui. A survey on in-context learning. In *EMNLP*, 2024.

[DSG22] Dominik Drexler, Jendrik Seipp, and Hector Geffner. Learning sketches for decomposing planning problems into subproblems of bounded width. In *ICAPS*, 2022. Code accessed from commit `7a7ea6` in https://github.com/drexlerd/sketch-learner.

[DXS+23] Danny Driess, Fei Xia, Mehdi S. M. Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, Wenlong Huang, Yevgen Chebotar, Pierre Sermanet, Daniel Duckworth, Sergey Levine, Vincent Vanhoucke, Karol Hausman, Marc Toussaint, Klaus Greff, Andy Zeng, Igor Mordatch, and Pete Florence. Palm-e: An embodied multimodal language model. In *ICML*, 2023.

[ENS95] Kutluhan Erol, Dana S. Nau, and V. S. Subrahmanian. Complexity, decidability and undecidability results for domain-independent planning. *Artif. Intell.*, 76:75–88, 1995.

[FBG21] Guillem Francès, Blai Bonet, and Hector Geffner. Learning general planning policies from small examples without supervision. In *AAAI*, 2021.

[FCGP19] Guillem Francès, Augusto B. Corrêa, Cedric Geissmann, and Florian Pommerening. Generalized potential heuristics for classical planning. In *IJCAI*, 2019.

[FN71] Richard Fikes and Nils J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artif. Intell.*, 2:189–208, 1971.

[GAC+22] Clement Gehring, Masataro Asai, Rohan Chitnis, Tom Silver, Leslie Pack Kaelbling, Shirin Sohrabi, and Michael Katz. Reinforcement learning for classical planning: Viewing heuristics as dense reward generators. In *ICAPS*, 2022.

[GB13] Hector Geffner and Blai Bonet. *A Concise Introduction to Models and Methods for Automated Planning*. Morgan & Claypool Publishers, 2013.

[GNT04] Malik Ghallab, Dana S. Nau, and Paolo Traverso. *Automated planning - theory and practice*. Elsevier, 2004.

[GTGBW+23] Rohan Anil Gemini Team Google, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M. Dai, Anja Hauth, Katie Millican, David Silver, Slav Petrov, Melvin Johnson, Ioannis Antonoglou, Julian Schrittwieser, Amelia Glaese, Jilin Chen, Emily Pitler, Timothy P. Lillicrap, Angeliki Lazaridou, Orhan Firat, James Molloy, Michael Isard, Paul Ronald Barham, Tom Hennigan, Benjamin Lee, Fabio Viola, Malcolm Reynolds, Yuanzhong Xu, Ryan Doherty, Eli Collins, Clemens Meyer, Eliza Rutherford, Erica Moreira, Kareem Ayoub, Megha Goel, George Tucker, Enrique Piqueras, Maxim Krikun, Iain Barr, Nikolay Savinov, Ivo Danihelka, Becca Roelofs, Anaïs White, Anders Andreassen, Tamara von Glehn, Lakshman Yagati, Mehran Kazemi, Lucas Gonzalez, Misha Khalman, Jakub Sygnowski, and et al. Gemini: A family of highly capable multimodal models. *CoRR*, abs/2312.11805, 2023.

[GVSK23] Lin Guan, Karthik Valmeekam, Sarath Sreedharan, and Subbarao Kambhampati. Leveraging pre-trained large language models to construct and utilize world models for model-based task planning. In *NeurIPS*, 2023.

[HD11] Yuxiao Hu and Giuseppe De Giacomo. Generalized planning: Synthesizing plans that work for multiple environments. In *IJCAI*, 2011.

[Hel02] Malte Helmert. Decidability and undecidability results for planning with numerical state variables. In *AIPS*, 2002.

[HG24] Till Hofmann and Hector Geffner. Learning generalized policies for fully observable non-deterministic planning domains. In *IJCAI*, 2024.

[Hip20] Dwayne Richard Hipp. Sqlite, 2020. Accessed from https://www.sqlite.org/index.html.

[HLC25] Sukai Huang, Nir Lipovetzky, and Trevor Cohn. Planning in the dark: Llm-symbolic planning pipeline without experts. In *AAAI*, 2025.

[HLMM19] Patrik Haslum, Nir Lipovetzky, Daniele Magazzeni, and Christian Muise. *An Introduction to the Planning Domain Definition Language*. Morgan & Claypool Publishers, 2019.

[HN01] Jörg Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation through heuristic search. *J. Artif. Intell. Res.*, 14:253–302, 2001.

[HNR68] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Syst. Sci. Cybern.*, 4(2):100–107, 1968.

[HTT+24] Mingyu Hao, Felipe Trevizan, Sylvie Thiébaux, Patrick Ferber, and Jörg Hoffmann. Guiding GBFS through learned pairwise rankings. In *IJCAI*, 2024.

[IBB+25] Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Manuel Y. Galliker, Dibya Ghosh, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Devin LeBlanc, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Allen Z. Ren, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, James Tanner, Quan Vuong, Homer Walke, Anna Walling, Haohuan Wang, Lili Yu, and Ury Zhilinsky. $\pi_{0.5}$: a vision-language-action model with open-world generalization. *CoRR*, abs/2504.16054, 2025.

[IM19] León Illanes and Sheila A. McIlraith. Generalized planning via abstraction: Arbitrary numbers of objects. In *AAAI*, 2019.

[KGR+22] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. *CoRR*, abs/2205.11916, 2022.

[Kha99] Roni Khardon. Learning action strategies for planning domains. *Artif. Intell.*, 113:125–148, 1999.

[KKSS24] Michael Katz, Harsha Kokel, Kavitha Srinivas, and Shirin Sohrabi. Thought of search: Planning with language models through the lens of efficiency. In *NeurIPS*, 2024.

[KS21] Rushang Karia and Siddharth Srivastava. Learning generalized relational heuristic networks for model-agnostic planning. In *AAAI*, 2021.

[KT99] Vijay R. Konda and John N. Tsitsiklis. Actor-critic algorithms. In *NIPS*, 1999.

[KVG+24] Subbarao Kambhampati, Karthik Valmeekam, Lin Guan, Mudit Verma, Kaya Stechly, Siddhant Bhambri, Lucas Saldyt, and Anil Murthy. Position: Llms can't plan, but can help planning in llm-modulo frameworks. In *ICML*, 2024.

[LAM+23] Kevin Lin, Christopher Agia, Toki Migimatsu, Marco Pavone, and Jeannette Bohg. Text2motion: from natural language instructions to feasible plans. *Auton. Robots*, 47(8):1345–1365, 2023.

[Lev05] H. J. Levesque. Planning with loops. In *IJCAI*, 2005.

[LG95] Philippe Laborie and Malik Ghallab. Planning with sharable resource constraints. In *IJCAI*, 1995.

[LHP+16] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *ICLR*, 2016.

[LJZ+23] Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone. LLM+P: empowering large language models with optimal planning proficiency. *CoRR*, abs/2304.11477, 2023.

[LKT+25] Yichao Liang, Nishanth Kumar, Hao Tang, Adrian Weller, Joshua B. Tenenbaum, Tom Silver, João F. Henriques, and Kevin Ellis. Visualpredicator: Learning abstract world models with neuro-symbolic predicates for robot planning. In *ICLR*, 2025.

[LPL+25] Xiaotian Liu, Ali Pesaranghader, Hanze Li, Punyaphat Sukcharoenchaikul, Jaehong Kim, Tanmana Sadhu, Hyejeong Jeon, and Scott Sanner. Open-world planning via lifted regression with llm-inferred affordances for embodied agents. In *ACL*, 2025.

[LTY+24] Fei Liu, Xialiang Tong, Mingxuan Yuan, Xi Lin, Fu Luo, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. Evolution of heuristics: Towards efficient automatic algorithm design using large language model. In *ICML*, 2024.

[MBM+16] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *ICML*, 2016.

[McD96] Drew V. McDermott. A heuristic estimator for means-ends analysis in planning. In *AIPS*, 1996.

[McD00] Drew V. McDermott. The 1998 AI planning systems competition. *AI Mag.*, 21(2):35–55, 2000.

[MG04] Mario Martín and Hector Geffner. Learning generalized policies from planning examples using concept languages. *Appl. Intell.*, 20:9–19, 2004.

[MGH+98] Drew McDermott, Malik Ghallab, Adele E. Howe, Craig A. Knoblock, Ashwin Ram, Manuela M. Veloso, Daniel S. Weld, and David E. Wilkins. PDDL-the planning domain definition language. Technical report, 1998.

[MK12] Mausam and Andrey Kolobov. *Planning with Markov Decision Processes: An AI Perspective*. Morgan & Claypool Publishers, 2012.

[MKS+13] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

[MKS+15] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nat.*, 518(7540):529–533, 2015.

[NPH+23] Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. Codegen: An open large language model for code with multi-turn program synthesis. In *ICLR*, 2023.

[NVE+25] Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. Alphaevolve: A gemini-powered coding agent for designing advanced algorithms, 2025. Accessed from https://deepmind.google/discover/blog/alphaevolve-a-gemini-powered-coding-agent-for-designing-advanced-algorithms/.

[Ope25] Open AI. Introducing openai o3 and o4-mini, 2025. Accessed from https://openai.com/index/introducing-o3-and-o4-mini.

[OSK+24] James T. Oswald, Kavitha Srinivas, Harsha Kokel, Junkyu Lee, Michael Katz, and Shirin Sohrabi. Large language models as planning domain generators. In *ICAPS*, 2024.

[PMB24] Jacob Pfau, William Merrill, and Samuel R. Bowman. Let's think dot by dot: Hidden computation in transformer language models. In *COLM*, 2024.

[RBN+24] Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. Mathematical discoveries from program search with large language models. *Nat.*, 625(7995):468–475, 2024.

[RH10] Gabriele Röger and Malte Helmert. The more, the merrier: Combining heuristic estimators for satisficing planning. In *ICAPS*, 2010.

[RW10] Silvia Richter and Matthias Westphal. The LAMA planner: Guiding cost-based anytime planning with landmarks. *J. Artif. Intell. Res.*, 39:127–177, 2010.

[San10] Scott Sanner. Relational dynamic influence diagram language (rddl): Language description. Technical report, 2010.

[SBG22] Simon Ståhlberg, Blai Bonet, and Hector Geffner. Learning general optimal policies with graph neural networks: Expressive power, transparency, and limits. In *ICAPS*, 2022.

[SCJ24] Javier Segovia-Aguas, Sergio Jiménez Celorrio, and Anders Jonsson. Generalized planning as heuristic search: A new planning search-space that leverages pointers over objects. *Artif. Intell.*, 330:104097, 2024.

[SCK+23] Tom Silver, Rohan Chitnis, Nishanth Kumar, Willie McClinton, Tomás Lozano-Pérez, Leslie Pack Kaelbling, and Joshua B. Tenenbaum. Predicate invention for bilevel planning. In *AAAI*, 2023.

[SDS+24] Tom Silver, Soham Dan, Kavitha Srinivas, Joshua B. Tenenbaum, Leslie Kaelbling, and Michael Katz. Generalized planning in PDDL domains with pretrained large language models. In *AAAI*, 2024.

[SIZ08] Siddharth Srivastava, Neil Immerman, and Shlomo Zilberstein. Learning generalized plans using abstract counting. In *AAAI*, 2008.

[SIZ11] Siddharth Srivastava, Neil Immerman, and Shlomo Zilberstein. A new representation and associated algorithms for generalized planning. *Artif. Intell.*, 175(2):615–647, 2011.

[SK23] Sarath Sreedharan and Michael Katz. Optimistic exploration in reinforcement learning using symbolic model estimates. In *NeurIPS*, 2023.

[SKH20] Jendrik Seipp, Thomas Keller, and Malte Helmert. Saturated cost partitioning for optimal classical planning. *J. Artif. Intell. Res.*, 67:129–167, 2020.

[SLA+15] John Schulman, Sergey Levine, Pieter Abbeel, Michael I. Jordan, and Philipp Moritz. Trust region policy optimization. In *ICML*, 2015.

[SMSM99] Richard S. Sutton, David A. McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *NeurIPS*, 1999.

[STT20] William Shen, Felipe Trevizan, and Sylvie Thiébaux. Learning Domain-Independent Planning Heuristics with Hypergraph Networks. In *ICAPS*, 2020.

[SVG+25] Kaya Stechly, Karthik Valmeekam, Atharva Gundawar, Vardhan Palod, and Subbarao Kambhampati. Beyond semantics: The unreasonable effectiveness of reasonless intermediate tokens. *CoRR*, abs/2505.13775, 2025.

[SWD+17] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[SZIG11] Siddharth Srivastava, Shlomo Zilberstein, Neil Immerman, and Hector Geffner. Qualitative numeric planning. In *AAAI*, 2011.

[TAE+24] Ayal Taitler, Ron Alford, Joan Espasa, Gregor Behnke, Daniel Fiser, Michael Gimelfarb, Florian Pommerening, Scott Sanner, Enrico Scala, Dominik Schreiber, Javier Segovia-Aguas, and Jendrik Seipp. The 2023 international planning competition. *AI Mag.*, 45:280–296, 2024.

[TTTX18] Sam Toyer, Felipe W. Trevizan, Sylvie Thiébaux, and Lexing Xie. Action schema networks: Generalised policies with deep learning. In *AAAI*, 2018.

[TTTX20] Sam Toyer, Sylvie Thiébaux, Felipe Trevizan, and Lexing Xie. Asnets: Deep learning for generalised planning. *J. Artif. Intell. Res.*, 68:1–68, 2020.

[TVS25] Alexander Tuisov, Yonatan Vernik, and Alexander Shleyfman. Llm-generated heuristics for AI planning: Do we even need domain-independence anymore? *CoRR*, abs/2501.18784, 2025.

[TZM25] Marcus Tantakoun, Xiaodan Zhu, and Christian Muise. Llms as planning modelers: A survey for leveraging large language models to construct automated planning models. *CoRR*, abs/2503.18971, 2025.

[UPS25] UPS. Global reporting initiative. Accessed from https://about.ups.com/content/dam/upsstories/images/our-impact/reporting/2024-UPS-GRI-Report.pdf, 2025.

[vHGS16] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *AAAI*, 2016.

[VMH+23] Karthik Valmeekam, Matthew Marquez, Alberto Olmo Hernandez, Sarath Sreedharan, and Subbarao Kambhampati. Planbench: An extensible benchmark for evaluating large language models on planning and reasoning about change. In *NeurIPS*, 2023.

[VMS21] Pulkit Verma, Shashank Rao Marpally, and Siddharth Srivastava. Asking the Right Questions: Learning Interpretable Action Models Through Query Answering. In *AAAI*, 2021.

[VMS22] Pulkit Verma, Shashank Rao Marpally, and Siddharth Srivastava. Discovering user-interpretable capabilities of black-box planning agents. In *KR*, 2022.

[VMSK23] Karthik Valmeekam, Matthew Marquez, Sarath Sreedharan, and Subbarao Kambhampati. On the planning abilities of large language models - A critical investigation. In *NeurIPS*, 2023.

[VSK24] Karthik Valmeekam, Kaya Stechly, and Subbarao Kambhampati. Llms still can't plan; can lrms? A preliminary evaluation of openai's o1 on planbench. *CoRR*, abs/2409.13373, 2024.

[WD92] Christopher J.C.H. Watkins and Peter Dayan. Q-learning. *Mach. Learn.*, 8:279–292, 1992.

[Wil92] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.*, 8:229–256, 1992.

[WT24] Ryan X. Wang and Sylvie Thiébaux. Learning generalised policies for numeric planning. In *ICAPS*, 2024.

[WT25] Ryan X. Wang and Felipe Trevizan. Leveraging action relational structures for integrated learning and planning. In *ICAPS*, 2025.

[WWS+22] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed H. Chi, Quoc Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. *CoRR*, abs/2201.11903, 2022.

[XGT24] Kai Xi, Stephen Gould, and Sylvie Thiébaux. Neuro-symbolic learning of lifted action models from visual traces. In *ICAPS*, 2024.

[XYZ+23] Yaqi Xie, Chen Yu, Tongyao Zhu, Jinbin Bai, Ze Gong, and Harold Soh. Translating natural language to planning goals with large-language models. *CoRR*, abs/2302.05128, 2023.

[YSC+22] Ryan Yang, Tom Silver, Aidan Curtis, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. PG3: policy-guided planning for generalized policy generation. In *IJCAI*, 2022.

A Money Usage Estimate

Table 4: Money usage estimate from calling LM APIs for all experiments; 400 API calls were made per model: 200 for PDDL files with semantic names, and 200 for without semantic names.

Model	Average $USD per query	Total $USD
DeepSeek-R1	0.00655	2.62
Gemini 2.0 Flash	0	0
Gemini 2.5 Flash Preview 04-17	0	0

B Best LM Generated Programs Chosen by Validation

Table 5: Best LM-generated program chosen by the validation procedure described in Section 4 for experiments of PDDL files with (semantic) and without (symbolic) semantic names. The LM models used in the experiments are DeepSeek-R1 (DS-R1), Gemini 2.0 Flash (Gem2.0), and Gemini 2.5 Flash Preview 04-17 (Gem2.5).

	Value Functions		Policies	
Domain	semantic	symbolic	semantic	symbolic
Blocksworld	Gem2.5	Gem2.5	Gem2.5	Gem2.0
Childsnack	DS-R1	Gem2.5	DS-R1	DS-R1
Ferry	DS-R1	DS-R1	Gem2.5	Gem2.5
Floortile	Gem2.5	Gem2.5	DS-R1	DS-R1
Miconic	Gem2.0	DS-R1	Gem2.0	Gem2.5
Rovers	Gem2.5	DS-R1	Gem2.5	Gem2.0
Satellite	Gem2.5	Gem2.5	Gem2.5	Gem2.5
Sokoban	Gem2.5	Gem2.5	DS-R1	Gem2.5
Spanner	Gem2.0	DS-R1	Gem2.0	Gem2.5
Transport	DS-R1	Gem2.5	Gem2.0	DS-R1

C LM Program Generation Times

Table 6: Time taken in seconds for LMs in Table 5 to generate a program.

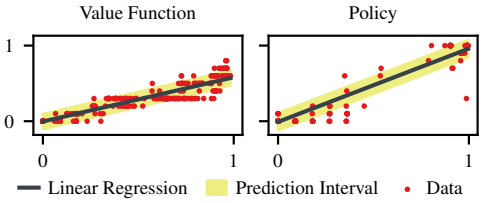
	Value Functions		Policies	
Domain	semantic	symbolic	semantic	symbolic
Blocksworld	78.0	108.2	107.4	3.4
Childsnack	329.8	119.7	260.2	277.3
Ferry	310.6	419.2	42.6	168.5
Floortile	118.4	104.5	528.2	474.2
Miconic	1.5	375.6	4.8	98.2
Rovers	50.8	331.0	98.6	3.0
Satellite	93.7	102.1	109.8	151.0
Sokoban	65.7	152.3	527.7	33.0
Spanner	2.9	477.3	2.2	60.3
Transport	434.2	113.7	3.6	394.7

D Correlation Between Validation and Test Performance

We perform a statistical correlation analysis between the validation score on validation problems and coverage on testing problems. Specifically, for each benchmark domain and LM model, we have multiple $V^{\text{LM}}/\pi^{\text{LM}}$ programs which we run heuristic search/policy execution on 11 small planning problems (every 9th problem in the training split) and 10 testing problems (every 9th problem in the testing split). These experiments were performed after the aforementioned experiments to prevent overfitting to the test set. Figure 4 illustrates the results.

Interestingly, for both value functions and policies, there is a statistically significant ($p < 0.05$) positive Pearson correlation ($\rho \simeq 1$) between the validation metric and test performance, which suggests that validation sets provide a useful proxy for estimating testing performance. However, the correlation is not perfect as we noted that the programs with the best validation scores did not provide the best overall coverage. The correlation remains high when conditioning on individual domains for where enough unique samples were provided, with an exception being Childsnack policies. We lastly note that performing the validation procedure for the $\pi^{\text{LM}} \otimes V^{\text{LM}}$ configuration correctly identifies whether value functions or policies perform better on each domain, a priori to running the test-time experiments.

Figure 4: *Left*: correlation coefficients conditioned on domain. *Right*: average coverage (y-axis) vs. validation score (x-axis) for LM-generated programs.

	Bl	Ch	Fe	Fl	Mi	Ro	Sa	So	Sp	Tr
V^{LM}	1.0	0.8	1.0	–	1.0	1.0	1.0	1.0	1.0	1.0
π^{LM}	1.0	0.4	1.0	–	1.0	0.8	0.9	–	1.0	1.0

Cells with – values indicate that there were fewer than 2 unique datapoints for each domain.