Using Large Language Models for Abstraction of Planning Domains

Bita Banihashemi[1] , **Megh Patel**[2] , **Yves Lespérance**[2]
[1]IGDORE
[2]York University
bita.banihashemi@igdore.org, megh2k@my.yorku.ca, lesperan@yorku.ca

Abstract

Generating an abstraction of a dynamic domain that aligns with a given purpose remains a significant challenge given that the choice of such an abstraction can impact an agent's ability to plan, reason, and provide explanations effectively. We model the agent's concrete behaviors in PDDL and investigate the use of in-context learning with large language models (LLMs) for the generation of abstract PDDL domains and problem instances, given an abstraction objective specified in natural language. The benchmark examples we use are new and have not been part of the data any LLMs have been trained on. We consider three categories of abstractions: abstraction of choice of alternative concrete actions, abstraction of sequences of concrete actions, and abstraction of action/predicate parameters, as well as combinations of these. The generated abstract PDDL domains and problem instances are then checked by symbolic validation tools as well as human experts. Our experiments show that GPT-4o can generally synthesize useful planning domain abstractions in simple settings, although they are better at abstracting over actions than over the associated fluents.

1 Introduction

The ability to generate abstractions that ignore irrelevant details is a crucial human cognitive ability that supports reasoning and communication. This has inspired significant research in AI, where for instance, abstraction has been exploited to improve the efficiency of planning (e.g., [Chen and Bercher, 2021]), provide explanations of agents' behavior (e.g., [Seegebarth *et al.*, 2012]), and in reinforcement learning (e.g., [Sutton *et al.*, 1999]).

[Banihashemi *et al.*, 2017; Banihashemi *et al.*, 2024] (BDL17) developed a general framework for *agent abstraction* in *dynamic domains* which is based on the situation calculus [McCarthy and Hayes, 1969; Reiter, 2001] and the ConGolog agent programming language [De Giacomo *et al.*, 2000]. The account formalizes notions of sound/complete abstractions between a high-level (HL) action theory and a low-level (LL) action theory representing the agent's possible behaviors at different levels of detail. These notions are based on the existence of a bisimulation relation between their respective models relative to a *refinement mapping* that maps high-level fluents to low-level state formulas and high-level actions to ConGolog programs over the low-level theory that implement them. It is shown that sound/complete abstractions have many useful properties that allow one to reason at the high level and refine the results at the low level, and they can also be used for monitoring and explanation. There is also related work on hierarchical planning, for instance, hierarchical task network (HTN) planning [Erol *et al.*, 1996] and planning with domain-specific control knowledge specified in linear temporal logic (LTL) [Bacchus and Kabanza, 2000].

(BDL17) specifies under what conditions a high-level action theory is a sound/complete abstraction of a low-level one under a given refinement mapping, but it does not say how one obtains such an abstraction. In general, there would be several different abstractions of a domain, each of which serves a particular purpose and could be used in a different application. Generating an abstraction of a domain that aligns with a given purpose remains a significant challenge. So far, there has only been limited work on solvers that can synthesize abstractions for a given mapping and verify their soundness and/or completeness (e.g., [Luo *et al.*, 2020; Luo, 2023; Fang *et al.*, 2025; Dong *et al.*, 2025]). But relying on human experts to generate such abstractions by hand is costly, time consuming, and does not scale.

Much recent work has investigated the use of large language models (LLMs) [Zhao *et al.*, 2023] for reasoning about action and the generation of planning specifications. While some work shows limitations of LLMs for reasoning about actions and planning (e.g., [Valmeekam *et al.*, 2024; He *et al.*, 2023]), other work shows good potential for the generation of Planning Domain Definition Language (PDDL) [McDermott *et al.*, 1998] specifications (e.g., [Oswald *et al.*, 2024; Guan *et al.*, 2023]).

It is clear that LLMs embed comprehensive world knowledge and that they can generate PDDL specifications. Thus it seems reasonable that they could be very helpful in generating useful abstractions of planning domains and problems and encode them in a formal language such as PDDL [Fox and Long, 2011].

In this paper, we investigate the ability of GPT-4o [Hurst *et al.*, 2024] to generate such abstractions. We define the *Plan-*

ning Domain Abstraction Generation (PDAG) task as: given (1) a concrete planning domain in PDDL (2) a concrete planning problem instance in PDDL, (3) a brief natural language description of the domain, and (4) a purpose of abstraction in natural language, use in-context learning with LLMs to generate (a) an abstract planning domain in PDDL and (b) an abstract planning problem instance in PDDL that meet the purpose of abstraction.

We consider three basic categories of abstractions: (1) abstraction of alternative concrete actions, e.g. $book\_hotel$ and $book\_airbnb$ could be abstracted to $book\_accommodation$, (2) abstraction of sequences of concrete actions, e.g., the sequence $enter\_username$ and $enter\_password$ could be abstracted to $login$, and (3) abstraction of action/predicate parameters, e.g., abstracting over the $room\_view$ parameter of $book\_room$ and dropping associated fluents, types, and actions. We also examine a fourth category which is a combination of (1) and (2), i.e., abstraction of alternative sequences of concrete actions. We have developed a number of benchmark examples for the PDAG task for each of these categories. We ran experiments to evaluate the capability of GPT-4o to solve the PDAG task for these benchmarks, which are new and have not been seen by the LLM before. We use zero-shot and one-shot prompting [Brown *et al.*, 2020; Dong *et al.*, 2024; Li, 2023], augmented with chain-of-thought reasoning [Wei *et al.*, 2022] and role-play [Kong *et al.*, 2024]. To evaluate the generated abstract planning domains and problems, we use the plan validator tool VAL [Howey *et al.*, 2004], in addition to evaluation by human experts. We also use the Fast Downward planner [Helmert, 2006] to generate plans and detect errors. The PDAG task and our benchmarks for it are inspired by the (BDL17) framework, but we do not require the LLM to generate the refinement mapping. Nor do we formally verify that the abstract PDDL domain and problem returned by the LLM is a solution to the PDAG task. We leave these topics for future work.

Our experiments show that GPT-4o can generate almost error-free results for abstraction of action/predicate parameters, but that it makes more mistakes as the difficulty of abstraction increases with abstraction of alternative concrete actions, abstraction of sequences of concrete actions, and abstraction of alternative sequences of concrete actions.

2 Background

Planning Domain Definition Language (PDDL) [McDermott *et al.*, 1998] is a widely used language for representing classical planning [Ghallab *et al.*, 2004] problems. In this paper, we use the STRIPS fragment of PDDL with typing. PDDL separates the definition of a planning problem into two parts: the *domain* definition and a *problem* definition. The *domain* definition provides a lifted representation of the relevant aspects and underlying rules of the world, and includes *types*, *predicates* and *actions*. The *problem* definition models a particular problem instance in the domain which specifies the *initial state*, *goal* condition, and the *object* names.

In-Context Learning refers to the ability of LLMs to generalize to novel tasks by interpreting examples or task descriptions provided within the input prompt, without requiring explicit task-specific training or parameter fine-tuning [Brown *et al.*, 2020; Dong *et al.*, 2024; Li, 2023]. In *one-shot* learning, the LLM is provided with a single task demonstration, along with a natural language description of the task. The example task is given together with its desired completion, and then a final task instance, i.e, user query, is provided to the LLM, and the LLM is expected to generate the completion for it. *Zero-shot* learning is similar to one-shot learning, except that no task demonstrations are provided. Here, we use zero-shot and one-shot prompts that are augmented with chain-of-thought (CoT) [Wei *et al.*, 2022] and role-play [Kong *et al.*, 2024] prompting techniques. CoT facilitates step-by-step reasoning by encouraging the model to generate intermediate reasoning steps before arriving at a final answer, thereby improving performance on commonsense and symbolic reasoning tasks. Role-Play prompting enhances task understanding by guiding the model to assume specific personas, which helps contextualize responses more effectively.

3 Planning Domain Abstraction Generation

We define the *Planning Domain Abstraction Generation* (PDAG) task as follows: given (1) a concrete planning domain in PDDL D_c, (2) a concrete planning problem instance in PDDL P_c, (3) a brief natural language description of the domain dd_c, and (4) a purpose of abstraction in natural language pa, use in-context learning with LLMs to generate (a) an abstract planning domain in PDDL D_a and (b) an abstract planning problem instance in PDDL P_a that meet the purpose of abstraction.

We can formalize the PDAG task based on the notion of sound and complete abstraction of situation calculus basic action theories (BATs) relative to a refinement mapping m of (BDL17). For this, the purpose of abstraction must be expressed as a set of constraints C_m on the refinement mapping m (these constraints restrict which fluents and actions may be included in the abstract theory and how they can be mapped to the concrete theory; see the next section for an example). First, note that for any domain D and problem P in the ADL [Pednault, 1989] fragment of PDDL, we can obtain an equivalent BAT $t(D, P) = \mathcal{D}$ as shown in [Claßen *et al.*, 2007]. Note that this BAT must have a finite object domain and that the initial situation description must be complete, i.e., it completely specifies which tuples of objects are in the extension of every fluent initially. Using this, we say that an abstract PDDL domain D_a and problem P_a is a *solution to the PDAG task* (D_c, P_c, C_m), where we have the concrete PDDL domain D_c and problem P_c and constraints on the mapping C_m, if there exists a refinement mapping m such that m satisfies C_m and $t(D_a, P_a)$ is a sound and complete abstraction of $t(D_c, P_c)$ relative to m. [Banihashemi *et al.*, 2024] identifies a set of necessary and sufficient properties for an abstract BAT $\mathcal{D}_a$ to be a sound and complete abstraction of a concrete BAT $\mathcal{D}_c$ relative to mapping m. Since the action theories involved have a finite object domain and a complete initial state specification, given a mapping m, it should be possible to check if these properties are satisfied using model checking techniques. But this requires identifying a suitable mapping m and one also must show that m satisfies the constraints C_m.

4 Abstraction of PDDL Domains and Benchmark Problems

In our experiments, we use a collection of new examples that we have developed and hence, have not been part of the training data of GPT-4o. Each example includes 4 components: a low-level PDDL domain specification, a low-level PDDL problem instance, a brief natural language description of the domain, and a brief natural language description of purpose of abstraction. We consider the following categories of abstraction tasks: abstraction of alternative concrete actions, abstraction of a sequence of concrete actions, abstraction of action/predicate parameters, and a combination of the first two, i.e., abstraction of alternative sequences of concrete actions.

Abstraction of Alternative Concrete Actions. In scenarios where the low-level domain offers multiple alternative actions to achieve a subgoal, such actions can be abstracted into a single high-level action. This is typically accompanied by the abstraction of associated predicates and types. Note that depending on the purpose of abstraction, the concrete domain, and the planning goal, some types, predicates, or actions may not need to be abstracted.

Example 1. Consider the following low-level domain which models travel arrangements. In order to go to the destination, one may book a flight or train, if a seat is available, and also book a hotel or airbnb, if a room is available.

```
(define (domain travelArrange01_LL)
(:requirements :strips :typing)
(:types
    hotel airbnb room flight trainRide seat - object)
(:predicates
   (booked_hotel  ?r -room  ?h - hotel)
   (booked_airbnb  ?r - room ?ab - airbnb)
   (available_room_hotel  ?r  - room ?h - hotel)
   (available_room_airbnb  ?r - room ?ab - airbnb)
   (bookedHotelOrAirbnb)
   (available_seat_flight ?s - seat ?f - flight)
   (available_seat_trainRide ?s -seat ?t - trainRide)
   (booked_flight ?s - seat ?f - flight)
   (booked_trainRide ?s - seat ?t - trainRide)
   (bookedFlightOrTrainRide))
(:action book_hotel
    :parameters (?h - hotel ?r - room)
    :precondition (available_room_hotel ?r ?h)
    :effect (and (booked_hotel ?r ?h)
                 (not (available_room_hotel ?r ?h))
                 (bookedHotelOrAirbnb)))
(:action book_airbnb
    :parameters (?ab - airbnb ?r - room)
    :precondition (available_room_airbnb ?r ?ab)
    :effect (and (booked_airbnb ?r ?ab)
                 (not (available_room_airbnb ?r ?ab))
                 (bookedHotelOrAirbnb)))
(:action book_flight
    :parameters (?f - flight ?s - seat)
    :precondition (available_seat_flight ?s ?f)
    :effect (and (booked_flight ?s ?f)
                 (not (available_seat_flight ?s ?f))
                 (bookedFlightOrTrainRide)))
(:action book_trainRide
    :parameters (?t - trainRide ?s - seat)
    :precondition (available_seat_trainRide ?s ?t)
    :effect (and (booked_trainRide ?s ?t)
                 (not (available_seat_trainRide ?s ?t))
                 (bookedFlightOrTrainRide))))
```

The goal in the associated problem instance is defined as follows (see [Banihashemi *et al.*, 2025] for the complete listings of concrete and abstract domains and problem instances of examples 1 to 4):

```
(:goal (and (bookedFlightOrTrainRide)
            (bookedHotelOrAirbnb)))
```

The purpose of abstraction may be expressed in natural language as "provide an abstraction of the concrete domain such that the high-level domain abstracts over booking various kinds of transportation and accommodation". This purpose of abstraction might be specified formally as a set of constraints on the mapping m, which should be entailed by the concrete action theory $\mathcal{D}_c$ obtained from the concrete PDDL domain and problem. For the booking of accommodation, we could have:

$$\exists r, h.bookedHotel(r, h, s) \vee \exists r, ab.bookedAirbnb(r, ab, s) \supset m(doneBookingAccomodation)[s]$$
$$Do(m(bookAccomodation(a, r)), s, s') \supset m(doneBookingAccomodation)[s']$$

This says that booking a hotel or an airbnb is sufficient to achieve the concrete condition into which $doneBookingAccomodation$ is mapped and that executing the program into which the abstract action $bookAccomodation(a, r)$ is mapped achieves $doneBookingAccomodation$ ($Do(\delta, s, s')$ means that there is an execution of the program δ starting in situation s and ending in situation s'). We could also have a constraint that ensures that there is no abstract action a_h that is mapped into the concrete action $bookHotel(h, r)$, i.e. $m(a_h(h, r)) = bookHotel(h, r)$, and thus this action cannot appear at the abstract level, and similarly for $bookAirbnb$. The constraints for the booking of transportation are similar.

In such a case, we would expect the LLM to generate the following high-level domain:

```
(define (domain travelArrange01_HL)
(:requirements :strips :typing)
(:types accommodation room transportation seat
 - object)
(:predicates
   (booked_accommodation  ?r - room ?a - accommodation)
   (available_room  ?r - room ?a - accommodation)
   (doneBookingAccommodation)
   (available_seat ?s - seat ?tp - transportation)
   (booked_transportation ?s - seat ?tp - transportation)
   (doneBookingTransportation))
(:action book_accommodation
    :parameters (?a - accommodation ?r - room)
    :precondition (available_room ?r ?a)
    :effect (and (booked_accommodation ?r ?a)
                 (not (available_room ?r ?a))
                 (doneBookingAccommodation)))
(:action book_transportation
   :parameters (?tp - transportation ?s - seat)
   :precondition (available_seat ?s ?tp)
   :effect (and (booked_transportation ?s ?tp)
                (not (available_seat ?s ?tp))
                (doneBookingTransportation))))
```

In the above, the types $hotel$ and $airbnb$ have been abstracted to type $accommodation$, actions $book\_hotel$ and $book\_airbnb$ to abstract action $book\_accommodation$, and predicates $booked\_hotel$ and $booked\_airbnb$ to high-level predicate $booked\_accommodation$. The type $room$ on the other hand, has not been abstracted.

The problem instance is then abstracted by using the types, predicates, and actions in the high-level domain. Objects also need to be assigned to the abstract types when necessary. ■

This category includes 8 benchmark examples, with domains modeling diverse scenarios such as apparel selection,

daily activities, household chores, and software development. The number of actions defined within these domains ranges from 3 to 6. The purpose of abstraction varies according to the domain. For example, in *SoftwareDev01*, where the low-level domain includes actions to resolve bugs off-by-one, operator precedence, wrong indentation, and unmatched parentheses, the purpose of abstraction instructs the LLM to provide an abstraction such that the high-level domain abstracts over resolving similar types of software program bugs, i.e., logical bugs and syntax bugs. In *DailyPlan01* on the other hand, where the concrete domain includes actions to read a book, watch a tutorial, cycle, hike, order food, and cook, the purpose of abstraction instructs the LLM to provide an abstraction such that the high-level domain abstracts over similar classes of daily activities.

Abstraction of Sequences of Concrete Actions. In this category, sequences of two or more low-level actions are abstracted to one high-level action. This typically also involves abstraction of conjunction of several low-level predicates into a single high-level predicate. As the previous category, some predicates or actions need not be abstracted.

Example 2. Consider the following concrete domain which models editing a file in an online editor hosted in a cloud. A user needs to first enter a valid username and then a valid matching password to login to his cloud account. Then he can open the file in an editor provided the file is originally closed and he has permission to edit it and after that, make changes to the file content.

```
(define (domain cloudApps01_LL)
...
 (:action enter_UserName
      :parameters (?u - userName)
      :precondition (valid_userName ?u)
      :effect (authenticated_userName ?u))
 (:action enter_passWord
      :parameters (?u - userName ?p - passWord)
      :precondition (and (valid_passWord ?p)
                         (authenticUserPassword ?u ?p)
                         (authenticated_userName ?u))
      :effect (authenticated_passWord ?p))
 (:action openFileInEditor
      :parameters (?f - file ?p - passWord ?u - userName)
      :precondition (and (closed_file ?f)
                         (hasEditPermission ?u ?f)
                         (authenticated_passWord ?p))
      :effect (and (openedFileInEditor ?f)
                   (not(closed_file ?f)) ))
 (:action changeFileConent
      :parameters (?f - file)
      :precondition (openedFileInEditor ?f)
      :effect (changedFileContent ?f)))
```

Now suppose that the purpose of abstraction states "provide an abstraction of the concrete domain such that the high-level domain abstracts over detailed steps of logging in and editing a file on cloud". The following high-level domain satisfies the purpose of abstraction:

```
(define (domain cloudApps01_HL)
...
(:action login
     :parameters (?u - userName ?p - passWord)
     :precondition (valid_credentials ?u ?p)
     :effect (logged_in ?u ?p))
(:action edit_file
     :parameters (?f - file ?p - passWord ?u - userName)
     :precondition (and (closed_file ?f)
                        (hasEditPermission ?u ?f)
                        (logged_in ?u ?p) )
     :effect (and (edited_file ?f)
                  (not(closed_file ?f)) ) ) )
```

Here, the sequence of actions $enter\_userName$ and $enter\_passWord$ is abstracted to $login$, and the predicate $logged\_in$ (which abstracts over the conjunction of predicates $authenticated\_userName$ and $authenticated\_passWord$) is a precondition for the abstract action $edit\_file$. Predicates $hasEditPermission$ and $closed\_file$ are retained in the abstract domain. The problem instance is then abstracted by using the types, predicates, and actions in the high-level domain. ■

This category includes 11 examples, with domains modeling scenarios such as car manufacturing, cooking, painting, order delivery, laptop purchase, and rescue robot. The number of actions defined in these domains ranges from 3 to 6.

Abstraction of Action/Predicate Parameters. This category involves modifications to the low-level PDDL domain by abstracting away one or more action/predicate parameters. Typically, this includes the removal of a PDDL type, which often results in the corresponding deletion or modification of predicates and actions that utilize this type, for instance dropping parameters of this type.

Example 3. Consider the following low-level domain which models hotel bookings. A hotel room is characterized by room type (e.g., single or double), and a room view (e.g., ocean view or garden view). It is possible to modify a room's type, e.g., convert a single room to a double room.

```
(define (domain travelArrange02_LL)
...
(:action book_hotel
  :parameters (?h - hotel ?r - room ?rv - r_view ?rt -
    ↪ r_type)
  :precondition (and (available_room_hotel ?r ?h)
                     (roomType ?r ?rt)
                     (roomView ?r ?rv))
  :effect (and (booked_hotel ?rt ?h ?rv)
               (not (available_room_hotel ?r ?h))))
(:action  change_RoomType
   :parameters (?r - room ?rt1 ?rt2 - r_type)
   :precondition (and (roomType ?r ?rt1)
                      (difRoomType ?rt1 ?rt2)
   :effect (and (roomType ?r ?rt2)
                (not (roomType ?r ?rt1))))
```

Suppose that the purpose of abstraction is defined as "provide an abstraction of the concrete domain such that the high-level domain must not include information about room view". Then we expect the LLM to abstract over the room view, and generate the following high-level domain, where the type r_view has been removed, and predicates that utilize r_view have been either adjusted or removed:

```
(define (domain travelArrange02_HL)
...
(:action book_hotel
    :parameters (?h - hotel ?r - room ?rt - r_type)
    :precondition (and (available_room_hotel ?r ?h)
                       (roomType ?r ?rt))
    :effect (and (booked_hotel ?rt ?h)
                 (not (available_room_hotel ?r ?h))))
(:action change_RoomType
    :parameters (?r - room ?rt1 ?rt2 - r_type)
    :precondition (and (roomType ?r ?rt1)
                       (difRoomType ?rt1 ?rt2))
    :effect (and (roomType ?r ?rt2)
                 (not (roomType ?r ?rt1))))
```

If instead we ask the LLM to abstract over the room type, then in addition to removing r_type and adjusting/removing the associated predicates, we also expect the LLM to eliminate the action $change\_RoomType$. The problem instance can then be adapted accordingly. ■

This category includes 10 examples with domains modeling diverse scenarios such as email composition, flight reservation, library, technical report writing, and travel (hotel) arrangements. The number of actions defined within these domains ranges from 1 to 5.

Abstraction of Alternative Sequences of Concrete Actions. In this category, the low-level domain contains multiple action sequences that provide alternative pathways to achieving a subgoal, and we require each such alternative sequence of actions to be abstracted into a single action.

Example 4. Consider a low-level domain which models alternative ways for holding a workshop at a campus: either a lecture hall is scheduled and the workshop is offered on campus, or web conferencing software is installed and the workshop is held online. Now suppose that the purpose of abstraction instructs the LLM to "abstract over delivering a workshop session". A high-level action which satisfies this purpose of abstraction may use the parameter type $teachingPlatform$ which abstracts over the disjunction of types $lectureHall$ and $webConferenceSoftware$, and a predicate $teachingCompleted$ (as an effect) that abstracts over the disjunction of the conjunction of predicates $lectureHallScheduled$ and $lecturedOnCampus$ when teaching platform is $lectureHall$, and the conjunction of predicates $installedVideoConferencing$ and $lecturedOnline$ when teaching platform refers to $webConferenceSoftware$. ■

This category includes 6 examples with domains modeling scenarios such as event planning, travel arrangements, repair robot, beverage preparation, and choosing apparel. The number of actions defined in these domains ranges from 4 to 12.

5 Implementation

5.1 Developing Prompts

Our approach relies on one-shot and zero-shot learning, combined with CoT and role-play. In the chat completions API of OpenAI's GPT-4o, prompts can be created by providing an array of messages that contain instructions for the model. It is possible to assign a different role to each message, and as a result, influence how the model might interpret the input. Here, we consider three roles: *system*, *user*, and *assistant*. A *system prompt* sets the overarching context, behavior, or persona for the responses generated. A *user prompt* contains the specific instructions or queries that a user provides to an LLM to elicit a desired response. An *assistant prompt* represents the LLM's response to user inputs, and is conditioned on the context provided by both the system and user prompts. Assistant prompts also serve as demonstrations of desired outputs when using in-context learning (e.g., providing example completions in one-shot prompts).

Abstraction of Alternative Concrete Actions: Zero-Shot Prompt. The system prompt requires the LLM to assume the role of a PDDL expert and to reason about the task in two stages: abstraction of domain and abstraction of problem instance (see [Banihashemi *et al.*, 2025] for the listings of prompts). In the first stage, it instructs the LLM to consider the purpose of abstraction and reason about generating the high-level domain ontology (types, predicates and actions), by combining or generalizing related elements, with the help of partial examples (e.g., two types 'hotel' and 'airbnb' can be combined into 'accommodation'). Note that inclusion of partial examples is necessary to guide the LLM towards the intended abstraction as our desired abstraction has not been part of any LLM's training data. In the second stage, the LLM is instructed to use the generated high-level domain components to generate the abstract problem instance. The LLM is required to follow a number of rules which include trying to minimize the number of domain ontology elements, using terminology for abstracted domain elements that preserves focus of the domain, complying with the STRIPS fragment of PDDL, and ensuring that the goal of the problem instance remains logically consistent with the abstract domain's purpose. A user's query is represented by a user prompt which includes a brief description of the low-level domain, low-level domain and problem files in PDDL, and the purpose of abstraction. Note that the partial examples in the system prompt are based on a different topic area than the PDDL domains/instances provided in user queries.

Abstraction of Alternative Concrete Actions: One-Shot Prompt. The system prompt is similar to the zero-shot approach but removes the partial examples, and instead the LLM is instructed to learn from the example provided, referred to as $Case1$. The one-shot example is modeled by a pair of user and assistant prompts. The user prompt includes a low-level domain and problem instance in PDDL, purpose of abstraction, and a brief explanation of the domain. In addition to the high-level domain and problem instance in PDDL, the assistant prompt includes a rationale for making choices of abstraction of actions, predicates and types. The user's query (referred to as $Case2$) is modelled as a user prompt like the zero-shot approach above. Note that the topic area in $Case1$ differs from that in $Case2$.

Abstraction of Sequences of Concrete Actions. This category only uses one-shot prompting, as preliminary experiments with zero-shot prompts produced unsatisfactory results. The system prompt directs the LLM through a two-stage reasoning process: domain abstraction followed by problem instance abstraction, guided by Case1 and predefined rules which are mainly similar to the previous category. The LLM is asked to consider the purpose of abstraction and identify sequences of actions that should be merged into high-level actions that achieve the same effect but eliminate unnecessary intermediate steps. The one-shot example and user query are modelled similarly to the one-shot approach in the previous category.

Abstraction of Action/Predicate Parameters. This category only uses a zero-shot prompting approach, as it proved sufficient for generating mostly correct results. The system prompt instructs the LLM to consider the purpose of abstraction and decide whether various domain elements (parame-

ters, types, predicates, actions) must be retained or eliminated in the abstract domain. The user query for this category is defined similar to the previous zero-shot prompt.

Abstraction of Alternative Sequences of Concrete Actions. Here we restrict our analysis to one-shot prompts, as initial experiments with zero-shot prompts produced unsatisfactory outcomes. The system prompt is structured similar to previous categories' system prompt for one-shot learning, however, it is more generic and includes abstraction instructions which combine elements from all previous categories The user query and one-shot demonstration are modelled similar to previous categories.

5.2 Evaluating the Generated Models

We use a hybrid evaluation approach that incorporates both human evaluation and automated tools. Our method first uses VAL [Howey *et al.*, 2004] to check for syntax errors in the generated high-level domain and problem instance, and then calls the Fast Downward planner [Helmert, 2006] to try to generate a plan.

Next, the generated high-level domain and problem instance are reviewed by human domain experts. Each generated high-level domain undergoes a comparative analysis with a sample abstract domain (already created by a knowledge engineer) which represents a sound abstraction of the low-level domain in user's query, and it is part of the solution to the PDAG task. This evaluation considers the correctness of abstract actions, predicates, types, and parameters generated, as well as the removal of actions, predicates, types, and parameters from the high-level domain that are considered unnecessary details wrt the purpose of abstraction. Concrete domain elements that must be retained in the generated abstract domain are also considered. Syntax errors are also noted. High-level problem instances are evaluated similarly.

Human expert evaluation is essential as there may be more than one correct approach to abstract a domain. E.g., in case of "CookFood01" when abstracting over sequence of low-level actions $wash, chop, marinate, grill$ using two high-level abstract actions $prepare$ and $cook$, one solution may abstract sequence of concrete actions $wash, chop$ to action $prepare$, while another solution may consider $wash, chop, marinate$ as refinement of action $prepare$. Moreover, the LLM often generates names for actions/predicates/types that may be different from those in the sample solution provided. So this part of the evaluation would be difficult to automate.

As our aim in this paper is to evaluate the feasibility of using advanced LLMs to generate abstract planning domain and problem files based on an initial input prompt, we do not consider providing corrective feedback from users or validation tools to the LLM to allow it to fix errors and re-generate high-level domains and/or problem files.

5.3 System Architecture

Fig. 1 shows the outline of our system. Initially, a system prompt (and possibly a user/assistant prompt in case of one-shot in-context learning) is provided to the LLM (GPT-4o) which describes the task, rules to follow, expected input and desired output. The user then provides his query which consists of a low-level PDDL domain and its brief description, PDDL problem instance, and the purpose of abstraction. After the LLM processes the query, it generates a response which includes a high-level domain and a high-level problem instance in PDDL as well as a natural language description of the justification for the specific choices of abstraction made.

Our code then extracts the generated high-level PDDL domain and problem instance as .pddl files and sends them to the validation module (which includes VAL and the Fast Downward Planner). The validation results and a plan in case of success, or error messages in case of failure, a summary report, in addition to the generated PDDL domain and problem instance as well as the natural language description of the justification for the specific choices of abstraction are then saved to a datastore. A human evaluator then reviews the stored information and saves the results of analysis to the datastore. Note that we don't *formally verify* that the LLM generated high-level domain/problem instance is a sound abstraction and satisfies the constraints on the mapping associated with the purpose of abstraction; this is left for future work.

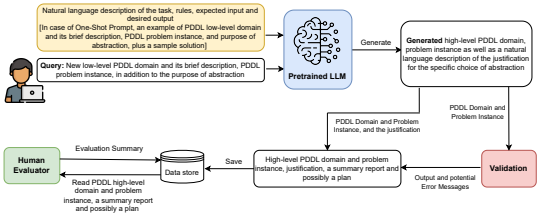

Figure 1: Generating Abstract PDDL Domain and Problem Instance

6 Empirical Evaluation

Each of our benchmark examples within each category was run five times. Tables 1 to 5 present the aggregated results. The table columns show the following metrics: **Changes Needed (CN):** correctness score of predicates/actions/types intended to be abstracted or eliminated as evaluated by human experts (e.g., generalizing *hotel* and *airbnb* to *accommodation*). **CN-SD:** standard deviation of CN scores. **Avoid Unnecessary Changes (AUC):** correctness score of predicates/actions/types/objects that were expected to remain unchanged in the high-level domain and problem instance as evaluated by human domain experts (e.g., keeping the type *room* in the abstract domain and problem instance in Example 1). **HDE, VAL, and FD** indicate if syntax errors were present and whether they were detected by human domain experts (HDE), VAL, or the Fast Downward planner (FD). If the planner fails to generate a plan, we count this as an FD error.

Our experiment results show that GPT-4o can generate almost error-free results for abstraction of action/predicate parameters, but as the difficulty of abstraction increases with abstraction of alternative concrete actions, sequences of concrete actions, and alternative sequences of concrete actions, the quality of generated domains/problem instances declines and the number of syntax errors increases. We observed that

	CN (Avg)	CN-SD	AUC (Avg)	AUC-SD	HDE (Count)	FD (Count)	VAL (Count)
DailyPlan01	75.11%	13.8	100%	0	2	0	2
DailyPlan02	85.07%	5.5	98.46%	3.08	3	3	3
HouseHold01	100%	0	100%	0	0	0	0
ClothesShop01	88.46%	14.13	74.55%	31.18	0	0	0
SoftwareDev01	94.28%	6.29	100%	0	0	0	0
SoftwareDev02	90.95%	4.86	97.71%	2.8	0	0	0
SoftwareDev03	96%	2.42	91.11%	4.44	0	0	0
SoftwareDev04	97.88%	1.73	100%	0	0	0	0

Table 1: Abstraction of Alternative Concrete Actions - Zero-Shot

	CN (Avg)	CN-SD	AUC (Avg)	AUC-SD	HDE (Count)	FD (Count)	VAL (Count)
DailyPlan01	93.19%	2.08	100%	0	2	2	2
DailyPlan02	100%	0	100%	0	0	0	0
HouseHold01	100%	0	100%	0	0	0	0
ClothesShop01	100%	0	100%	0	0	0	0
SoftwareDev01	100%	0	100%	0	0	0	0
SoftwareDev02	87.14%	11.82	68%	27.01	1	1	0
SoftwareDev03	98.12%	2.31	100%	0	2	0	2
SoftwareDev04	100%	0	100%	0	0	0	0

Table 2: Abstraction of Alternative Concrete Actions - One-Shot

the majority of mistakes were in generating new predicates which abstract over concrete predicates. We also noticed that changing the order of action declarations or including additional constructs (actions, types, objects, predicates) which are irrelevant to the goal of the task do not cause additional errors. In general, the LLM was able to name abstracted actions/predicates/types reasonably well; however, in some cases, names for high-level domain constructs were incorrect. E.g., in *DailyPlan01*, the abstraction of alternative concrete actions *orderFood* and *cook* was named *eat*. See [Banihashemi *et al.*, 2025] for a more detailed discussion on evaluation results.

7 Related Work

Several authors have studied using LLMs to generate PDDL specifications given natural language descriptions of the task. [Liu *et al.*, 2023] assumes that a PDDL domain description and a contextual example demonstrating the conversion of a natural language problem within the domain into a PDDL problem are provided by human experts, and the LLM is tasked with generation of the PDDL specification of a problem instance (based on the domain) given a natural language description of the problem. [Xie *et al.*, 2023] studies extracting a planning goal given a natural language instruction, by using zero-shot and few-shot prompting. [Guan *et al.*, 2023] follows a strategy where few-shot prompting is used to generate a PDDL representation of a single action at a time, while dynamically updating a list of predicates, and ultimately repeating the process with all the extracted predicates. [Oswald *et al.*, 2024] also uses in-context learning to generate PDDL domains on an action-by-action basis using context examples from other domains. The query in the input prompt includes the allowed predicates that can be used in the definition of that action and a natural language description of the action. The above approaches do not involve generating abstractions of low-level PDDL domains or problem instances. To address complex sequential decision-making, [Liu *et al.*, 2024] introduces SkillAct, a prompting method that integrates reusable, task-relevant skill descriptions into prompts. Skills abstract high-level behaviors from agent trajectories (observation-action sequences) and are derived from the LLM's embedded world knowledge via prompting. The skills/generated outputs are not formalized in PDDL.

	CN (Avg)	CN-SD	AUC (Avg)	AUC-SD	HDE (Count)	FD (Count)	VAL (Count)
CookFood01	79.2%	6.01	88.61%	13.41	0	0	0
CookFood02	72.4%	7.2	95.45%	9.09	0	0	0
CarManufacturing01	48.04%	9.71	92.68%	14.63	2	2	2
CleanItem02	71.59%	7.65	100%	0	2	1	1
LaptopShop02	65.15%	15.15	83.64%	13.36	0	0	0
DeliveryRobot01	78%	15.86	80.47%	10.36	2	1	2
DeliveryRobot02	86.21%	0	92.45%	0	0	0	0
DeliveryRobot03	100%	0	94.67%	2.67	0	0	0
DeliveryRobot04	92.55%	5.17	89.33%	9.98	1	1	1
RescueRobot02	89.92%	4.96	96.42%	5.22	0	0	0
RescueRobot03	87.5%	5.24	92.54%	5.26	1	1	1

Table 3: Abstraction of Sequences of Concrete Actions - One-Shot

	CN (Avg)	CN-SD	AUC (Avg)	AUC-SD	HDE (Count)	FD (Count)	VAL (Count)
Email01-0	100%	0	100%	0	0	0	0
Email01-1	100%	0	100%	0	0	0	0
Flight01	100%	0	100%	0	0	0	0
Flight02	100%	0	100%	0	0	0	0
Library01-1	100%	0	100%	0	0	0	0
Library01-2	96.92%	6.15	96.10%	7.62	0	0	0
TechReport01	100%	0	100%	0	0	0	0
TechReport02	100%	0	100%	0	0	0	0
Travel02	100%	0	100%	0	0	0	0
Travel03	100%	0	100%	0	0	0	0

Table 4: Abstraction of Action/Predicate Parameters - Zero Shot

	CN (Avg)	CN-SD	AUC (Avg)	AUC-SD	HDE (Count)	FD (Count)	VAL (Count)
BeveragePreparation01	43.92%	27.79	100%	0	9	1	9
ClothesShop02	65.37%	13.59	100%	0	0	0	0
EventPlanning01	56.62%	1.5	96.1%	2.46	13	3	14
RepairRobot04	67.03%	5.21	87.06%	7.8	3	3	3
RoomRedesign01	69.6%	3.32	100%	0	12	3	12
Travel05	62.14%	11.85	100%	0	10	2	11

Table 5: Abstraction of Alternative Sequences of Concrete Actions

8 Conclusion and Future Work

In this paper, we investigated the feasibility of using an LLM (GPT-4o) to generate high-level PDDL domains and problem instances from low-level PDDL representations, guided by a specified abstraction purpose.

In future work, we plan to extend our set of examples and study additional abstraction categories. We will also consider using the more expressive ADL [Pednault, 1989] fragment of PDDL. Building a dataset of planning examples that can be used for fine-tuning an LLM is another avenue for further research. We are also interested in designing other types of prompts that could enhance the generated results. We will also examine extending the current abstraction task to incorporate the generation of a refinement mapping so that one can then check that the output high-level model is a sound abstraction of the low-level model relative to this mapping. A significant area for future work lies in the development of (partially) automated validation mechanisms to ensure the correctness of the abstracted domains and problems generated by LLMs. More work on the formalization of the abstraction purpose and automated mechanisms that check the generated models' adherence to it is also indicated.

Acknowledgements

This work is partially supported by the National Science and Engineering Research Council of Canada and by York University.

References

[Bacchus and Kabanza, 2000] Fahiem Bacchus and Froduald Kabanza. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence*, 116(1-2):123–191, 2000.

[Banihashemi *et al.*, 2017] Bita Banihashemi, Giuseppe De Giacomo, and Yves Lespérance. Abstraction in situation calculus action theories. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, pages 1048–1055. AAAI Press, 2017.

[Banihashemi *et al.*, 2024] Bita Banihashemi, Giuseppe De Giacomo, and Yves Lespérance. Abstracting situation calculus action theories. *CoRR*, abs/2410.14712, 2024.

[Banihashemi *et al.*, 2025] Bita Banihashemi, Megh Patel, and Yves Lespérance. Using large language models for abstraction of planning domains - Extended Version. To appear on arXiv, 2025.

[Brown *et al.*, 2020] 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. *CoRR*, abs/2005.14165, 2020.

[Chen and Bercher, 2021] Dillon Chen and Pascal Bercher. Fully observable nondeterministic HTN planning - formalisation and complexity results. In *Proceedings of the Thirty-First International Conference on Automated Planning and Scheduling, ICAPS 2021*, pages 74–84. AAAI Press, 2021.

[Claßen *et al.*, 2007] Jens Claßen, Patrick Eyerich, Gerhard Lakemeyer, and Bernhard Nebel. Towards an integration of golog and planning. In *IJCAI*, pages 1846–1851, 2007.

[De Giacomo *et al.*, 2000] Giuseppe De Giacomo, Yves Lespérance, and Hector J. Levesque. ConGolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence*, 121(1-2):109–169, 2000.

[Dong *et al.*, 2024] Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Zhiyong Wu, Baobao Chang, Xu Sun, Jingjing Xu, Lei Li, and Zhifang Sui. A survey on in-context learning. *CoRR*, abs/2301.00234, 2024.

[Dong *et al.*, 2025] Hao Dong, Zheyuan Shi, Hemeng Zeng, and Yongmei Liu. An automatic sound and complete abstraction method for generalized planning with baggable types. In *AAAI-25, Sponsored by the Association for the Advancement of Artificial Intelligence*, pages 14875–14884. AAAI Press, 2025.

[Erol *et al.*, 1996] Kutluhan Erol, James A. Hendler, and Dana S. Nau. Complexity results for HTN planning. *Annals of Mathematics and Artificial Intelligence*, 18(1):69–93, 1996.

[Fang *et al.*, 2025] Liangda Fang, Xiaoman Wang, Zhang Chen, Kailun Luo, Zhenhe Cui, and Quanlong Guan. A syntactic approach to computing complete and sound abstraction in the situation calculus. In *AAAI-25, Sponsored by the Association for the Advancement of Artificial Intelligence*, pages 14911–14921. AAAI Press, 2025.

[Fox and Long, 2011] Maria Fox and Derek Long. PDDL2.1: an extension to PDDL for expressing temporal planning domains. *CoRR*, abs/1106.4561, 2011.

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

[Guan *et al.*, 2023] 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 *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023*, 2023.

[He *et al.*, 2023] Weinan He, Canming Huang, Zhanhao Xiao, and Yongmei Liu. Exploring the capacity of pretrained language models for reasoning about actions and change. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2023*, pages 4629–4643. Association for Computational Linguistics, 2023.

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

[Howey *et al.*, 2004] 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 (ICTAI 2004)*, pages 294–301. IEEE Computer Society, 2004.

[Hurst *et al.*, 2024] Aaron Hurst, Adam Lerer, Adam P. Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, Aleksander Madry, Alex Baker-Whitcomb, Alex Beutel, Alex Borzunov, Alex Carney, Alex Chow, Alex Kirillov, Alex Nichol, Alex Paino, Alex Renzin, Alex Tachard Passos, Alexander Kirillov, Alexi Christakis, Alexis Conneau, Ali Kamali, Allan Jabri, Allison Moyer, Allison Tam, Amadou Crookes, Amin Tootoonchian, Ananya Kumar, Andrea Vallone, Andrej Karpathy, Andrew Braunstein, Andrew Cann, Andrew Codispoti, Andrew Galu, Andrew Kondrich, Andrew Tulloch, Andrey Mishchenko, Angela Baek, Angela Jiang, Antoine Pelisse, Antonia Woodford, Anuj Gosalia, Arka Dhar, Ashley Pantuliano, Avi Nayak, Avital Oliver, Barret Zoph, Behrooz Ghorbani, Ben Leimberger, Ben Rossen, Ben Sokolowsky, Ben Wang, Benjamin Zweig, Beth Hoover, Blake Samic, Bob McGrew,

Bobby Spero, Bogo Giertler, Bowen Cheng, Brad Lightcap, Brandon Walkin, Brendan Quinn, Brian Guarraci, Brian Hsu, Bright Kellogg, Brydon Eastman, Camillo Lugaresi, Carroll L. Wainwright, Cary Bassin, Cary Hudson, Casey Chu, Chad Nelson, Chak Li, Chan Jun Shern, Channing Conger, Charlotte Barette, Chelsea Voss, Chen Ding, Cheng Lu, Chong Zhang, Chris Beaumont, Chris Hallacy, Chris Koch, Christian Gibson, Christina Kim, Christine Choi, Christine McLeavey, Christopher Hesse, Claudia Fischer, Clemens Winter, Coley Czarnecki, Colin Jarvis, Colin Wei, Constantin Koumouzelis, and Dane Sherburn. Gpt-4o system card. *CoRR*, abs/2410.21276, 2024.

[Kong *et al.*, 2024] Aobo Kong, Shiwan Zhao, Hao Chen, Qicheng Li, Yong Qin, Ruiqi Sun, Xin Zhou, Enzhi Wang, and Xiaohang Dong. Better zero-shot reasoning with role-play prompting. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), NAACL 2024*, pages 4099–4113. Association for Computational Linguistics, 2024.

[Li, 2023] Yinheng Li. A practical survey on zero-shot prompt design for in-context learning. In *Proceedings of the 14th International Conference on Recent Advances in Natural Language Processing, RANLP 2023*, pages 641–647. INCOMA Ltd., Shoumen, Bulgaria, 2023.

[Liu *et al.*, 2023] 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.

[Liu *et al.*, 2024] Anthony Zhe Liu, Jongwook Choi, Sungryull Sohn, Yao Fu, Jaekyeom Kim, Dong-Ki Kim, Xinhe Wang, Jaewon Yoo, and Honglak Lee. SkillAct: Using skill abstractions improves LLM agents. In *ICML 2024 Workshop on LLMs and Cognition*, 2024.

[Luo *et al.*, 2020] Kailun Luo, Yongmei Liu, Yves Lespérance, and Ziliang Lin. Agent abstraction via forgetting in the situation calculus. In *ECAI 2020 - 24th European Conference on Artificial Intelligence*, volume 325 of *Frontiers in Artificial Intelligence and Applications*, pages 809–816. IOS Press, 2020.

[Luo, 2023] Kailun Luo. Automated verification of propositional agent abstraction for classical planning via CTLK model checking. In *Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2023*, pages 6475–6482. AAAI Press, 2023.

[McCarthy and Hayes, 1969] J. McCarthy and P. J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence*, 4:463–502, 1969.

[McDermott *et al.*, 1998] Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. PDDL – the planning domain definition language. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, 1998.

[Oswald *et al.*, 2024] James T. Oswald, Kavitha Srinivas, Harsha Kokel, Junkyu Lee, Michael Katz, and Shirin Sohrabi. Large language models as planning domain generators. In *Proceedings of the Thirty-Fourth International Conference on Automated Planning and Scheduling, ICAPS 2024*, pages 423–431. AAAI Press, 2024.

[Pednault, 1989] Edwin P. D. Pednault. ADL: exploring the middle ground between STRIPS and the situation calculus. In *Proceedings of the 1st International Conference on Principles of Knowledge Representation and Reasoning (KR'89)*, pages 324–332. Morgan Kaufmann, 1989.

[Reiter, 2001] Ray Reiter. *Knowledge in Action. Logical Foundations for Specifying and Implementing Dynamical Systems*. The MIT Press, 2001.

[Seegebarth *et al.*, 2012] Bastian Seegebarth, Felix Müller, Bernd Schattenberg, and Susanne Biundo. Making hybrid plans more clear to human users - A formal approach for generating sound explanations. In *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling, ICAPS 2012*. AAAI, 2012.

[Sutton *et al.*, 1999] Richard S. Sutton, Doina Precup, and Satinder Singh. Between MDPs and Semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1-2):181–211, 1999.

[Valmeekam *et al.*, 2024] Karthik Valmeekam, Kaya Stechly, Atharva Gundawar, and Subbarao Kambhampati. Planning in strawberry fields: Evaluating and improving the planning and scheduling capabilities of LRM o1. *CoRR*, abs/2410.02162, 2024.

[Wei *et al.*, 2022] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022*, 2022.

[Xie *et al.*, 2023] 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.

[Zhao *et al.*, 2023] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie, and Ji-Rong Wen. A survey of large language models. *CoRR*, abs/2303.18223, 2023.