Bridging Natural Language and Automated Planning: Generating PDDL Problem Descriptions from Linguistic Instructions and Scene Descriptions with LLMs

Xiaomeng Huang
Faculty Electrical Engineering and Information Technology
Aachen, Germany
xiaomeng.huang@alumni.fh-aachen.de

***Abstract*—Effective human-robot interaction is crucial for enabling robots to perform complex tasks in real-world environments. Large Language Models (LLMs) can help bridge the gap between human instructions in natural language and robotic task planning. However, directly generating plans from user instructions using LLMs may lead to a lack of interpretability due to the uncertainty of their outputs. The Planning Domain Definition Language (PDDL), widely used in robotic planning, offers the advantage of interpretability. This work proposes a method that utilizes LLMs to generate problem definitions in PDDL, allowing automated planners to generate task plans based on natural language instructions and scene descriptions. To enhance reliability and interpretability, the generation process is systematically divided into subprocesses, making error detection and correction feasible. The proposed method is evaluated using both offline and online LLM models, demonstrating that it is workable when leveraging a powerful model.**

***Index Terms*—Large Language Models (LLMs), Natural language processing (NLP), Planning Domain Definition Language (PDDL), Robotic Task Planning, Human Robot Interaction (HRI)**

I. Introduction

Effective interaction between humans and robots is essential for enabling robots to perform complex tasks in real-world environments. Large Language Models (LLMs) have emerged as powerful tools for facilitating this interaction by allowing natural language instructions to be processed. However, the black-box nature of LLMs raises concerns about interpretability, which is crucial for gaining user trust and understanding the decision-making process of robotic systems. [1]

Robot task planning has traditionally been addressed through symbolic planning, where modern automated planners rely on the Planning Domain Definition Language (PDDL) to formally describe planning problems, which offers the advantage of interpretability. [2] A PDDL-based planning problem consists of two key components: the domain definition and the problem definition. The domain definition specifies state variables and available actions, which are typically determined by the environment and the robot's capabilities. In contrast, the problem definition includes task-specific elements such as relevant objects, their initial states, and the desired goal state. These components vary depending on the task.

In this work, LLMs are employed to generate problem definitions in PDDL, enabling automated planners to create plans based on human instructions and scene descriptions, which allows non-experts to specify tasks intuitively. To enhance reliability, the generation process is systematically decomposed into subprocesses. This structured approach ensures greater transparency and robustness in robotic task planning, making LLM-driven planning more interpretable and dependable.

The remainder of this paper is structured as follows. The section II. BACKGROUND provides fundamental introductions to PDDL and LLMs. The section III. RELATED WORK reviews existing studies in this domain. The section IV. METHOD presents a detailed decomposition of the proposed approach. The section V. IMPLEMENTATION outlines its realization, illustrated with pseudocode. The section VI. EVALUATION assesses the method's effectiveness using different LLM models. Finally, the section VII. CONCLUSION summarizes the findings, discusses the limitations of the approach, and explores potential directions for future research.

II. Background

A. PDDL

Planning Domain Definition Language (PDDL) is a standardized language designed to describe automated planning problems. It provides a formal structure for defining planning tasks, enabling planners to generate sequences of actions that achieve a specified goal. [2]

1) Structure of PDDL: A PDDL planning problem consists of two key components: Domain Definition and Problem Definition.

a) Domain Definition: The domain definition describes the general rules governing a planning problem, including available actions, object types, and how the environment changes in response to actions. It consists of the following elements:

- Types
  Defines categories of objects in the environment (e.g., `robot`, `room`, `object`).
- Predicates
  Represents conditions that describe the current state of the world (e.g., `(at robot room1)` means the `robot` is in `room1`).
- Actions
  Specifies executable operations, each defined by:
  - Preconditions: Conditions that must be met before an action can be executed.
  - Effects: Changes in the environment resulting from the action.

b) Problem Definition: The problem definition specifies a concrete planning task within a given domain. It includes:

- Objects
  The specific instances of the defined types (e.g., a particular robot or room).
- Initial State
  The starting conditions of the planning problem, describing the current state of all relevant objects.
- Goal State
  The desired outcome that the planner must achieve by executing a sequence of actions.

2) Usage in Automated Planning: Once a planning problem is defined in PDDL, an automated planner processes it in two main steps:

- Parsing
  The planner first analyzes the PDDL domain and problem definitions to understand the available actions, object relationships, and constraints.
- Solving
  Using search and optimization techniques (e.g., breadth-first search, depth-first search, heuristic search, A* algorithm), the planner generates a valid sequence of actions that transitions the system from the initial state to the goal state.

Figure 1 illustrates the automated planning system. This study assumes that the domain definition is already provided by Keisuke *el al.* [3], and focuses on generating the problem definition. In the section VI. EVALUATION, an automated planner is used to verify whether a valid plan can be successfully generated.

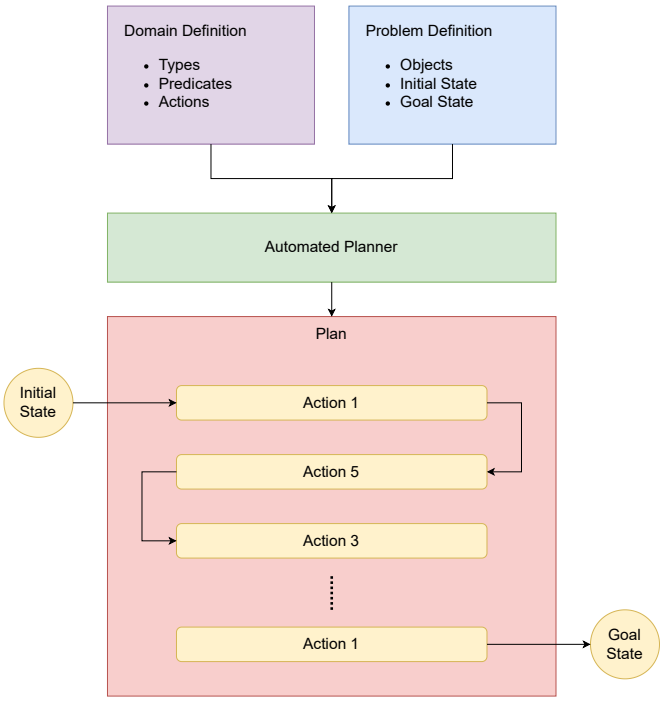

Figure 1: Automated Planning System; An automated planner takes two inputs: a domain definition and a problem definition. Based on these inputs, it generates a plan, which is a sequence of actions that transitions the system from the initial state to the goal state. [4]

B. LLMs

Large Language Models (LLMs) are advanced deep learning models trained on massive text datasets to understand and generate human-like language. These models are designed to perform a wide range of natural language processing (NLP) tasks, including text completion, summarization, and question answering. [5][6] By leveraging vast amounts of linguistic data, LLMs can generate coherent and contextually relevant responses, making them useful for text-based applications. [7][8]

1) Mechanisms: LLM is based on two fundamental mechanisms:

- Probabilistic Token Prediction
  At their core, LLMs function by predicting the most likely next token (word, subword, or character) in a sequence based on prior context. This process is driven by statistical probability: given a sequence of input tokens, the model assigns probabilities to possible next tokens and selects the one with the highest likelihood. For example, in the partial sentence "The sun rises in the . . .", an LLM will likely predict "east" because this token has the highest probability based on linguistic patterns learned during training. This probabilistic nature enables LLMs to generate diverse yet contextually relevant outputs. [9]
- Transformer Architecture
  Modern LLMs are built on the Transformer architecture, which allows them to efficiently process and generate text. A key component of Transformers is the self-attention mechanism, which enables the model to analyze relationships between all words in a sequence simultaneously. This allows LLMs to capture both local and long-range dependencies in text, ensuring a more coherent and context-aware understanding. Additionally, Transformers employ positional encoding, which helps the model retain word order information, ensuring that generated text follows proper syntactic and semantic structures. [10]

2) Approaches of Utilizing: LLMs can be leveraged in two primary ways: Fine-Tuning and Prompt Engineering. [11]

a) Fine-Tuning: Fine-tuning is a process in which a pre-trained LLM is further trained on a smaller, task-specific dataset to adapt its behavior to a particular application.

This additional training allows the model to specialize in domain-specific language, improving accuracy for tasks such as legal document generation. However, this approach comes with several challenges. It requires substantial computational resources, often necessitating high-performance GPUs or TPUs to handle the training process efficiently. Additionally, fine-tuning demands a well-curated dataset, as poor-quality or insufficient data can lead to degraded model performance. Another limitation is its lack of flexibility, as a model fine-tuned for a specific task may not generalize well to other applications without additional retraining. [12][13][14][15]

b) Prompt Engineering: Prompt engineering is an alternative approach to utilizing LLMs, where carefully designed input prompts guide the model's output without modifying its internal parameters. By structuring prompts effectively, users can influence the model's behavior to generate responses that align with specific requirements.

The advantages of prompt engineering include [16]:

- No additional training
  Since it does not require updating model parameters, it avoids the computational cost associated with fine-tuning.
- High adaptability
  A single model can be used for multiple tasks simply by modifying the input prompt, making it more flexible than fine-tuned approaches.
- Efficient implementation
  Prompt-based methods allow users to leverage LLMs without the need for specialized machine learning expertise, making them accessible for a wide range of applications.

By crafting effective prompts, users can control the model's behavior and improve the quality of its responses, making this method particularly useful for structured output generation tasks.

3) In Context Learning (ICL): An effective form of prompt engineering is In Context Learning (ICL). Unlike fine-tuning, ICL does not modify the model's parameters. Instead, it leverages the LLM's ability to recognize patterns by providing a series of input-output examples within the prompt itself. [17]

In ICL, a structured prompt is constructed with three key components:

- Task Description, which specifies the goal
- In-Context Examples, which provide a set of input-output pairs demonstrating the expected transformation
- New Input, which is appended at the end

The model infers the underlying pattern from the examples and applies it to generate the correct output for the new input. For instance, given a task as shown in Figure 2 where a date in "YYYY-MM-DD" format must be reformatted into "Month Day Year" with exclamation marks, an LLM can learn the transformation simply by observing a series of annotated examples.

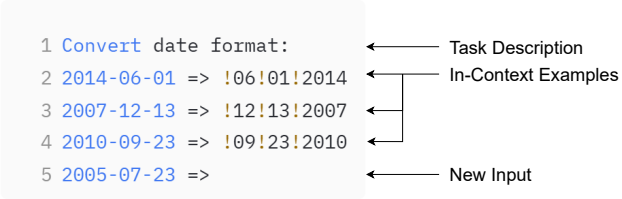

Figure 2: Example task used to explain how ICL works; The Task Description defines the objective: converting a date from "YYYY-MM-DD" format to "Month Day Year" with exclamation marks. The In-Context Examples provide several input-output pairs, demonstrating the correct transformation pattern. Finally, the Prompt presents a new input date at the end. The model should recognize the pattern from the examples, and apply the learned transformation to generate the correctly formatted output

This study leverages ICL's effective pattern recognition capability to generate PDDL descriptions. By providing a well-designed set of examples, the model can recognize and replicate the structured format of PDDL, ensuring accurate and consistent outputs. A detailed explanation of how ICL is applied in this context will be provided in Section IV. METHOD.

III. Related Work

Task planning using natural language has been widely studied, with various approaches proposed to bridge the gap between linguistic instructions and symbolic actions.

One common approach involves converting natural language instructions into symbolic actions using neural networks [18], [19]. Neural networks are generally efficient, requiring less computing power while being optimized for specific tasks.

For example, Paxton *et al.* introduced the DREAMCELL architecture [18], which integrates natural language and sensor data (including RGB, pose, and depth images) to predict future states and generate intermediate goals for task execution. By training the model end-to-end with simulated data, the system learns an interpretable representation for planning. Experiments in a simulated block-stacking task demonstrate its effectiveness in generating accurate plans.

However, neural networks often lack flexibility, as they typically require retraining when applied to new tasks and struggle with complex or ambiguous instructions. To address the limitations, recent studies have explored the use of LLMs for task planning [20]–[27]. LLMs offer greater adaptability, as they can comprehend a wide range of instructions without requiring retraining. They can also handle ambiguous or underspecified instructions by leveraging contextual information to infer user intent.

For instance, Ding *et al.* proposed LLM-GROP, a framework designed for multi-object rearrangement tasks [22]. The system takes user requests as input, utilizes LLMs to infer

symbolic and geometric spatial relationships between objects, and employs the Grounded Robot Task and Motion Planning (GROP) module to compute optimal task-motion plans. Evaluations on table-setting tasks demonstrate that LLM-GROP achieves the highest user ratings and shortest execution times compared to baseline methods, and it has also been successfully deployed on a real robot.

Despite their advantages, LLMs exhibit notable limitations, particularly in long-term reasoning for complex tasks. When confronted with intricate planning scenarios, LLM-generated outputs often fail to produce executable plans that successfully accomplish the given task [28].

Huang *et al.* identified this issue and proposed improvements such as semantic translation of actions and dynamic example selection to enhance plan executability [24]. Experiments conducted in the VirtualHome environment indicate that their approach significantly improves execution success rates, though with occasional trade-offs in correctness.

Building on this direction, this study introduces a framework that leverages the PDDL to enhance the executability of generated plans. As discussed in the previous section, structuring problem descriptions strictly according to PDDL makes them both human-readable and machine-interpretable. Additionally, since PDDL descriptions consist of multiple structured components, errors can be effectively localized and corrected. By decomposing the generation of PDDL-based problem descriptions into distinct steps, this framework aims to reduce the unpredictability of LLM outputs and enhance the reliability of translating natural language instructions into executable plans.

IV. Method

A. Problem Formulation

In this work, Blocksworld, a classic domain in AI planning, is chosen as the manipulation environment to illustrate the proposed framework. In Blocksworld, a robot is tasked with rearranging a set of blocks from an initial configuration to a target configuration. As shown in Figure 3, the positions of blocks A, B, C and D should to be changed to transition from the initial state to goal state.

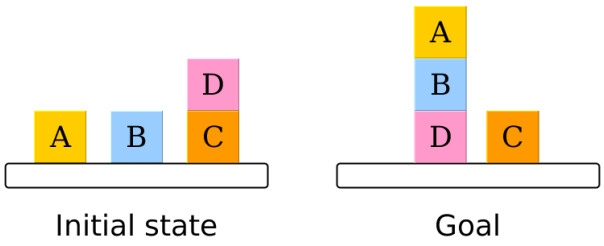

Figure 3: Blocksworld example [29]

The objective of this work is to combine linguistic instructions and descriptions of the current scene, with relevant background information, to generate a structured problem description containing objects, initial states, and specified goals. The overall system workflow is depicted in Figure 4 and consists of four parts:

- **Input**
  The input consists of two elements, (L, S), where L represents linguistic instructions and S represents scene description.
- **Context**
  The context is composed of two components, (D_D, D_E), where D_D represents the parts that are common to all problems in the scenario, and D_E contains concrete examples used to facilitate learning by the LLM.
- **Output**
  The output includes three elements, (O, I, G), where O denotes the objects present in the scene, I represents the initial state, and G defines the specified goal state.
- **Process**
  The process leverages the learning capabilities of an LLM to process the input and generate the desired output.

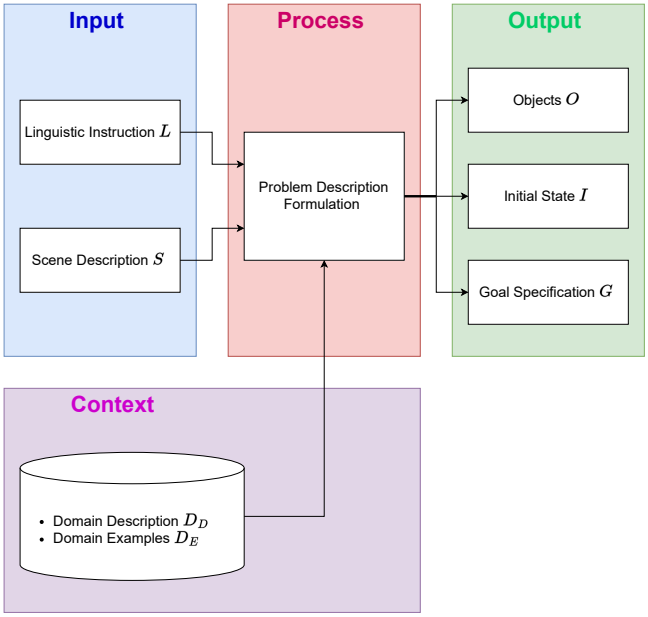

Figure 4: System Workflow; Instruction L, which contains user instructions, and Scene Description S, which contains a description of the environment, are used as inputs, and the system generates, with the help of a description of the domain D_D and some examples D_E, a problem description as output, containing a list of objects O, an initial state I, and a description of the goal G.

Further details on the Input, Context, Output are provided in Section IV.*B. Input, Context, Output*, while the Process is explained in Section IV.*C. Process*. The goal of this work is to establish a mechanism $M : (L, S) \xrightarrow{D_D, D_E} (O, I, G)$ where the generated outputs are expected to be machine-readable and suitable for automated plan generation.

B. Input, Context, Output

Within the input, L represents a sequence of words describing the tasks to be performed, such as specifying the location or relative arrangement of blocks. An example of L is provided in Code Block 1. S is a structured description that contains information about objects detected in an image. The image may be captured by a robot's vision sensor or provided manually by a human. The description includes attributes such as the color of a block (e.g., `green block`) and its coordinates within the image. These coordinates are represented as bounding box values (e.g., `[122.13333333333333, 268.36656084656084, 164.0774603174603, 316.8931216931217]`, where the first pair denotes the upper-left corner and the second pair denotes the lower-right corner of the bounding box. Code Block 2 provides an example of S formatted in the json language.

Code Block 1: Example code block of Input L

```
Create a stack of block:
pink over red over yellow over green
```

Code Block 2: Example code block of Input S

```
{
   "green_block": {
      "bbox": [
         122.13333333333333,
         268.36656084656084,
         164.0774603174603,
         316.8931216931217
      ],
      "phrase": "green block",
      "caption": "it is a cube"
   },
}
```

D_D, corresponding to the Domain Definition introduced in Section II. *A. PDDL*, defines the common elements shared across all tasks in this domain: object types (e.g., `block`, `robot`), predicates (e.g., `ontable`, indicating that an object is located on the table), and symbolic actions (e.g., `pick-up`, `put -down`, `stack`, `unstack`). D_E provides examples that show the correct output for the current input. These examples serve as references for LLM to learn how to build mapping from input to output.

O lists the objects involved in the task (e.g., `green_block - block`, indicating that a green block belongs to the `block` type). I represents the initial state of the environment and is described using predicates with arguments (e.g., `(ontable gree_block - block)`, which signifies that the green block is placed on the table). G specifies the desired goal state as a set of logical conditions. For example, `(and (on pink_block red_block)(on red_block yellow_block)(on yellow_block green_block))` indicates that the blocks must be stacked in a specific order: pink on red, red on yellow, and yellow on green. All three components must be formatted in accordance with the PDDL language specification, as demonstrated in Code Block 3.

Code Block 3: Example code block of Output (O, I, G)

```
{
   (define (problem blocksworld1)
      (:domain blocksworld)
      (:objects
         green_block - block
         robot - robot
      )
      (:init
         (ontable green_block)
         (handempty robot)
      )
      (:goal
         (and
            (on pink_block red_block)
            (on red_block yellow_block)
            (on yellow_block green_block)
         )
      )
   )
}
```

C. Process

The ICL framework introduced in section II.*B. LLMs* is applied here, with the process divided into two subprocesses: **training** and **inferencing**.

The training phase does not involve updating model parameters but rather refers to the model identifying patterns from the provided examples, learning how inputs correspond to outputs based on the given demonstrations. These examples, referred to as "In-Context Examples" within the ICL framework, are stored in Context D_E.

During the inference phase, the model applies the learned pattern to a new input, referred to as "new input" in the ICL framework and represented by Input (L, S). It then generates a structured output, represented by Output (O, I, G), by following the learned transformation.

1) Training: In order to be able to output O, I, and G, three abilities need to be acquired: the ability to **estimate the objects**, the ability to **estimate the initial state**, and the ability to **estimate the goal**.

It is important to note that all items labeled with $train, i$ correspond to subsets of the training dataset D_E. This means that the process described is applicable to any subset of the training data.

In [3], Keisuke *el al.* used the model Grounding-DINO [30], which accomplishes the recognition of objects in the image. However, since this work primarily focuses on LLM-based interaction, object identification is performed using predefined rules that operate solely on textual descriptions. Figure 5 illustrates the process of obtaining the ability to estimate objects by defining a rule. The rule defines how to convert types extracted from D_D (e.g., `block`), and object labels extracted from descriptions similar to $S_{train,i}$ (e.g., `green_block`), into a form that conforms to the PDDL standard, which prepares data for subsequent LLM training.

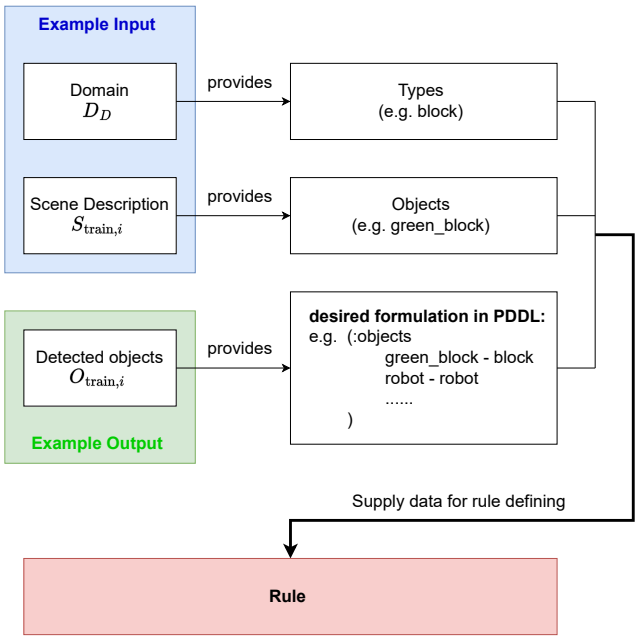

Figure 5: The process of acquiring the ability to estimate objects with the help of defining rule; This rule should be able to extract relevant information from the description of the domain D_D and the description of the scene similar to $S_{train,i}$ from the input, and generate an object list of objects written in PDDL syntax.

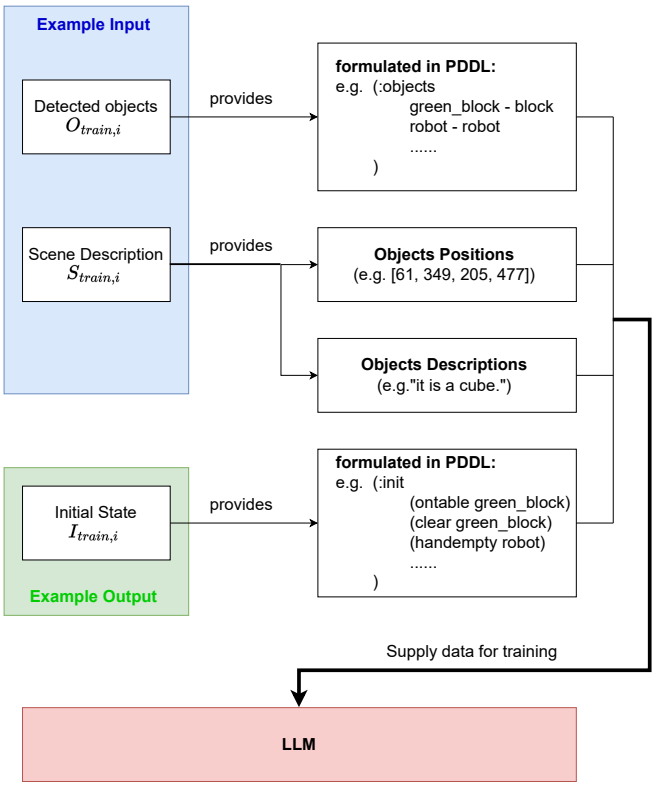

Figure 6: Training process for the ability to estimate initial states; During training, the LLM tries to learn how to create a mapping from the inputs, which are similar to Objects $O_{train,i}$ and Scene Description $S_{train,i}$, to the outputs, which are similar to Initial State $I_{\text{train},i}$.

Fig. 6 illustrates the process of training the ability to predict the initial state. A dataset consisting of example input $(O_{\text{train},i}, S_{\text{train},i})$ and corresponding example output $(I_{\text{train},i})$ is used for a single training session. Here, $O_{\text{train},i}$ provides a list of objects present in the scene, $S_{\text{train},i}$ enriches the objects with additional details, such as location and descriptive attributes. This information enables the LLM to better understand the current scene through the description. $I_{\text{train},i}$ defines the correct initial state. Through repeated exposure to such input-output mappings, the LLM learns to generate well-structured outputs for similar scenarios.

Figure 7 demonstrates the training process for the ability of goal estimation. In each training session, a dataset containing example input $(O_{\text{train},i}, I_{\text{train},i}, L_{\text{train},i})$, and corresponding example output $(G_{\text{train},i})$ is utilized. Here, $L_{\text{train},i}$ represents the user's command, and $G_{\text{train},i}$ defines the target state that the scene should reach after executing the given command. Through this training phase, the LLM learns to accurately generate goal descriptions by understanding how object states need to be updated in response to similar instructions.

2) Inferencing: For inferencing, the input must come from data that was not used during training but still shares similarities with the training examples. For instance, the user instruction might change from `Create a stack of block : pink over red over yellow over green` in the training phase to `Create two stacks of blocks: blue over pink over red, yellow over orange over green`, or the block position might be adjusted from `[61, 349, 205, 477]` in the training phase to `[202, 169, 332, 229]`. Based on this input, O, I, G in the output are generated sequentially.

Since the prediction of objects does not rely on the LLM's capabilities, the list of objects, formatted in PDDL using predefined rules from the training phase, is used as O.

A framework that integrates a model capable of estimating the initial state is referred to as **Initial state estimator**. Figure 8 illustrates its inferencing process. The initial state, generated by the LLM and formatted in PDDL, is denoted as I.

Similarly, a framework incorporating a model for goal estimation is called **Goal estimator**. Figure 9 depicts its inferencing process. The goal specification, produced by the LLM and formatted in PDDL, is represented as G.

V. Implementation

The implementation is developed on the Python programming language, with reference to the code provided by Keisuke *el al.* [3]. To enhance understanding of the overall process, the pseudocode presented in Algorithm 1 provides a structured representation of the method.

VI. Evaluation

In this section, the datasets provided by Keisuke *el al.* [3], named Problem Description Generation (ProDG) dataset,

Algorithm 1 Generating PDDL Problem Definition using In Context Learning

Require: Domain Definition D_D, Training data $D_E = \{(S_{train,i}, L_{train,i}, O_{train,i}, I_{train,i}, G_{train,i})\}$, Input (S_j, L_j)
Ensure: Generated Problem Definition (O_j, I_j, G_j)

Define rule R for formulating objects in PDDL using $S_{train,i}$ and $O_{train,i}$

for each task T_j **do**
    Use R to formulate objects in PDDL: O_j

    Initialize prompt P_j with Task Description:
        "An initial state formulated in PDDL should be generated based on objects formulated in PDDL with their positions and descriptions."
    for each training task T_i **do**
        Concatenate the prompt P_j with In-Context Example using $O_{train,i}$, $S_{train,i}$, $I_{train,i}$
    end for
    Concatenate the prompt P_j with new input using O_j, S_j
    Infer LLM to generate Initial State I_j

    Initialize prompt P_j with Task Description:
        "A goal specification formulated in PDDL should be generated based on objects formulated in PDDL, initial state, and linguistic instructions."
    for each training task T_i **do**
        Concatenate the prompt P_j with In-Context Example using $O_{train,i}$, $I_{train,i}$, $L_{train,i}$, $G_{train,i}$
    end for
    Concatenate the prompt P_j with new input using O_j, I_j, L_j
    Infer LLM to generate Goal Specification G_j

    Collect O_j, I_j, G_j to construct the output (O_j, I_j, G_j).
end for

was used for training, and a dataset different from ProDG was prepared and used for evaluation. Multiple LLMs are embedded to investigate the feasibility and performance of the proposed method.

A. Preparation of data for evaluation

In machine learning, a common practice is to allocate data for training and testing in an 8:2 ratio. However, applying this ratio here would mean that, given the 10 datasets in ProDG for training, only 2 datasets would be available for testing, which is statistically insignificant. Therefore, this approach was discarded in favor of exploring the capabilities of LLMs with limited training data.

To conduct the evaluation, 10 additional datasets were created by mimicking the ProDG data. During this process, the following aspects were carefully considered:

- The number of blocks in the initial state
- The maximum number of stack layers in the initial state
- The number of stacks in the goal state

To ensure that tasks vary in difficulty while remaining comparable, the number of blocks in the initial state was fixed at 5. The maximum number of stack layers in the initial state was chosen from {1, 2, 3, 4, 5}, and the number of stacks in the goal state was selected from {1, 2}.

Figure 10 visualizes three of these tasks. In particular, 10a and 10b share the same initial state but have different goal states, while 10a and 10c share the same goal state but have different initial states. It is important to note that these images are provided only for better understanding and the actual input to the LLM remains similar to the Scene Description format shown in Code Block 2.

B. Evaluation metrics

The evaluation metrics are categorized based on the type of LLM used and further divided into efficiency and accuracy in generating the problem description file (O, I, G).

1) Type of LLM: The evaluation involved two offline models and one online model, as listed in Table I.

Offline / Online	Model	Parameters	Size
Offline	llama3.2	3 B	2.0 GB
	llama3.2-vision	11 B	7.9 GB
Online	claude3.5-sonnet (20240620)	N/A	N/A

Table I: Overview of LLM Types [31] [32]

The two offline models, llama3.2 and llama3.2-vision, differ in terms of parameters and model size. The evaluation was conducted on a system equipped with an NVIDIA GeForce RTX 3090 graphics card with 24GB of RAM [33], and llama3.2-vision was selected as the largest model that could run smoothly given these hardware constraints [31].

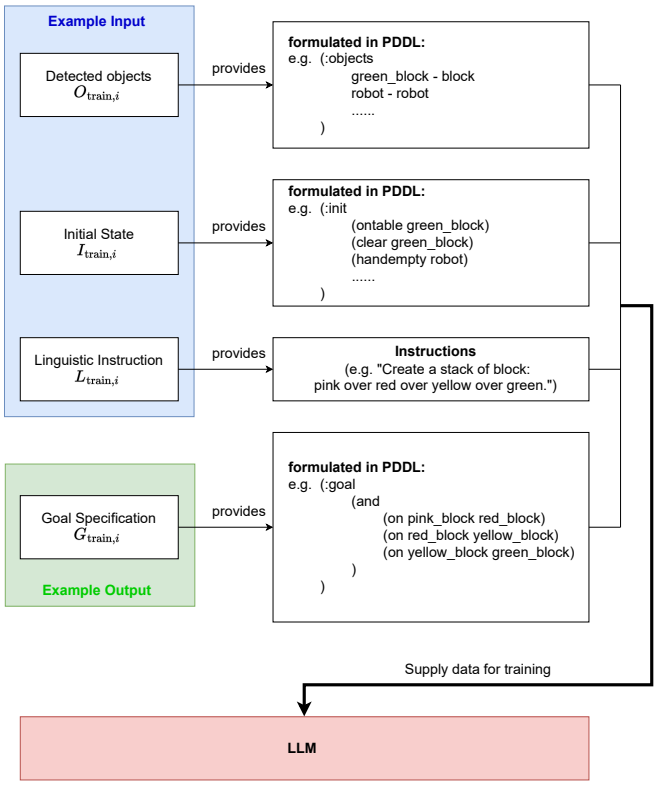

Figure 7: Training process for the ability to estimate goal; During training, the LLM tries to learn how to create a mapping from the inputs, which are similar to Objects $O_{\text{train},i}$, Initial State $I_{\text{train},i}$, and Instruction $L_{\text{train},i}$, to the outputs, which are similar to Goal Specification $G_{\text{train},i}$.

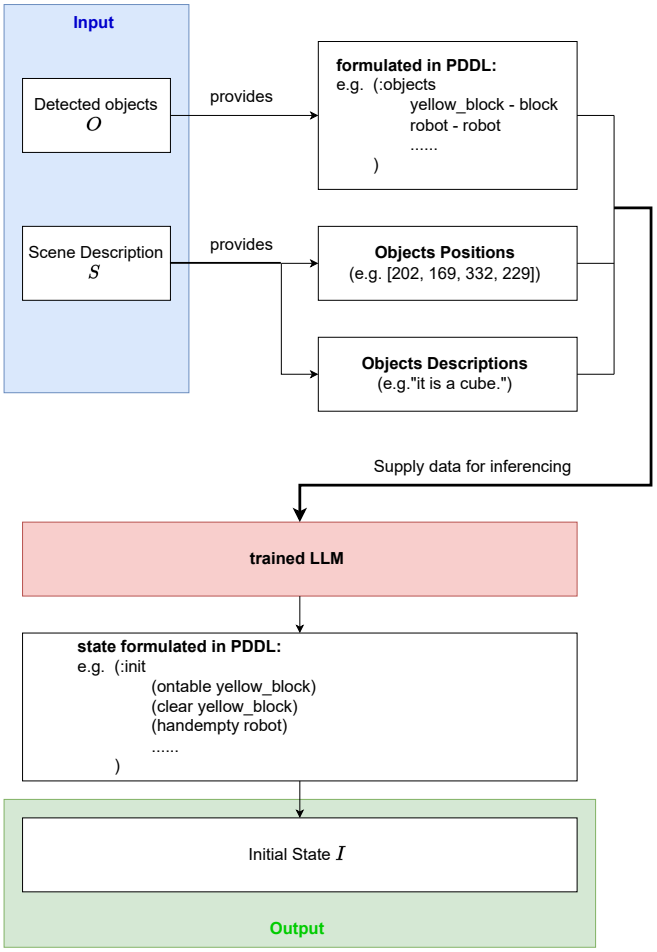

Figure 8: Inferencing process of the Initial state estimator; The trained LLM extracts the relevant information from the Scene Description S and Objects O at the input, and generates Initial State I written in PDDL syntax.

For the online model, claude3.5-sonnet was chosen based on performance comparisons [34], where it was reported to outperform both GPT-4o and Llama-400B.

2) Efficiency: The average time required to generate the problem description for each task was measured, providing insight into the efficiency of each model.

3) Accuracy: Ideally, the generated problem descriptions should be fully compliant with the PDDL syntax and correctly recognized by the planner to generate a valid plan. The assessment of accuracy can be categorized into $\boldsymbol{R}_{\textbf{syntax}}$ and $\boldsymbol{R}_{\textbf{plan}}$.

- $\boldsymbol{R}_{\textbf{syntax}}$: The percentage of problem descriptions that are syntactically correct.
- $\boldsymbol{R}_{\textbf{plan}}$: The percentage of problem descriptions that are both syntactically correct and successfully enable the planner to generate a plan.

To verify whether a valid plan could be produced, the evaluation utilized the PDDL planner provided by Magnaguagno *et al.* [35], which employs Breadth-First Search (BFS) as the default solver.

C. Evaluation result

For each model, three tests were conducted using the same set of tasks, and the results are presented in Table II. The time

Model	Nr. Test	**Time / task (ms)**	$\boldsymbol{R}_{\textbf{syntax}}$	$\boldsymbol{R}_{\textbf{plan}}$
llama3.2	1	4074.4	0.0	0.0
	2	3949.6	0.0	0.0
	3	3657.7	0.0	0.0
	Avg	**3893.9**	**0.0**	**0.0**
llama3.2-vision	1	50189	1.0	0.1
	2	43101	1.0	0.1
	3	43432	1.0	0.2
	Avg	**4557.4**	**1.0**	**0.13**
claude3.5-sonnet (20240620)	1	61828	1.0	0.9
	2	60307	1.0	0.9
	3	59018	0.9	0.7
	Avg	**6038.4**	**0.97**	**0.83**

Table II: Evaluation results

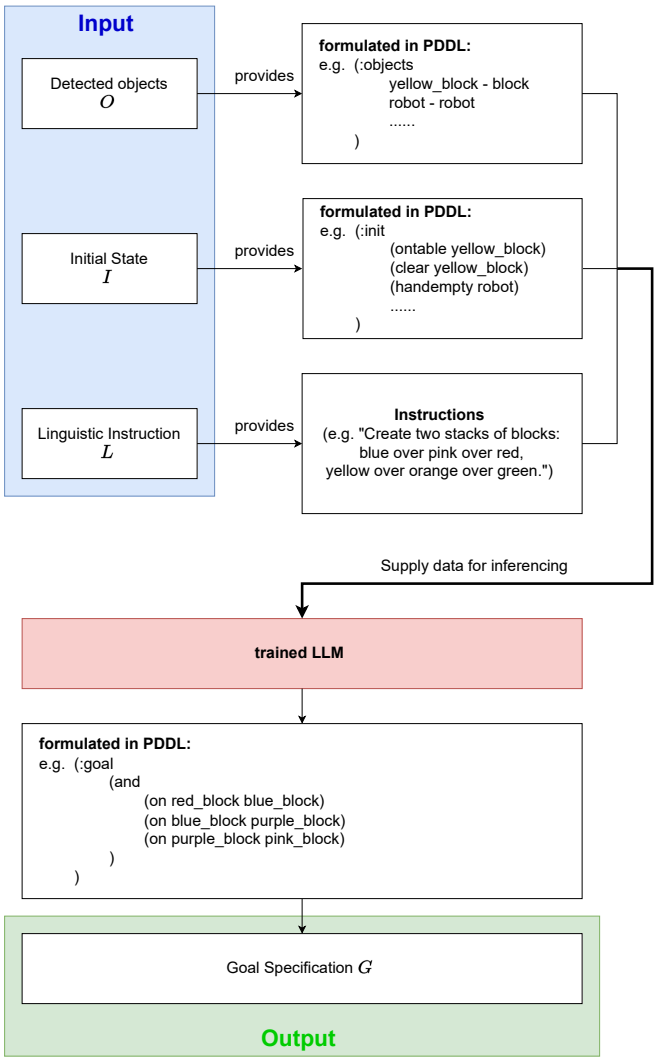

Figure 9: Inferencing process of the Goal estimator; The trained LLM extracts the relevant information from Objects O, Initial State I and Instruction L at the input and generates a Goal specification G written in PDDL syntax.

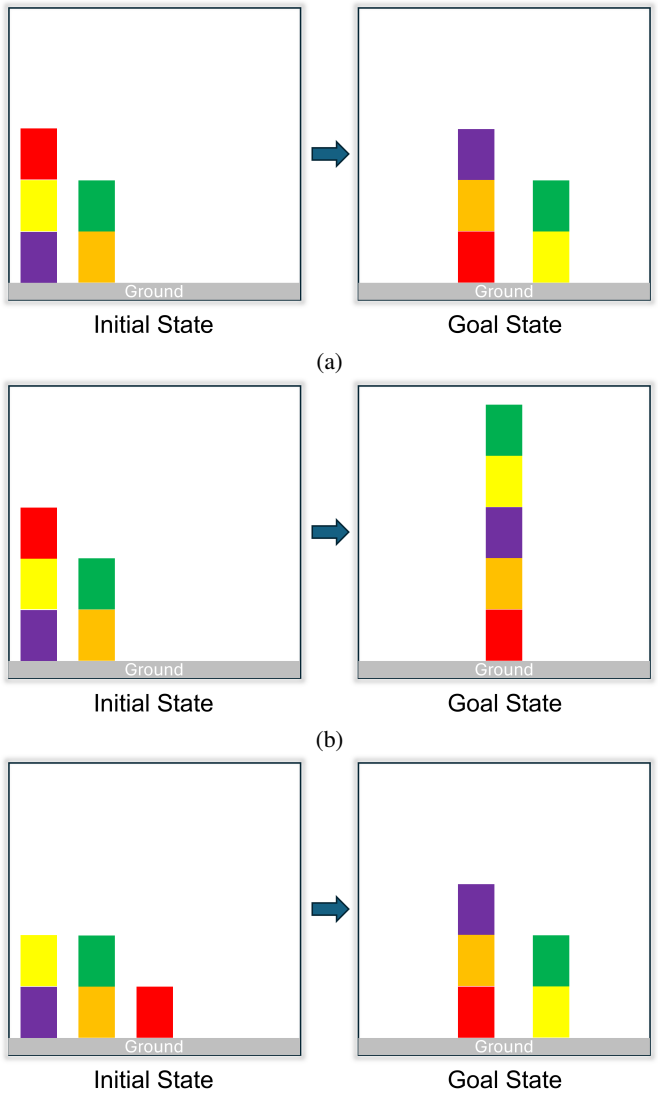

Figure 10: Example tasks for evaluation

required per task, representing efficiency, is recorded in the column "Time / task (ms)", while accuracy is reflected in the columns "R_{syntax}" and "R_{plan}". The average values across the three trials are summarized in the "Avg" row for each model.

It is important to note that these results are indicative of the specific testing environment. Factors such as the computational performance of the evaluation hardware, network conditions, and the method of request submission to the LLM may influence the outcomes.

D. Interpretation of the result

When comparing the offline models, llama3.2-vision, which has a larger number of parameters than the more lightweight llama3.2, demonstrates a significant improvement in syntactic accuracy, as reflected by an increase in R_{syntax} from 0.0 to 1.0. Additionally, it enhances the usability of the generated descriptions, with R_{plan} rising from 0.0 to 0.13. However, llama3.2-vision does not provide an advantage in efficiency, as it requires more time to generate results, with the processing time increasing from 3893.9 ms to 4557.4 ms.

When comparing offline and online models, the online model claude3.5-sonnet exhibits notably stronger performance, achieving high accuracy with R_{syntax} at 0.97 and R_{plan} at 0.83, both approaching 1.0. However, this higher accuracy comes at the expense of efficiency, as the online model requires significantly more processing time, with a runtime exceeding 6000 ms.

VII. Conclusion

In this paper, a method is proposed to generate problem descriptions in the Planning Domain Definition Language (PDDL) specification using Large Language Models (LLMs), enabling an automated planner to perform task planning based

on natural language instructions from the user and scene descriptions to facilitate robot execution. By breaking the generation process into sub-processes, the approach enhances interpretability and reliability.

The evaluation results indicate that while the offline model performed poorly, particularly in assisting the planner with generating plans, the online model demonstrated significantly better performance. This suggests that the proposed approach can be effectively implemented with a more capable language model.

However, the method has certain limitations, particularly in the usability of the generated outputs. Even for online models, neither of the two accuracy rates consistently reaches 1.0. Furthermore, network instability adds an additional layer of uncertainty. To improve the reliability and practicality of the outputs, future research could explore the following directions:

- Refining outputs through re-prompting
  Incorporating feedback mechanisms, such as leveraging error messages from planners, could help improve the accuracy of the generated problem definitions. By iteratively refining responses based on detected inconsistencies or formatting issues, the model's performance may be further optimized. [21] [3]
- Integrating human feedback
  Actively involving human oversight in the validation and correction of outputs could ensure better alignment with real-world conditions and user intentions. This approach may involve interactive refinement loops where users provide corrections or clarifications to guide the model toward more precise and usable outputs. [25]

Acknowledgement

I would like to express my sincere gratitude to my professor, Prof. Dr. rer. nat. Alexander Ferrein, for his valuable guidance and support throughout this research. I also extend my thanks to my tutors, Maximillian Kirsch and Ninad Kulkarni, for their assistance and insightful feedback during the course of this work. Their contributions have been essential to the completion of this study.

References

[1] L. H. Gilpin, D. Bau, B. Z. Yuan, A. Bajwa, M. Specter, and L. Kagal, *Explaining explanations: An overview of interpretability of machine learning*, 2019. arXiv: 1806.00069 [cs.AI]. [Online]. Available: https://arxiv.org/abs/1806.00069.

[2] P. Haslum, N. Lipovetzky, D. Magazzeni, and C. Muise, "An introduction to the planning domain definition language," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:132910614.

[3] K. Shirai, C. C. Beltran-Hernandez, M. Hamaya, *et al.*, "Vision-Language Interpreter for Robot Task Planning," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*, May 2024, pp. 2051–2058. DOI: 10.1109/ICRA57147.2024.10611112. [Online]. Available: https://ieeexplore.ieee.org/document/10611112.

[4] S. Miglani, N. Yorke-Smith, and N. Yorke-Smith, *Nl to pddl*. [Online]. Available: https://repository.tudelft.nl/record/uuid:1727bc3f-c0ca-4439-9590-914339678723 (visited on 03/13/2025).

[5] H. Xia, Z. Yang, Q. Dong, *et al.*, *Unlocking efficiency in large language model inference: A comprehensive survey of speculative decoding*, 2024. arXiv: 2401.07851 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2401.07851.

[6] S. Wang, Y. Zhao, X. Hou, and H. Wang, *Large language model supply chain: A research agenda*, 2024. arXiv: 2404.12736 [cs.SE]. [Online]. Available: https://arxiv.org/abs/2404.12736.

[7] Z. Zhou, J. Song, K. Yao, Z. Shu, and L. Ma, *Isr-llm: Iterative self-refined large language model for long-horizon sequential task planning*, 2023. arXiv: 2308.13724 [cs.RO]. [Online]. Available: https://arxiv.org/abs/2308.13724.

[8] P. Zeng, Z. Ning, J. Zhao, *et al.*, *The cap principle for llm serving: A survey of long-context large language model serving*, 2024. arXiv: 2405.11299 [cs.DB]. [Online]. Available: https://arxiv.org/abs/2405.11299.

[9] Microsoft, *How Generative AI and LLMs work - .NET*, en-us, 2024. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/ai/conceptual/how-genai-and-llms-work (visited on 03/13/2025).

[10] Amazon Web Services, *What is LLM? - Large Language Models Explained - AWS*, en-US. [Online]. Available: https://aws.amazon.com/what-is/large-language-model/ (visited on 03/13/2025).

[11] J. Shin, C. Tang, T. Mohati, M. Nayebi, S. Wang, and H. Hemmati, *Prompt engineering or fine-tuning: An empirical assessment of llms for code*, 2025. arXiv: 2310.10508 [cs.SE]. [Online]. Available: https://arxiv.org/abs/2310.10508.

[12] Z. Sheng, "The training process and methods for llms using an own knowledge base," *Journal of Artificial Intelligence Practice*, 2024. [Online]. Available: https://api.semanticscholar.org/CorpusID:272126224.

[13] Y. Liu, A. Singh, C. D. Freeman, J. D. Co-Reyes, and P. J. Liu, *Improving large language model fine-tuning for solving math problems*, 2023. arXiv: 2310.10047 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2310.10047.

[14] R. Pan, X. Liu, S. Diao, *et al.*, *Lisa: Layerwise importance sampling for memory-efficient large language model fine-tuning*, 2024. arXiv: 2403.17919 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2403.17919.

[15] J. Zou, M. Zhou, T. Li, S. Han, and D. Zhang, *Promptintern: Saving inference costs by internalizing recurrent prompt during large language model fine-tuning*, 2024. arXiv: 2407.02211 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2407.02211.

[16] M. F. Hanafi, Y. Katsis, I. Jindal, and L. Popa, "A comparative analysis between human-in-the-loop systems and large language models for pattern extraction tasks," in *DASH*, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:256461088.

[17] Q. Dong, L. Li, D. Dai, *et al.*, *A survey on in-context learning*, 2024. arXiv: 2301.00234 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2301.00234.

[18] C. Paxton, Y. Bisk, J. Thomason, A. Byravan, and D. Fox, *Prospection: Interpretable plans from language by predicting the future*, 2019. arXiv: 1903.08309 [cs.AI]. [Online]. Available: https://arxiv.org/abs/1903.08309.

[19] P. Sharma, A. Torralba, and J. Andreas, *Skill induction and planning with latent language*, 2022. arXiv: 2110.01517 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2110.01517.

[20] M. Ahn, A. Brohan, N. Brown, *et al.*, *Do as i can, not as i say: Grounding language in robotic affordances*, 2022. arXiv: 2204.01691 [cs.RO]. [Online]. Available: https://arxiv.org/abs/2204.01691.

[21] Y. Chen, J. Arkin, C. Dawson, Y. Zhang, N. Roy, and C. Fan, *Autotamp: Autoregressive task and motion planning with llms as translators and checkers*, 2024. arXiv: 2306.06531 [cs.RO]. [Online]. Available: https://arxiv.org/abs/2306.06531.

[22] Y. Ding, X. Zhang, S. Amiri, *et al.*, "Integrating action knowledge and llms for task planning and situation handling in open worlds," *Autonomous Robots*, vol. 47, no. 8, pp. 981–997, Aug. 2023, ISSN: 1573-7527. DOI: 10.1007/s10514-023-10133-5. [Online]. Available: http://dx.doi.org/10.1007/s10514-023-10133-5.

[23] Y. Ding, X. Zhang, C. Paxton, and S. Zhang, *Task and motion planning with large language models for object rearrangement*, 2023. arXiv: 2303.06247 [cs.RO]. [Online]. Available: https://arxiv.org/abs/2303.06247.

[24] W. Huang, P. Abbeel, D. Pathak, and I. Mordatch, *Language models as zero-shot planners: Extracting actionable knowledge for embodied agents*, 2022. arXiv: 2201.07207 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2201.07207.

[25] W. Huang, F. Xia, T. Xiao, *et al.*, *Inner monologue: Embodied reasoning through planning with language models*, 2022. arXiv: 2207.05608 [cs.RO]. [Online]. Available: https://arxiv.org/abs/2207.05608.

[26] K. Lin, C. Agia, T. Migimatsu, M. Pavone, and J. Bohg, "Text2motion: From natural language instructions to feasible plans," *Autonomous Robots*, vol. 47, no. 8, pp. 1345–1365, Nov. 2023, ISSN: 1573-7527. DOI: 10.1007/s10514-023-10131-7. [Online]. Available: http://dx.doi.org/10.1007/s10514-023-10131-7.

[27] I. Singh, V. Blukis, A. Mousavian, *et al.*, *Progprompt: Generating situated robot task plans using large language models*, 2022. arXiv: 2209.11302 [cs.RO]. [Online]. Available: https://arxiv.org/abs/2209.11302.

[28] K. Valmeekam, S. Sreedharan, M. Marquez, A. Olmo, and S. Kambhampati, *On the planning abilities of large language models (a critical investigation with a proposed benchmark)*, 2023. arXiv: 2302.06706 [cs.AI]. [Online]. Available: https://arxiv.org/abs/2302.06706.

[29] IBM Research, "When ai plans ahead," *The Gradient*, 2019. [Online]. Available: https://thegradient.pub/when-ai-plans-ahead/.

[30] S. Liu, Z. Zeng, T. Ren, *et al.*, *Grounding dino: Marrying dino with grounded pre-training for open-set object detection*, 2024. arXiv: 2303.05499 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2303.05499.

[31] Ollama. "Ollama." (2025), [Online]. Available: https://github.com/ollama/ollama?tab=readme-ov-file#model-library (visited on 03/13/2025).

[32] anthropic. "Claude sonnet." (2025), [Online]. Available: https://www.anthropic.com/claude/sonnet (visited on 03/13/2025).

[33] Nvidia. "Geforce rtx 3090 family." (2025), [Online]. Available: https://www.nvidia.com/en-us/geforce/graphics-cards/30-series/rtx-3090-3090ti/ (visited on 03/13/2025).

[34] anthropic. "Claude sonnet." (2025), [Online]. Available: https://www.anthropic.com/news/claude-3-5-sonnet (visited on 03/13/2025).

[35] M. Magnaguagno, F. Meneguzzi, and Anwar, *Python pddl parser: Version 1.1*, version v1.1, Dec. 2020. DOI: 10.5281/zenodo.4391071. [Online]. Available: https://doi.org/10.5281/zenodo.4391071.