ABSTRACT

Title of Dissertation Proposal: TRANSLATING NATURAL LANGUAGE TO VISUALLY GROUNDED VERIFIABLE PLANS

Angelos Mavrogiannis
Doctoral Student, Department of Computer Science

Advised by: Yiannis Aloimonos
Professor, Department of Computer Science

To be useful in household environments, robots may need to understand natural language in order to parse and execute verbal commands from novice users. This is a challenging problem that requires mapping linguistic constituents to physical entities and at the same time orchestrating an action plan that utilizes these entities to complete a task. Planning problems that previously relied on querying manually crafted knowledge bases can now leverage Large Language Models (LLMs) as a source of commonsense reasoning to map high-level instructions to action plans. However, the produced plans often suffer from model hallucinations, ignore action preconditions, or omit essential intermediate actions under the assumption that users can infer them from context and prior experience.

In this proposal, we present our work on translating natural language instructions to visually

grounded verifiable plans. First, we motivate the use of classical concepts such as Linear Temporal Logic (LTL) to verify LLM-generated action plans. By expressing these plans in a formal language notation that adheres to a set of rules and specifications, we can generate discrete robot controllers with provable performance guarantees. Second, we focus on grounding linguistic instructions to visual sensory information and we find that Vision Language Models (VLMs) often struggle with identifying non-visual attributes. Our key insight is that non-visual attribute detection can be effectively achieved by active perception guided by visual reasoning. To this end, we present a perception-action API that consists of perceptual and motoric functions. When prompted with this API and a natural language query, an LLM generates a program to actively identify attributes given an input image. Third, we present ongoing work using the Planning Domain Definition Language (PDDL) as an action representation. By binding perceptual functions to action preconditions and effects explicitly modeled in the PDDL domain, we visually validate successful action execution at runtime, producing visually grounded verifiable action plans.

TRANSLATING NATURAL LANGUAGE TO VISUALLY GROUNDED VERIFIABLE PLANS

by

Angelos Mavrogiannis

Dissertation Proposal submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2024

Examining Committee:
Dr. Yiannis Aloimonos, Chair
Dr. Jordan Boyd-Graber, Department Representative
Dr. Cornelia Fermüller, Member

Table of Contents

Chapter 1: Introduction

1.1 Translating Natural Language Instructions to Verifiable Plans

To be useful in household environments, robots may need to understand and execute instructions from novice users. Natural language is possibly the easiest way for users to provide instructions to robots but it is often too vague. This motivates the need for mapping natural language to actionable, robot-executable commands. This is a challenging problem, especially for complex activities that include temporally correlated subtasks, such as following instructions in a manual, performing a delicate assembly task, or executing a multi-step cooking recipe.

Cooking is one of the most common household activities and poses a unique set of challenges to robots [1]. It usually requires following a recipe, written assuming that the reader has some background experience in cooking and commonsense reasoning to understand and complete the instruction steps. Recipes often feature ambiguous language [2], such as omitting arguments that are easily inferred from context (the known "Zero Anaphora" problem [3]; see Fig. 3.3b where the direct object of the verb "cook" is missing), or, more crucially, underspecified tasks under the assumption that the reader possesses the necessary knowledge to fill in the missing steps. For example, recipes with eggs do not explicitly state the prerequisite steps of cracking them and extracting their contents. Additionally, although inherently sequential, recipes often include additional explicit sequencing language (e.g. until, before, once) that clearly defines the temporal action boundaries.

Motivated by these observations, our key insight is that combining a source of cooking domain

knowledge with a formalism that captures the temporal richness of cooking recipes could enable the extraction of unambiguous, robot-executable plans. Building on this insight, we present Cook2LTL [4], a system that receives a cooking recipe in natural language form, reduces high-level cooking actions to robot-executable primitive actions through the use of LLMs, and produces unambiguous task specifications written in the form of Linear Temporal Logic (LTL) [5] formulae (See Fig. 3.1). These plans are then suitable for use in downstream robotic tasks. We build and evaluate our method based on a subset of recipes from the Recipe1M+ corpus [6]. We run Cook2LTL on these recipes and show that by caching the action reduction policy, we incrementally build a queryable action library and limit proprietary LLM API calls with significant benefits in cost (-42%) and computation time (-59%) compared to a baseline that queries the LLM for every unseen action at runtime. We demonstrate the transferability of Cook2LTL to a robotic platform through experiments in a simulated kitchen in AI2-THOR [7].

A natural extension when deploying such a system to a real robotic platform requires validating action preconditions and postconditions with perceptual evidence during plan execution. This enables detecting plan disruptions and dynamic changes in the environment by perceptually validating whether an action was successfully executed. Instead of executing an action, failing, and then re-prompting an LLM to repair the plan, our approach can predict failures resulting from unsatisfied action postconditions, halt execution, and re-plan without succumbing to failure. The Planning Domain Definition Language (PDDL) offers a well-structured task representation that is convenient for modeling cooking activities by explicitly encoding action preconditions and effects in the PDDL domain. This is particularly useful for tracking the state change of an ingredient or food item. State changes can be phase transitions, for example butter melting on a pan, or changes in quantity in terms of the visible ingredients in the kitchen, such as the tomato slices resulting from slicing a tomato. Using this representation also allows us to leverage already existing PDDL infrastructure, such as VAL [8], a tool for syntactically validating generated PDDL files.

To integrate these ideas in a robotic platform, we are developing NL2PDDL2Prog, a system that

receives a cooking instruction in the form of natural language, produces a PDDL [9] goal specification, and generates a plan that is parsed into an executable python program. By binding perceptual functions to action preconditions and postconditions in the program, we visually ground successful action execution at runtime. If any of these conditions is not satisfied, program execution halts and a precondition or postcondition error is raised, displaying the specific source of failure. Our approach is similar to the work of Shirai et al. [10] but they only leverage visual input at the initial stage to acquire a list of available objects or in the case of plan failure. By incorporating perceptual feedback to ground action preconditions and effects between every action, our proposed framework is applicable to a wider variety of kitchen tasks of higher complexity.

1.2 Grounding Linguistic Instructions to Visual Sensory Information

Connecting natural language instructions to the physical world often requires robots to detect object attributes in order to discriminate between candidate objects. This is a challenging problem as instructions might be linguistically ambiguous, for example "Can you please get me the second mug from the right on that shelf?". Identifying attributes can also be required implicitly to determine the state or affordance [11] of an object in order to verify the feasibility of an action. These attributes might not be directly perceivable through vision sensors, for example "Is this lightweight enough to pick up?".

To ground linguistic instructions in embodied settings, we focus on actively identifying object attributes in a programmatic fashion, combining perceptual and motoric functions. Attributes usually appear in the form of descriptive adjectives, some of which might not be adequately represented in the training sets of data-driven perceptual models. Furthermore, while attributes might not necessarily characterize an object in an absolute scale, they are often applicable based on context [12]. We argue that attribute detection is highly contextual - an analogy of J. R. Firth's famous quotation "You shall know a word by the company it keeps" [13] applies to attributes. More specifically, characterizing an object as big, tall, or heavy can sometimes depend on the other objects and their respective attribute values in the

current environmental context. Ambiguity can also arise due to occlusion or partial observability [14]. These problems might not occur when studying attribute detection in static images, but can be common in a household environment where a robot is tasked with executing user instructions. In these cases, erroneous attribute detection can be detrimental, producing and executing an action plan involving an incorrect object, or misinterpreting the affordance of an object and failing to even execute the action.

Existing attribute detectors [14–20] are mainly obtained by either supervised training [21] or contrastive pre-training [22]. While attribute detection is an active area of research, it is often studied separately from embodied reasoning. To bridge this gap, we model attribute detection as visual reasoning with programs. This provides us with a powerful representation for reasoning in the presence of embodied agents and allows us to utilize the space of plans and movements via robot actions as programs [23].

Summarizing these ideas, our main observation is that modern real-world robotic systems relying on visually-driven attribute detection using VLMs in isolation can be myopic in language grounding. Our key insight is that combining different VLMs as visual reasoning functions with a robot control API can benefit from the code synthesis and commonsense reasoning capabilities of LLMs to actively reason about attribute detection in the form of computer programs. We prompt an LLM with an attribute detection API on a dataset that we curate, consisting of embodiment-crucial **location**-, **size**-, and **weight**-related attributes and construct a perception-action API for active attribute detection [24]. Our key contributions can be summarized as follows:

- We highlight some of the drawbacks of using VLMs for attribute detection in isolation and the complementary reasoning capabilities that emerge by reasoning in the form of LLM-generated visual programs.
- We construct a perception-action API by integrating visual reasoning with robot control functions and demonstrate its benefits by invoking active perception behaviors towards solving attribute detection queries.
- We release an end-to-end framework that integrates this perception-action API on a real robotic

platform using visual servoing-based control.

Chapter 2: Related Work

We begin our literature review by surveying robotic cooking (Sec. 2.1), which is our key domain for studying the translation of natural language to verifiable robot-executable plans. We continue by reviewing recent work in LLM planning (Sec. 2.2) and ways to validate LLM-generated plans using classical concepts like LTL or PDDL (Sec. 2.3). Finally, we look into attribute detection (Sec. 2.4) and visual reasoning with programs (Sec. 2.5) as they contribute towards our effort in invoking active perception behaviors through LLM-generated programs.

2.1 Robotic Cooking

Cooking has been an important means of studying action understanding [25–28]. The EU project POETICON [25] viewed cognitive systems as a set of languages {natural, visual, motoric} and integrated these languages towards understanding cooking actions. Along these lines, Yang et al. [26] processed YouTube videos using Convolutional Neural Networks (CNNs) and a grammatical approach [27] to produce parse trees that could be used for generating cooking actions. Several works have focused on modeling and learning the lower-level mechanics of manipulation in cooking actions with the end-goal of building intelligent aspiring robot chefs [29–31]. Bollini et al. [1], Beetz et al. [28], and Liu et al. [29] developed autonomous end-to-end robotic cooking systems tailored to the specific tasks of baking, stir-frying, and making pancakes, respectively. RoboCook [32] and Mobile ALOHA [33] recently introduced versatile learning-based systems that intricately handle cooking-specific complex soft body manipulation tasks and mobile manipulation tasks in the kitchen, respectively. Most of these works are constrained to

preparing a unique dish [1, 28, 29], require significant adaptations and additional training [32] to support novel recipes, or cannot handle verbal instructions as input [31, 33]. Translating recipes to actionable robot plans is a challenging problem due to the linguistic richness and semantic ambiguity of recipes [2]. Recent approaches have built LLM-powered human-robot collaborative cooking interfaces equipped with a human intention prediction module [34, 35]. We leverage LLMs to decompose abstract free-form recipe text into executable robot plans [4]. Our AI2-THOR [7] simulation in Sec. 3.3.2 demonstrates the transferability of our work to a real robot while allowing the system to adapt to new recipes. In our ongoing work, proposed in Chapter 5, we incorporate perceptual evidence to visually ground action preconditions and postconditions at runtime.

2.2 LLM Planning

The field of robotics has been broadly and profoundly impacted by LLMs. Planning problems that previously relied on querying manually-crafted knowledge bases [36] can now leverage LLMs as a source of commonsense reasoning to map natural language commands to sequences of lower-level actions that can be easier to parse by robot controllers [37–39]. This has been achieved predominantly by a few-shot prompting scheme where the LLM receives a set of examples of sample tasks and action plans as input, and generates an action plan for an unseen task at inference time. These LLM-based works have shown great performance in grounding high-level actions to a well-defined set of actions for task planning but come with certain limitations. For instance, the framework of Ichter et al. [37] cannot handle open-vocabulary or combinatorial tasks, the one by Huang et al. [39] might produce action plans including items that are not present in the current environment, and the model of Huang et al. [38] does not guarantee that the returned actions are admissible in the current context.

To constrain the output to a given environment, subsequent works leveraged the code synthesis ability of LLMs to project natural language to an intermediate representation of commands-function calls in the form of a computer program, bound by the more rigorous syntactic rules of programming

languages [23, 40, 41]. More specifically, these works include action precondition checking through conditional and assertion statements [23], reasoning about task execution using control flow tools [41] and recursively defining undefined functions [40], or simply invoking VLMs in task execution functions [42]. In the context of cooking, Wang et al. [43] have used LLMs to break down high-level cooking actions into actionable plans. However, their approach requires access to demonstrations of the intermediate steps of the cooking task at hand. In our work [4], we adapt the methodology proposed by Singh et al. [23], where the task planning problem is formulated as a pythonic few-shot prompting scheme. The prompt consists of a pythonic import of a set of primitive actions, a definition of a list of available objects, and a few example task plans in the form of pythonic functions. Their experiments showed that prompting an LLM for task planning in a programmatic fashion outperforms verbose descriptive prompts by restricting the output plan to the constrained set of primitive actions and objects available in the current environment.

Furthermore, many of these approaches focus predominantly on action planning and ground linguistic constituents to physical entities by using a single call to an Open-Vocabulary object Detection (OVD) model [22, 44–46]. Due to the limitations of these models [16], they might not be able to handle attribute detection in challenging scenarios. Similar to [47], we combine the expressiveness of an intermediate programmatic representation and the complementary reasoning capabilites of LLMs and VLMs [48] to reason about attribute detection under an LLM-prompting scheme [24].

2.3 Validating LLM Output with Classical Concepts

While enforcing more rigor to the structure of the output by prompting the model to output a computer program, there are no guarantees on the validity of the program [23], which might require ad-hoc code reviewing to ensure compliance with the specifications of the environment. In long-horizon planning, such methods still suffer from model hallucinations, ignoring action pre-conditions, or omitting essential intermediate actions under the assumption that users can infer them from context and prior experience [49]. To bridge this gap, a recent line of research has introduced formal logic

and standardized formulations from classical AI planning as additional mechanisms to ensure plan validity before deployment to the robot [4, 50–62]. One of these mechanisms is Linear Temporal Logic (LTL) [5], which was initially used in formal verification for computer programs. Since then, it has been extensively used in robotics [63–65] as a formalism that enables the extraction of guarantees on robot performance given a robot model, a high-level description of its actions, and a class of admissible environments.

There has been considerable work on translating natural language instructions to task specifications in the form of LTL [51, 66–70] and its variants [53, 71] with the end-goal of using the output specifications to generate an automaton that validates LLM-generated plans [55, 56]. Most approaches try to address the main bottleneck which is the high cost of obtaining annotations of natural language with their equivalent LTL logical forms. Gopalan et al. [66] orchestrate a data collection and augmentation pipeline to build a synthetic domain and translate natural language to LTL formulae using Seq2Seq models [72]. Alternatively, Patel et al. [67], Wang et al. [68] learn from trajectories paired with natural language to reduce the need for human annotation, however a lot of trajectories are required to implicitly supervise the translator. Berg et al. [69], Liu et al. [70] ground referring expressions to a known set of atomic propositions and translate to LTL formulae using Seq2Seq models [73] and LLMs [74], respectively. Similarly, Pan et al. [51], Chen et al. [54] use the paraphrasing abilities of LLMs to generate synthetic datasets tackling the scarcity of labeled LTL data.

Our work [4] is more similar to the work of Chen et al. [54] and Hsiung et al. [75], abstracting natural language to an intermediate representation layer before grounding to the final atomic propositions. An important limitation of these methods is that they are based on thoroughly curated datasets or well-structured synthetic data generation pipelines. On the contrary, we deal with unstructured free-form recipe text scraped from the internet. Moreover, most of these works in embodied settings have mainly been applied to navigation and simple pick-and-place tasks or combinations of these. Our web-scraped cooking recipe corpus offers a richer and more diverse action space.

As a more human-readable specification language that has been widely used to enforce a

standardized structure in long-horizon planning problems [76], PDDL [9] has been an alternative mechanism to validate LLM-generated plans. Recent work has been translating natural language to PDDL specifications [57–62] via LLM few-shot prompting. However, most of these approaches often rely on feedback from interactions with simulators to evaluate predicates. In contrast, in our current work we attach perceptual callback functions to action preconditions and postconditions and visually ground the corresponding predicates, inspired by the work of Migimatsu and Bohg [77].

2.4 Attribute Detection

Attribute detection has been a fundamental problem in the computer vision community with early work [21, 78–80] on learning visual attribute classifiers to describe unseen objects. There has been work on identifying relative attributes [12, 81] but these approaches require prior training and can be limited to a certain domain ([81] demonstrate relative attributes of shoes and facial characteristics), while we show a general method [24] that works in a zero-shot fashion. Recent approaches have applied Open-Vocabulary object Detection (OVD) task [16, 17] to attribute detection. The goal in OVD is to detect unseen classes of objects defined at inference time in the form of textual queries. Bravo et al. [16] showed that the performance of various VLMs in zero-shot attribute detection is still low compared to OVD. However, most of these approaches focus on visually-perceivable attributes in disembodied settings. On the other hand, we focus on embodiment-crucial attributes such as the weight of an object, leveraging the physical reasoning capabilities of VLMs [82]. Our simulations in AI2-THOR and our robot demonstration (See Fig. 4.1) shows that our end-to-end framework can invoke active perception behaviors to reason about object attributes, inspired by prior work [14, 19].

2.5 Visual Reasoning with Programs

Generating and executing programs for vision applications originated from Neural Module Networks (NMNs) [83–85], on the basis of the idea that complex vision tasks are fundamentally compositional. Motivated by this idea, NMNs decompose a task into trainable modules that learn specific perceptual functions. However, these models produce domain-limited programs, rely on hand-tuned parsers [83] or are difficult to optimize [84, 85]. To overcome these shortcomings, a recent line of work has proposed a formulation of generating visual programs to deal with image-based natural language queries through in-context learning with an LLM. The programs consist of pseudocode instructions [86] or executable python code [87–89] and intermediate variables that map to computer vision models, image processing subroutines, or LLMs. These intermediate variables are consumable downstream and illustrate a step-by-step reasoning process towards the task at hand, which is primarily related to language grounding or Visual Question Answering (VQA). Our work [24] invokes active perception robot behaviors guided by visual programming towards attribute detection.

Chapter 3: Cook2LTL: Translating Cooking Recipes to LTL Formulae using Large Language Models

3.1 Preliminaries

This section provides a short background on LTL and LLMs, which are the tools we are using in our pipeline.

3.1.1 Linear Temporal Logic

LTL is a temporal logic that was developed for formal verification of computer programs through model checking [5]. It is suitable for expressing task specifications and verifying system performance in safety-critical applications. These task specifications are expressed through the use of this grammar:

$$\phi ::= p \mid \neg p \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \mathcal{G}\,\phi \mid \mathcal{F}\,\phi \mid \phi_1\,\mathcal{U}\,\phi_2 \tag{3.1}$$

where ϕ is a task specification, ϕ_1 and ϕ_2 are LTL formulae, and $p \in$ is an atomic proposition drawn from a set $\mathcal{P}$ of atomic propositions (APs). $\neg, \wedge, \vee$ are the known symbols from standard propositional logic denoting negation, conjunction, and disjunction, respectively. As an extension, LTL supports additional temporal operators. More specifically, $\mathcal{G}\,\phi$ denotes that ϕ holds globally, $\mathcal{F}\,\phi$ denotes that ϕ must eventually hold, and $\phi_1\,\mathcal{U}\,\phi_2$ indicates that ϕ_1 must hold for all time steps until ϕ_2 becomes true for the first time. In this work, we utilize LTL as a formal language to express temporally-extended

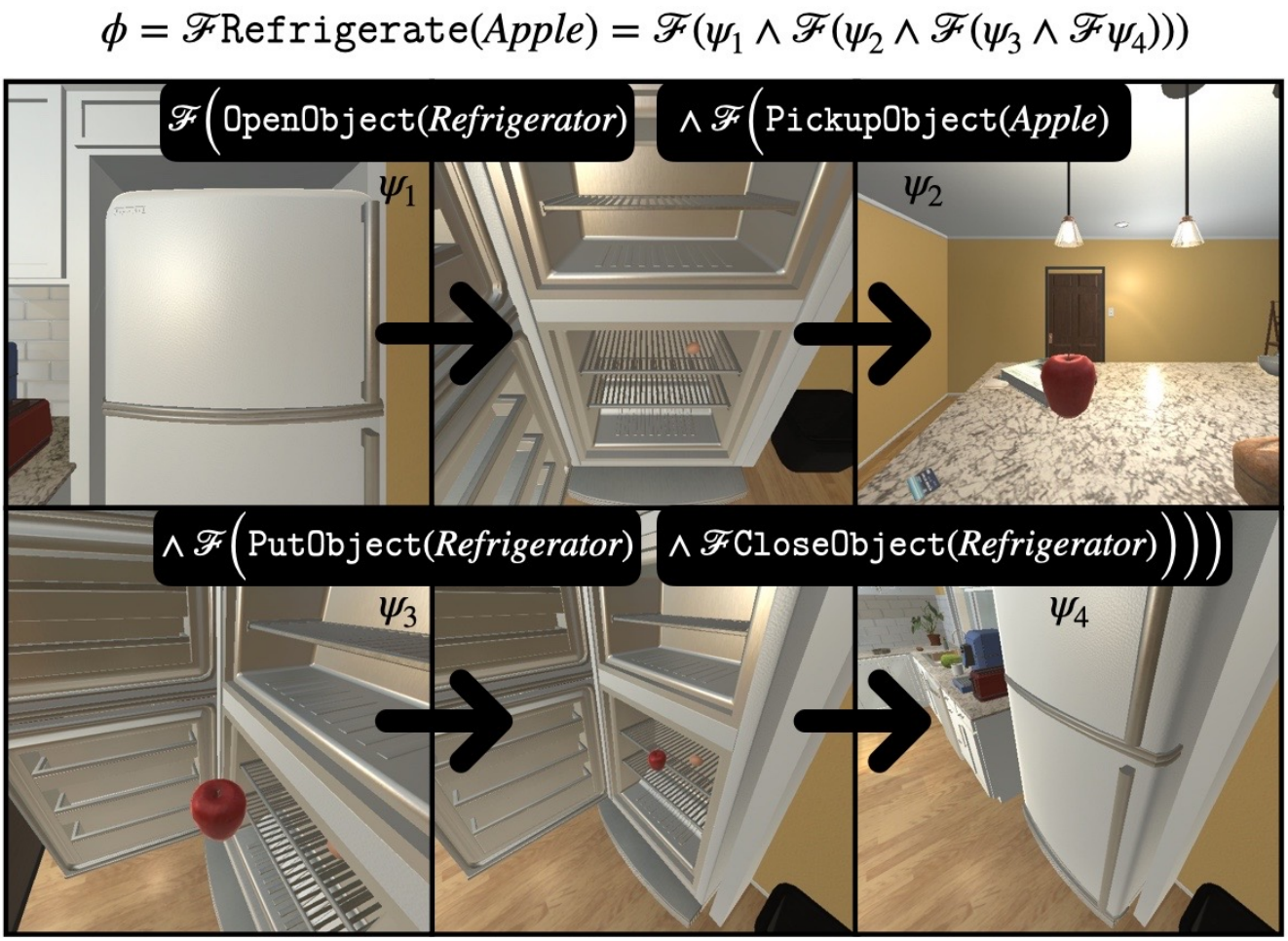


Figure 3.1: **Cook2LTL in AI2-THOR** [7]: The robot is given the instruction *Refrigerate the apple*. Cook2LTL produces an initial LTL formula ϕ (*top left*); then it queries an LLM to retrieve the low-level admissible primitives for executing the action; finally it generates a formula consisting of 4 atomic propositions $(\psi_1, \psi_2, \psi_3, \psi_4)$ that provide the required task specification and yield these consecutive scenes.

cooking tasks.

3.1.2 Large Language Models

Given a piece of text $W = \{w_1, w_2, \ldots, w_n\}$ consisting of n words $w_i, i = 1, \ldots, n$, a language model estimates the probability $p(W)$. This is done in an auto-regressive manner, leveraging the chain rule to factorize the probability [90]:

$$p(W) = p(w_1, w_2, \ldots, w_n) = \prod_{i=1}^{n} p(w_i | w_1, \ldots, w_{i-1}) \tag{3.2}$$

Generating text can then be achieved recursively. Given a set of preceding words $\{w_1, w_2, \ldots, w_{i-1}\}$, the model estimates the probability distribution for the next word $p(w_i | w_1, \ldots, w_{i-1})$. LLMs, such as BERT [91] and GPT-3 [74] are pre-trained on large-scale internet corpora and have dominated across a series of downstream natural language processing (NLP) tasks [92]. In this work, we leverage the domain knowledge encoded into such models in order to reduce high-level tasks to actions on a lower level of abstraction.

3.2 Translating Cooking Recipes to LTL Formulae

3.2.1 Problem Statement

Consider a robot in a kitchen, equipped with a limited set of primitive actions $\mathcal{A}$. We assume that a primitive action in a cooking environment can be described by a set of salient categories $\mathcal{C} = \{$`Verb, What?, Where?, How?, Time, Temperature`$\}$. We define an action description `a` as a function consisting of a main `Verb` as the function name, with a set of one or more of the other

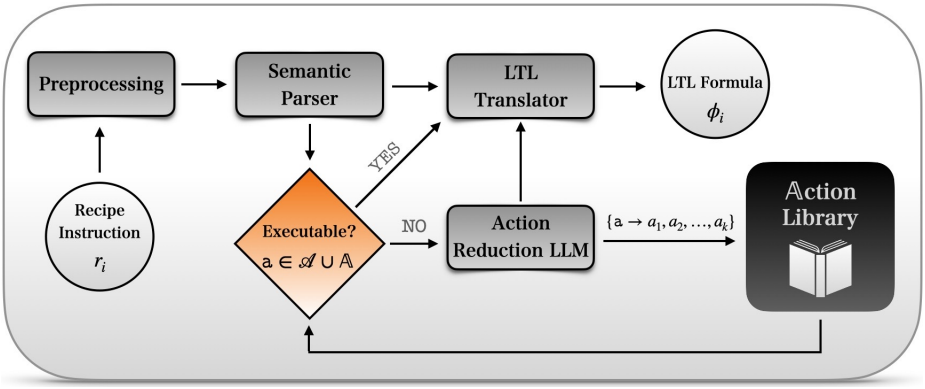


Figure 3.2: **Cook2LTL System:** The input instruction r_i is first preprocessed and then passed to the semantic parser, which extracts meaningful chunks corresponding to the categories $\mathcal{C}$ and constructs a function representation $\mathtt{a}$ for each detected action. If $\mathtt{a}$ is part of the action library $\mathbb{A}$, then the LTL translator infers the final LTL formula ϕ. Otherwise, the action is reduced to a sequence of lower-level admissible actions $\{a_1, a_2, \ldots a_k\}$ from $\mathcal{A}$, and the reduction policy is cached to $\mathbb{A}$ for future use. The LTL translator then yields the final LTL formulae based on the derived actions.

categories as its parameters:

$$\mathtt{a} = \mathtt{Verb}(\mathtt{What?}, \mathtt{Where?}, \mathtt{How?}, \mathtt{Time}, \mathtt{Temperature})$$

The robot is tasked with executing a cooking recipe R that consists of a list of k instruction steps $\{r_1, r_2, \ldots, r_k\}$, where each instruction step r_i is an imperative sentence in natural language describing a robot command. Each instruction step r_i may include one or more cooking actions. Our goal is to generate a set of task specifications written in the form of a set of LTL formulae $\Phi = \{\phi_1, \phi_2, \ldots, \phi_n\}$ that implement the recipe under the constraint of **only** including actions that belong to the set of primitive actions $\mathcal{A}$ that the robot is capable of executing.

3.2.2 System Architecture

To solve this problem, we propose Cook2LTL, the system architecture summarized in Fig. 3.2. Given an instruction r_i and a set of actions $\mathcal{A}$, Cook2LTL:

1. Semantically parses r_i into a function representation $\mathtt{a}$ for every detected high-level action.

2. Reduces each high-level action $\mathtt{a} \notin \mathcal{A}$ to a combination of primitive actions from $\mathcal{A}$.

3. Caches the action reduction policy for future use, thereby gradually building an action library that consists of parametric functions that express high-level cooking actions in the form of primitive actions.

4. Translates r_i into an LTL formula ϕ_i with function representations as atomic propositions.

Algorithmically, these steps are summarized in Alg. 1. In the following subsections, we expand on the components of Cook2LTL in more detail.

Algorithm 1 Cook2LTL

Input: A high-level instruction step r, a set of primitive actions $\mathcal{A}$, and an action library $\mathbb{A}$
Output: An LTL action formula ϕ

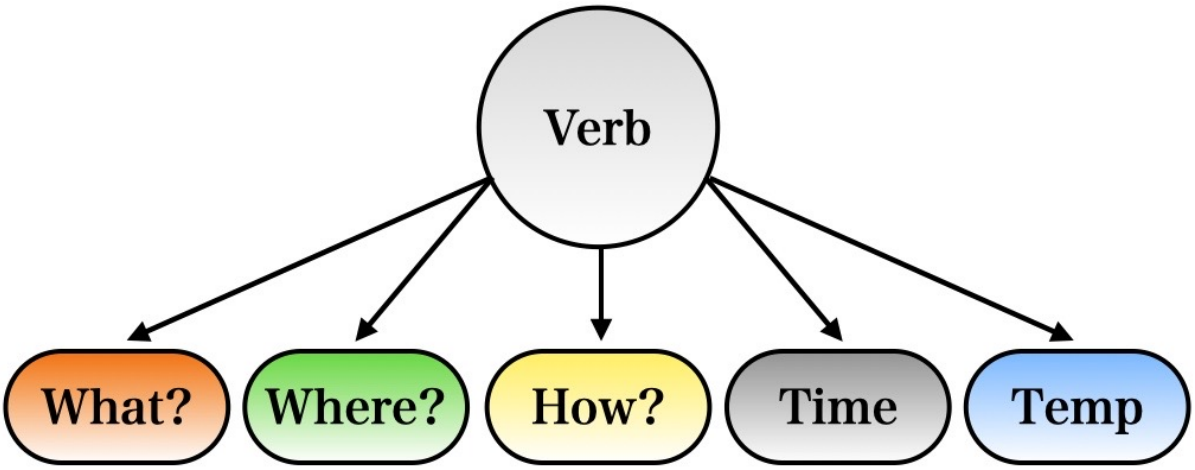
$\mathbb{A} \leftarrow \mathbb{A} \cup \mathcal{A}$
$r \leftarrow f_{PRE}(r)$ ▷ Preprocessing
$\{\mathtt{a_1}, \mathtt{a_2}, \ldots, \mathtt{a_n}\} \leftarrow f_{SP}(r)$ ▷ Semantic Parsing
$A \leftarrow \{\mathtt{a_1}, \mathtt{a_2}, \ldots, \mathtt{a_n}\}$
$\phi \leftarrow f_{LTL}(\mathtt{a_1}, \mathtt{a_2}, \ldots, \mathtt{a_n})$ ▷ Initial LTL Translation
for $\mathtt{a}_i \in A$ **do**
    if $\mathtt{a_i} \notin \mathbb{A}$ **then**
        $\{a_1, a_2, \ldots, a_k\} \leftarrow f_{AR}(\mathtt{a}_i)$ ▷ Action Reduction
        $\mathtt{a_i} \leftarrow \{a_1, a_2, \ldots, a_k\}$
        $\mathbb{A} \leftarrow \mathbb{A} \cup \{\mathtt{a} \rightarrow a_1, a_2, \ldots, a_k\}$ ▷ Caching
    end if
end for
$\phi \leftarrow f_{LTL}(A)$ ▷ Final LTL Translation
return ϕ

3.2.3 Semantic Parsing and Data Annotation

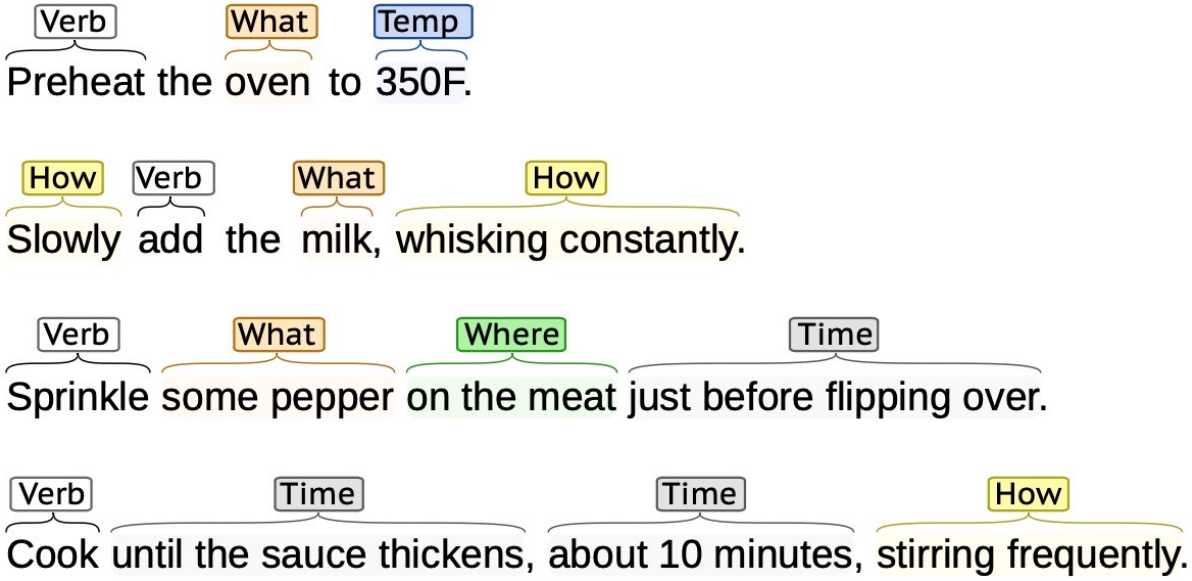
Our translation system requires a semantic parsing module capable of extracting meaningful chunks corresponding to the parametric function representation components of a cooking action. To this end, we fine-tune a named entity recognizer with the addition of salient categories $\mathcal{C}$ as labels. We choose a neural approach over a syntactic parse because the latter would require arduous manual rule crafting for every different mapping of part-of-speech (POS) tags to these categories. Additionally, explicit POS-tagging-based approaches often struggle with handling the intricacies of cooking discourse, such as imperative form sentences omitting context-implicit parts of speech.

In the absence of a labeled dataset with a schema matching $\mathcal{C}$, we create our own data building upon the large cooking recipe dataset Recipe1M+ [6]. Specifically, we consider a subset of 100 recipes from Recipe1M+, leading to 1000 recipe instruction steps. We use brat [93] to manually annotate chunks in each step corresponding to the following salient categories: $\mathcal{C} = \{$`Verb, What?, Where?, How?, Temperature, Time`$\}$, which is a similar annotation scheme as the one seen in recent work [94].

Fig. 3.3 shows these categories and a set of example recipe steps taken from Recipe1M+ [6]. `Verb` is the main action verb in a recipe step. `What?` represents the direct object of the `Verb` and is often an ingredient, but can correspond to other entities such as a kitchen utensil or an appliance. `Where?` is usually a prepositional phrase, it implies a physical location (e.g. table, bowl) but can often be an ingredient to which the `Verb` applies. `How?` is usually either a gerund form of a verb, expressing concurrency and hence giving rise to a secondary cooking action, or complements the main cooking action (e.g. "Drizzle *with olive oil*"). The `Time` category consists of temporal expressions composed of keywords that are important for the translation of the commands to LTL formulae (*until*, *before* etc.). Finally, `Temperature` can explicitly list the degrees (e.g. 350F) to which food should be cooked or refer to a temperature-related state of some ingredient (e.g. *medium heat*). These salient categories form the function representation of an action found in r_i.



(a) Salient categories $\mathcal{C}$ considered for semantic parsing.



(b) Recipe steps annotated with the salient categories $\mathcal{C}$

Figure 3.3: We annotate Recipe1M+ [6] instruction steps with the salient categories $\mathcal{C}$ ={`Verb`, `What?`, `Where?`, `How?`, `Temperature`, `Time`} and fine-tune a named entity recognizer to segment chunks corresponding to $\mathcal{C}$.

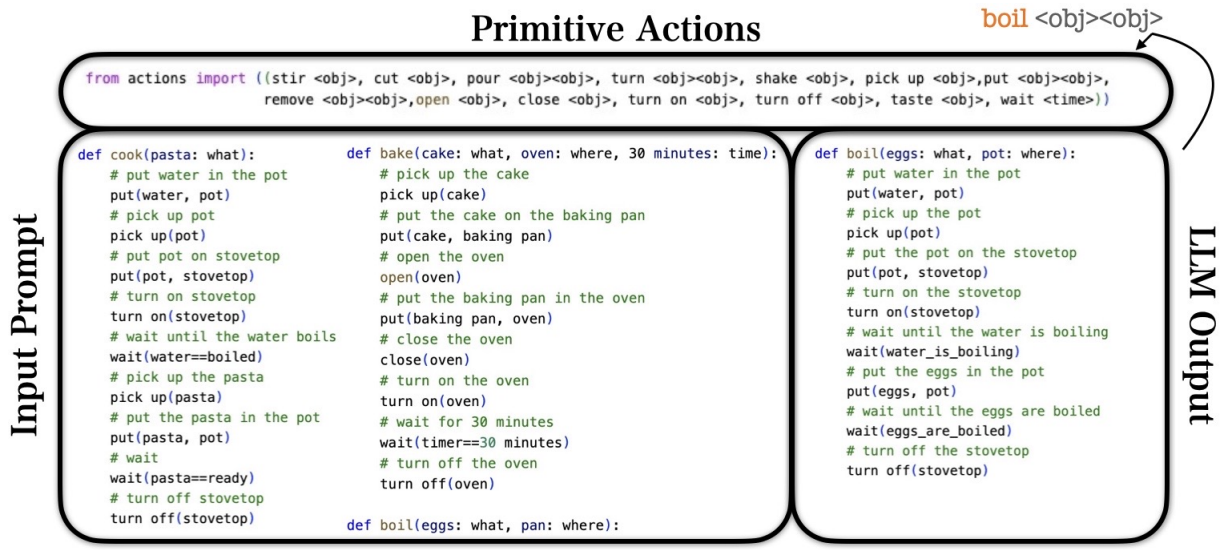


Figure 3.4: Inspired by ProgPrompt [23], Cook2LTL uses an LLM prompting scheme to reduce a high-level cooking action (e.g. `boil eggs`) to a series of primitive manipulation actions. The prompt consists of an import statement of the primitive action set and example function definitions of similar cooking tasks. The key benefit of using this paradigm is that it constrains the output action plan of the LLM to only include subsets of the available primitive actions. We extend this prompting scheme by reusing derived LLM policies. In this case, the action `boil` is added to future import statements in the input prompt, enabling the model to invoke the derived `boil` function which is now considered given to the system.

3.2.4 Reduction to Primitive Actions

Some of the function representations captured in the previous step contain high-level actions that might not be directly executable by the robot, which can only execute actions that belong to the primitive set $\mathcal{A}$. Therefore, our system requires a module capable of mapping an action $\mathtt{a} \notin \mathcal{A}$ to an action $a \in \mathcal{A}$, if possible, or reducing $\mathtt{a}$ to a sequence of actions $a_1, a_2, \ldots, a_k$ where $a_i \in \mathcal{A}, i = 1, 2, \ldots, k$. Our system initially checks whether $\mathtt{a} \in \mathcal{A}$ to validate a formula for execution, and if $\mathtt{a} \in \mathcal{A}$, $\mathtt{a}$ is forwarded to the LTL translator.

LLM Action Reduction: If $\mathtt{a} \notin \mathcal{A}$ we employ an LLM-based methodology inspired by the work in [23] to extract a lower-level plan exclusively consisting of primitive actions from $\mathcal{A}$. Specifically, we design an input prompt consisting of: i) a pythonic import of the available actions in the environment, ii) two example function definitions decomposing high-level cooking actions into primitive sets of actions from $\mathcal{A}$, iii) the function representation $\mathtt{a}$ extracted by the semantic parsing module in the form of a pythonic function name with its parameters. As shown in [23] and Fig. 3.4, the LLM follows the style and pattern of the input function and only includes available actions in the output. The key advantage of this method is the flexibility in changing the admissible primitive actions depending on the robot capabilities and the environment. This change can simply be achieved by modifying the primitive actions in the pythonic import.

Action Library: Extending ProgPrompt [23], every time we query the LLM for action reduction, we cache $\mathtt{a}$ and its action decomposition for future use through a dictionary lookup manner. This gradually builds a dynamic knowledge base in the form of an executable action library $\mathbb{A}$ consisting of various high-level actions along with their function bodies made out of primitive actions from $\mathcal{A}$. At runtime, instead of only checking whether a detected action $\mathtt{a}$ matches an action $a \in \mathcal{A}$, we additionally check if $\mathtt{a} \in \mathbb{A}$. In case there is a match, we replace $\mathtt{a}$ with the action in $\mathbb{A}$. Additionally, we add $\mathtt{a}$ to the pythonic import part of the prompt, allowing the model to invoke it when generating future policies (e.g. the action `boil` in Fig. 3.4). The key benefit comes from avoiding to continuously query an LLM

for action reduction, thus replacing potential latency resulting from an LLM API call with a fixed $\mathcal{O}(1)$ dictionary lookup time. It also reduces the cost associated with querying a proprietary LLM API.

3.2.5 LTL Translation

The final step in our pipeline translates the intermediate function representations acquired from semantic parsing and action reduction into an LTL formula. The implicit sequencing of recipes is elegantly captured by the sequenced visit specification pattern [95]:

$$F(l_1 \wedge \mathcal{F}(l_2 \wedge \ldots \mathcal{F} l_n))) \tag{3.3}$$

This pattern has been used [67, 70, 96] to model a visit of a set of locations $L = \{l_1, l_2, \ldots, l_n\}$ in sequence one after the other in a navigational setting, adapted to the execution of consecutive cooking actions $\mathtt{a}_1, \mathtt{a}_2, \ldots, \mathtt{a}_n$ in our case. Building on this pattern, we acquire conjunction, disjunction, and negation constituents for each segmented chunk corresponding to the categories $\mathcal{C}$ through a dependency parse. Then, we write down a formula ϕ which includes high-level actions $\mathtt{a}$ with a combination of the following LTL operators $\{(\mathcal{F} : \texttt{Finally}), (\wedge : \texttt{and}), (\vee : \texttt{or}), (\neg : \texttt{not})\}$. Every action $\mathtt{a}_i$ is translated to one or more primitive actions from $\mathcal{A}$. In the latter case, the generated low-level plan for $\mathtt{a}_i$ is parsed into a subformula ψ_i based on Equation 3.3. The `Time` parameter passed to the action reduction LLM often includes explicit sequencing language (such as *until*, *before*, or *once*). The LLM has been prompted to return a `Wait` function in these cases (see example in Fig. 3.4), which is then parsed into the $(\mathcal{U} : \texttt{until})$ operator and substituted in ψ. The final formula ϕ consists of subformulae $\psi_1, \psi_2, \ldots, \psi_n$ comprised by primitive actions in $\mathcal{A}$:

$$\phi = F(\mathtt{a}_1 \wedge \mathcal{F}(\mathtt{a}_2 \wedge \ldots \mathcal{F} \mathtt{a}_n))) = F(\psi_1 \wedge \mathcal{F}(\psi_2 \wedge \ldots \mathcal{F} \psi_n))) \tag{3.4}$$

where:

$$\begin{cases} \psi_i = a_i & , a_i \in \mathcal{A} \text{ ,or} \\ \psi_i = f(a_1, a_2, \ldots, a_k, \mathbb{O}) & , \mathbb{O} = \{\mathcal{F}, \wedge, \vee, \neg, \mathcal{U}\} \end{cases} \tag{3.5}$$

3.3 Evaluation

3.3.1 Ablation Study

To investigate the performance of Cook2LTL, we conduct an ablation study against two variants. For each run, the input is a recipe from a held-out subset of Recipe1M+ and the output is a series of task specifications in the form of LTL formulae Φ towards executing the recipe under the constraints of admissible actions $\mathcal{A}$. In all the experiments we use the OpenAI API and the *gpt-3.5-turbo* model. The initial preprocessing step consists of filling in the implicit objects (zero anaphora resolution) in the recipes and segmenting each recipe into sentences. We begin by deploying a partial version of our system (AR*) as a baseline, consisting of the preprocessing, semantic parsing, and action reduction modules. We expect that our action reduction policy adheres to the admissible actions of the environment by a significant amount. We incrementally add the functionality of invoking cached policies, first when encountering a primitive action (AR), and then when an action is found in the action library (AR+$\mathbb{A}$), starting from an empty library and gradually building it with the LLM-generated policies along the way. We anticipate a significant benefit in terms of computational load and cost efficiency resulting from capitalizing on reusable policies, compared to querying the action reduction LLM for every unseen action encountered at runtime. We formalized these insights into the following hypotheses:

H1: Our action reduction policy generation constrains the LLM output to the admissible actions $\mathcal{A}$ in our environment.

H2: Our enhanced Cook2LTL system that includes the action library component is more time- and cost-efficient than the baseline action reduction-comprised partial system.

Figure 3.5: Tasks we tested Cook2LTL in AI2-THOR (left to right): `microwave the potato`; `chop the tomato`; `cut the bread`; `refrigerate the apple`.

	Active Modules		
Metric	AR*	AR	Cook2LTL (AR+𝔸)
Executability (%)	0.91 ± 0.01	0.92 ± 0.01	$\mathbf{0.94 \pm 0.01}$
Time (min)	14.85 ± 1.05	9.89 ± 0.46	$\mathbf{6.05 \pm 0.12}$
Cost ($)	0.19 ± 0.01	0.16 ± 0.00	$\mathbf{0.11 \pm 0.00}$
API calls (#)	275 ± 0.00	231 ± 0.00	$\mathbf{134 \pm 0.00}$

Table 3.1: Performance of Cook2LTL against baselines across 50 Recipe1M+ [6] recipes (10 runs per recipe).

To evaluate these hypotheses, our metrics are: 1. *Executability (%)*, which is the fraction of actions in the generated plan that are admissible in the environment; 2. *Time (min* or *sec)* which measures the runtime influenced by the LLM API calls; 3. *Cost ($)* which is the overall cost for a batch of experiments and depends on the number of input and output tokens; 4. the number of the LLM *API calls*.

3.3.2 Results & Discussion

Based on the quantitative results in Table 3.1 we make the following observations regarding our hypotheses.

	AR		Cook2LTL (AR+$\mathbb{A}$)	
Task	SR (%)	Time (sec)	SR (%)	Time (sec)
Microwave the potato	5.4 ± 1.95	27.29 ± 3.66	$\mathbf{8 \pm 4.47}$	$\mathbf{3.26 \pm 1.30}$
Chop the tomato	2.4 ± 1.52	16 ± 0.96	$\mathbf{4 \pm 5.47}$	$\mathbf{1.61 \pm 0.76}$
Cut the bread	$\mathbf{9 \pm 0.71}$	12.85 ± 0.84	8 ± 4.47	$\mathbf{1.12 \pm 0.16}$
Refrigerate the apple	7.6 ± 0.55	14.6 ± 0.38	$\mathbf{8 \pm 4.47}$	$\mathbf{1.56 \pm 0.44}$

Table 3.2: We demonstrate the performance of Cook2LTL on 4 simple cooking tasks in AI2-THOR. We observe that Cook2LTL (AR+$\mathbb{A}$) is time efficient but propagates initial incorrect LLM-generated sets of actions to subsequent runs.

H1: Our first hypothesis is confirmed. In every part of the ablation study the system has a high executability with a maximum value of 94% when using the action library. This is a natural consequence of incorporating a new action in the prompt every time it is decomposed to sub-actions by the LLM. The policies for the cached actions are now part of the system, and hence they are considered admissible in the environment, leading to an increased executability value.

H2: The enhanced action library-based Cook2LTL system (AR+$\mathbb{A}$) outperforms the baseline (AR*) and primitive action-focused variant (AR) in all 4 metrics. We have discovered that learning new action policies through prompting an LLM and reusing them in a dictionary lookup manner in subsequent recipes decreases the number of API calls by 51% and 50% compared to the AR* and AR versions of the system. Consequently, a lower number of API calls leads to a significantly reduced runtime and cost. More specifically, the integration of the action library into our system decreases runtime by 59% and 42% compared to the AR* and AR versions, and cost by 42% and 31%, respectively.

3.3.3 Demonstration in AI2-THOR

We demonstrate the performance of Cook2LTL in a simulated AI2-THOR [7] kitchen environment (See Fig. 3.1). AI2-THOR has a small set of ingredients and objects and hence cannot support the

full execution of recipes found on the web; however the limited action space aligns with the notion of primitive actions and offers room for highlighting the key ideas of our system. To showcase the potential of our approach, we constructed a set of 4 kitchen tasks that are admissible in AI2-THOR and executed them by invoking Cook2LTL. We assume that the kitchen is *mise en place* so the locations of the objects are known to the agent. In AI2-THOR, we design a minimal parser that receives an LTL formula and converts it to a series of actions. We adapt the imported primitive actions and example functions in the prompt to the ones that are supported in the simulation. Fig. 3.5 contains screenshots from our experiments. We run 5 sets of experiments where we execute each task 10 consecutive times. We measure the success rate SR and execution time due to the LLM API calls and compare the performance of the AR and Cook2LTL (AR+$\mathbb{A}$) variants. The success rate is the fraction of executions that achieved the task-dependent goal conditions (e.g. *tomato=sliced*) that we defined a priori. During our simulations we observe that Cook2LTL is still significantly more time efficient compared to baselines, however its SR is entirely dependent on the first LLM-generated plan, and fails when this plan is not executable (See Table 3.2).

3.4 Limitations & Future Work

System: We annotated a small part of the Recipe1M+ dataset [6] with our salient categories but we would need more data to improve the entity recognizer for reliably transferring the system to a real-world robot. Finally, some actions being substituted by action library policies lead to non-executable plans. Our system would benefit from an additional mechanism that robustly ensures the correctness of the LLM-generated plans based on environment feedback.

Sim2real: AI2-THOR is not tailored towards simulating cooking tasks but rather supports the general area of task planning. Thus, we would need a cooking-specific simulator to support a more diverse set of recipes that correspond to the rich web-scraped recipes that we built our system on. In terms of transferring simulation to a real robot, we plan to use the Yale-CMU-Berkeley (YCB) Object

and Model set [97] towards supporting a basic set of simple cooking tasks for benchmarking preliminary experiments.

Task representation: The final layer of our system uses LTL as an expressible notation tool capturing temporal task interdependence, but our system is compatible with other task representations, such as PDDL [98], which incorporates action preconditions and postconditions in the problem setting and has recently been explored with LLMs [57, 60].

Chapter 4: Discovering Object Attributes by Prompting Large Language Models with Perception-Action APIs

4.1 Method

4.1.1 Problem Statement

Consider a robot equipped with a set of sensors $\mathcal{S}$ in a scene with a set of objects $\mathcal{O}$. The robot is tasked with executing a natural language instruction $inst = f(a, g, o, img)$, where a is a high-level action, g is an object attribute, o is an object, and img is an input image. Our goal is to determine whether an object exists with this attribute, expressed by the predicate $g(o)$, and localize it in img by obtaining its bounding box coordinates $\mathcal{X} = \{x_{min}, y_{min}, x_{max}, y_{max}\}$. If $\exists o$, such that $g(o)$ holds, a visual navigation policy $\pi(a(\mathcal{X}))$ is deployed, which allows the robot to leverage its sensors S, navigate and manipulate object o given its 2D bounding box coordinates $\mathcal{X}$ towards task completion.

We adopt a generalized definition of an attribute, viewing it as an abstract property of an object that does not necessarily map to a visual representation, including primarily descriptive adjectives related to the size (*big*) or weight (*heavy*) of an object, but also prepositional phrases indicating spatial relationships (*the second object from the left*).

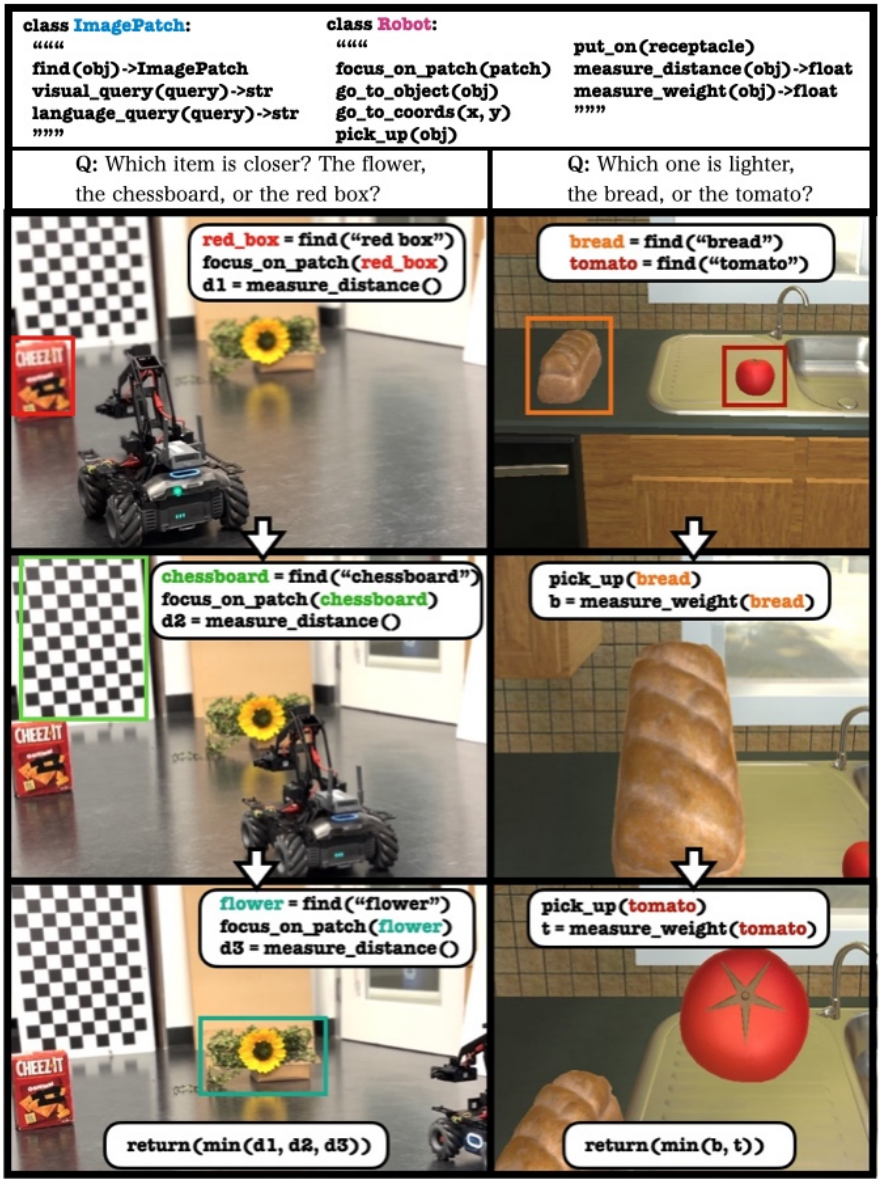


Figure 4.1: Demonstration of our perception-action API solving a minimum distance query on a real robot (*left*) and a minimum weight query in simulation (*right*). The LLM receives a perception-action API and a natural language query as input (*top*). It then generates code that invokes API functions leveraging on-board sensors (camera, distance sensor, force/torque sensor) to actively identify these attributes.

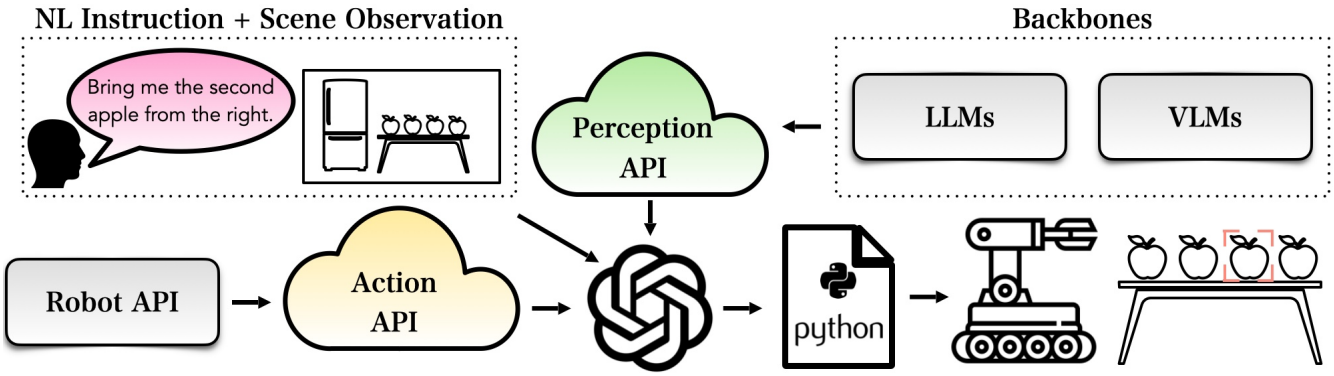


Figure 4.2: We describe our end-to-end framework for embodied attribute detection. The LLM receives as input a perception API with LLMs and VLMs as backbones, an action API based on a Robot Control API, a natural language (NL) instruction from a user, and a visual scene observation. It then produces a python program that combines LLM and VLM function calls with robot actions to actively reason about attribute detection.

4.1.2 Prompt-based Attribute Detection

We adopt the methodology of Surís et al. [87] into constructing a Python API for attribute detection. The API consists of a main `ImagePatch` class that is instantiated by an input image img. `find` is the fundamental function of the API that uses an OVD model (MM-Grounding-DINO [99] or GLIP [100]) to locate an object o and return the detection-resulting cropped patch $\mathcal{X}$ from the image given a chunk of natural language $inst$. While we do not explicitly define a function for spatial reasoning, we provide in-context examples encoded in the docstring of the function. The examples perform pixelwise math given bounding box coordinates $\mathcal{X}$ of detected objects $\mathcal{O}$ that are returned from calls to `find` in order to reason about relative object locations on the image frame. `visual_query` calls a pre-trained VLM (BLIP-2 [101]) to provide a textual answer to a visual query given an image. `language_query` recursively calls the LLM with a textual query such as the visually-extracted return value from `visual_query`. This API can be viewed as an internal dialogue between VLMs and LLMs, similar to the idea of Socratic Models [48], but enhanced by structural programming tools hosted on a pythonic platform. In the following subsections we describe the complementary reasoning capabilities that emerge from calling

these functions.

4.1.3 Programmatic Reasoning

Reasoning in the form of programs inherits the expressiveness of programming languages through control flow tools, data structures, and built-in methods. LLM-generated programs invoke loops to iterate over detected object patches and conditional statements to determine whether an object exists ($\exists o$) in the input image img and whether it possesses an attribute g, grounding the predicate $g(o)$. Python lists are used to store instances of image patches dynamically with the `append` function. Other built-in functions such as `sort` and the `lambda` function utilize horizontal and vertical coordinates of the detected bounding boxes and their centroids within simple mathematical operations and leverage basic geometrical notions (e.g. computing the area of an image patch) to reason about the size or relative position of detected entities. The generated code is interpretable and mainly consists of elementary arithmetic in the image frame. The commonsense reasoning component of the LLM proves to be essential in mapping complex language queries to these computations, as well as adapting attribute interpretation to image-specific contexts, as shown in Sec. 4.2.5.

4.1.4 Vision-Informed Language Reasoning

Combining LLMs and VLMs in the input API prompt unlocks complementary reasoning capabilities through information passing between different model calls in the the context of an LLM-generated program. This model interplay can be particularly efficient when dealing with non-visually perceivable attributes, such as estimating the weight of an object. In this case, `visual_query` can serve as a zero-shot object recognition function, subsequently passing information to the input of the `language_query`, which deduces factual knowledge on the weight of the recognized object, as explained in Sec 4.2.2. Similarly, when a task instruction involves an attribute that does not typically describe an object in an absolute scale, recognizing adjacent objects in addition to the object at hand

establishes context. Then, posing a textual query with this visually-acquired information might reduce ambiguity and provide the correct grounding thanks to the domain knowledge of LLMs. We demonstrate an empirical evaluation of such use cases in Sec. 4.2.

4.1.5 Embodied Attribute Detection

Attribute detection in embodied settings often requires active perception. To this end, we formulate an action-perception API by integrating the attribute detection API with a high-level robot control API (See Fig. 4.2). The robot control API is implemented as a `Robot` Python class that consists of sensors as member variables and methods that map to simple navigation and pick-and-place actions. The robot can navigate to an object with the `go_to_object` function which implements a visual navigation policy by calling `go_to_coords` with an image patch $\mathcal{X}$ as a parameter. `pick_up` and `put_on` implement picking and placing actions. Assuming the lack of an on-board RGB-D camera or a depth estimation model, a robot could employ additional sensors to measure the distance to an object in order to reason about scene geometry or depth. This can be achieved by the `measure_distance` function which calls `focus_on_patch`, a function that aligns the geometric center of the image frame to an object patch and can then retrieve the distance sensor measurement to compute the distance from the camera to that object. A demonstration on a real robot is shown in Fig. 4.1 (*left*). Similarly, `measure_weight` can measure the weight of an object grasped by the robot, under the precondition that the robot first navigates and picks it up. These preconditions are encoded in the example use of the function in a docstring. We integrate this perception-action API into an AI2-THOR simulated environment and a real robot and demonstrate its benefits in Sec. 4.2.5.

4.2 Evaluation

We design a set of experiments to showcase some of the drawbacks of using OVD or VQA models for attribute grounding in isolation and we highlight the complementary commonsense reasoning that

emerges by visual reasoning with LLM-generated programs including actions.

4.2.1 Spatial Reasoning

We evaluate the spatial reasoning capabilities of the attribute detection API by comparing its ability to ground linguistically complex spatial queries with an open-vocabulary object detector [100]. We manually craft a dataset that consists of 200 challenging spatial queries based on the Odd-One-Out (O^3) Dataset [102]. Every image in this dataset includes multiple instances of an object or similar objects with an instance being slightly different to stand out. We leverage the multiple instances of an object to invoke reasoning that requires differentiating between objects based on their relative attributes rather than obvious qualitative differences between entirely different objects. Therefore, instead of focusing on relative attributes that localize the object with respect to another object of different type in the image [87, 103] (e.g. "*the car to the left of the tree*"), our 100 **location** queries require commonsense reasoning in the form of counting and establishing the relative order of an arranged set of objects, such as "*second umbrella from the left at the second to last row*" or "*the window in the middle at the bottom*". Our 100 **size** queries utilize descriptive size-related adjectives (long, wide, short, large etc.) in their superlative and absolute form, such as "*the tallest item*" or "*the wide line*", respectively. We test the same queries on both forms and expect that the superlative form will outperform the absolute, forcing a specific object to stand out by emphasizing its attribute. We anticipate that the attribute detection API will outperform OVD by incorporating pixelwise mathematical operations and expressive python utility functions.

4.2.2 Non-visually Perceivable Attributes

To evaluate more profound reasoning capabilities, we focus on the **weight** of an object as a representative sample of non-visually perceivable attributes, as it is crucial for executing essential manipulation tasks. In the absence of a dataset with a suitable schema for our use case, we prompt GPT-4 to generate sets of objects of different weight, along with the ground truth label of the heaviest

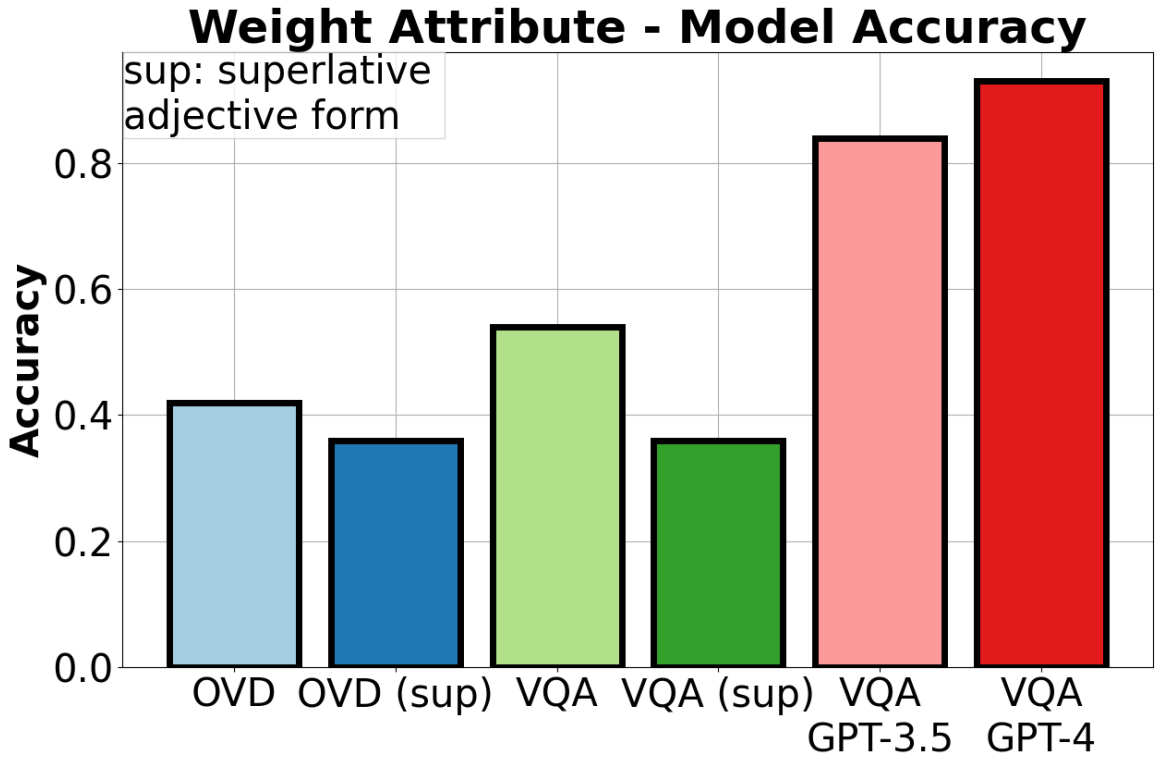


Figure 4.3: The accuracy of OVD (GLIP), VQA (BLIP-2), and VQA+GPT in determining the heaviest object in an image.

object. To simplify the task, we design the prompt so that the object weight distribution is monotonic and clearly distinguishable by a human observer: *Generate* 100 *triplets of objects where each object is significantly heavier than the other (for example: feather, dog, car).* After acquiring the generated textual data, we utilize it to extract relevant images from the web and arrange them to form an image dataset where each data sample is an image that includes three objects of monotonically decreasing weight. Assuming some rudimentary commonsense reasoning functionality in VLMs [104], we expect that they are capable of identifying obvious differences in weight and hence dealing with examples that are intuitive to humans, such as selecting the heaviest object between a handbag, a kangaroo, and a bus (See Fig. 4.5). We compare the performance of OVD: `find(`*"a heavy object"*`)`, VQA: `visual_query(`*"Out of these items, which one is the heaviest?"*`)`, and vision-informed language reasoning (VQA+GPT) that is invoked by our prompt API: `visual_query(`*"What are the items in this image?"*`)` $\rightarrow$ `language_query(`*"Out of these items, which one is more likely to be the heaviest one?"*`)`.

4.2.3 Evaluation in Embodied Settings

To evaluate our perception-action API in embodied settings, we adapt it to a simulated AI2-THOR [7] household environment. We assume that the robot comes with a proximity sensor and a force/torque sensor mounted on the wrist of the gripper, capable of measuring the weight of an object. To replicate the behavior of these sensors, we query the simulator for the distance between an object centered on the frame captured by the on-board robot camera, and build a queryable dictionary that maps an object to an approximate weight when the robot is holding that object. We measure the accuracy (%) of our perception-action API in estimating the relative distances of objects from the robot camera, and identifying the most lightweight object. Our baselines are OVD, VQA, GPT-4o, and the attribute detection API (VQA/OVD+GPT-4). We use the following prompt templates: "*Out of the* {*objects*}*, which one is closer to me?*", "*Out of the* {*objects*}*, which one is the most lightweight?*". We anticipate

that the LLM-generated programs from the perception-action API are capable of actively interacting with the environment to identify object attributes leveraging sensor-powered visual reasoning functions and robot actions.

4.2.4 Hypotheses

We formalize these insights into the following hypotheses:

H1: OVD+GPT outperforms OVD- and VQA-only baselines in **location**- and **size**-related queries.

H2: VLMs possess the rudimentary reasoning capability to tackle evident **weight** estimation queries.

H3: The superlative form of a descriptive adjective yields a better grounding performance than the absolute form.

H4: Our perception-action API solves attribute detection queries by actively interacting with the environment.

To evaluate these hypotheses, we measure the grounding accuracy by comparing the bounding boxes returned by OVD and OVD+GPT. For OVD, we report results from GLIP [100] since it outperforms MM-Grounding-DINO [99] on our data. In the case of the weight attribute, we additionally consider the textual output of VQA+GPT. We demonstrate results from GPT-3.5, GPT-4, and GPT-4o in the embodied settings.

4.2.5 Results & Discussion

Based on the results in Fig. 4.3, Fig. 4.4, and Table 4.1 we make the following observations regarding our hypotheses.

H1: Our first hypothesis is confirmed. We find that OVD+GPT significantly outperforms OVD (See Fig. 4.4) in **location** queries by 134% and in **size** queries by 67%. Qualitative examples are shown in Fig. 4.6. In example *a*, OVD and OVD+GPT correctly identify the apple as the small fruit. Example *b*

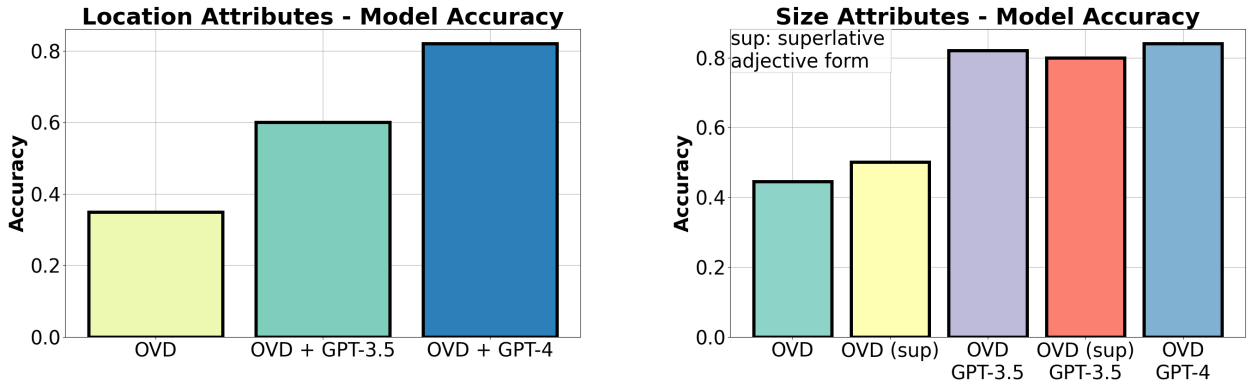


Figure 4.4: We compare the accuracy of OVD-only (GLIP) with (OVD+GPT) on our **location** (*left*) and **size** (*right*) datasets.

shows a common mutual failure case where the bounding box for one paper clip mistakenly includes all of them. We also observe failure cases due to occlusion. Examples *c*, *d* show cases where OVD+GPT outperforms OVD. In these examples, all instances of the object in the query are localized with the `find` function, and then the coordinates of the bounding boxes and their centroids are used to compute relative distances and areas and sort them, if needed. In terms of the **size**-related queries, GPT-4 can adapt the generated code to the context of an image and interpret what dimension *long* or *short* corresponds to based on the orientation of an object in an image. On the other hand, GPT-3.5 is more rigid and tends to tie certain adjectives to hardcoded dimensions following common norms. For example, in Fig. 4.6 (*c*), it cannot connect the adjective *short* to the red pepper, since it is horizontally aligned. Such minor details explain the slight discrepancy in performance, which is still superior than vanilla OVD when identifying attributes with an attribute detection API. We would need to add targeted examples in the prompt API covering all failure cases to induce equal accuracy from both models. Finally, in example *d*, OVD+GPT-4 understands that the arrangement of the tarts is forming a row and column pattern. On the contrary, OVD fails to recognize this pattern, and OVD+GPT-3.5 exhibits a context-agnostic interpretation of rows, dividing the image into parts based on the image height, which yields incorrect results.

H2: Our second hypothesis is only partially confirmed. Based on Fig. 4.3, the combination of

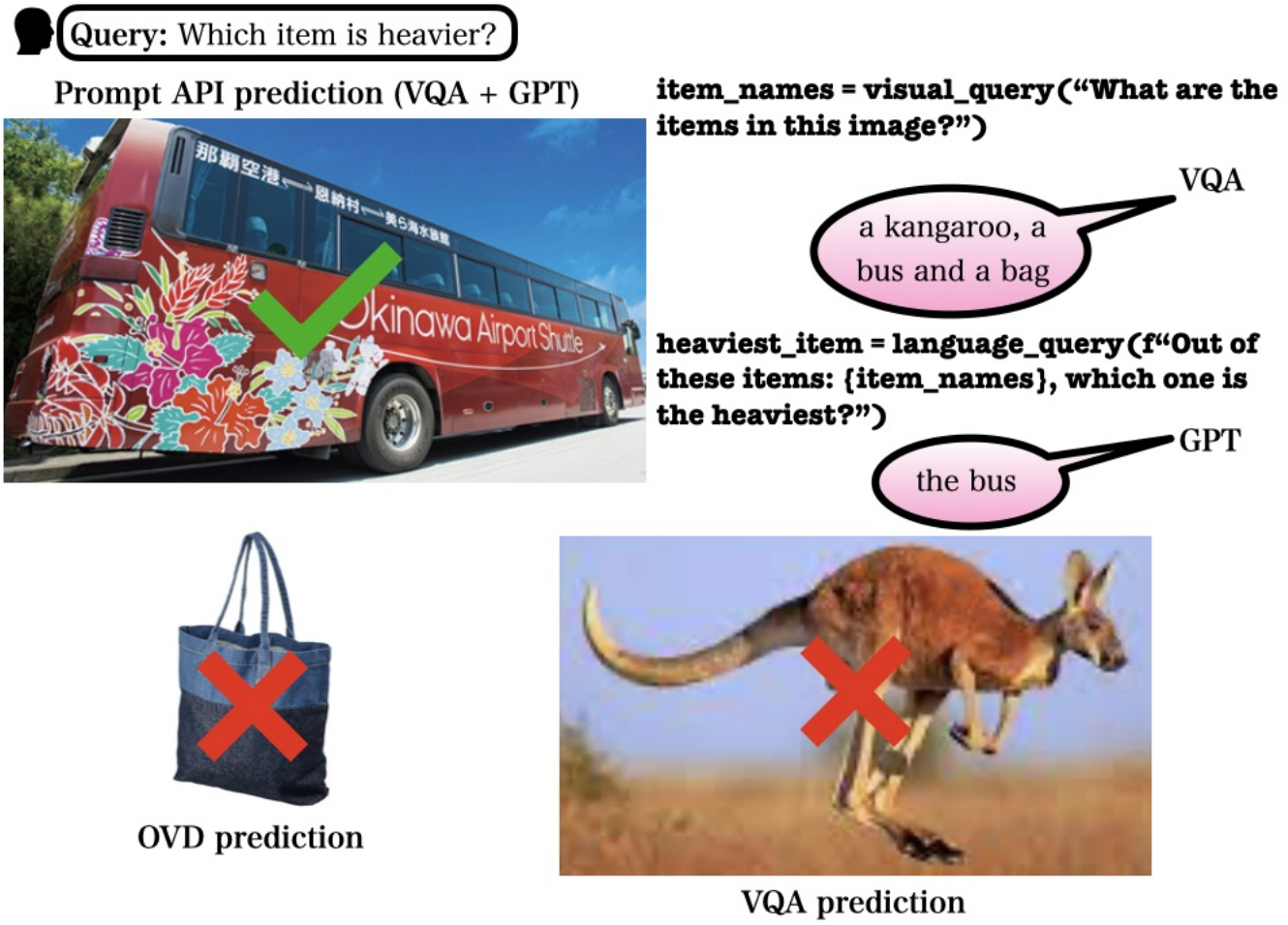


Figure 4.5: An example where OVD and VQA fail to identify the heaviest object in the image (), but the API prompt-generated code (VQA+GPT) returns the correct answer ().

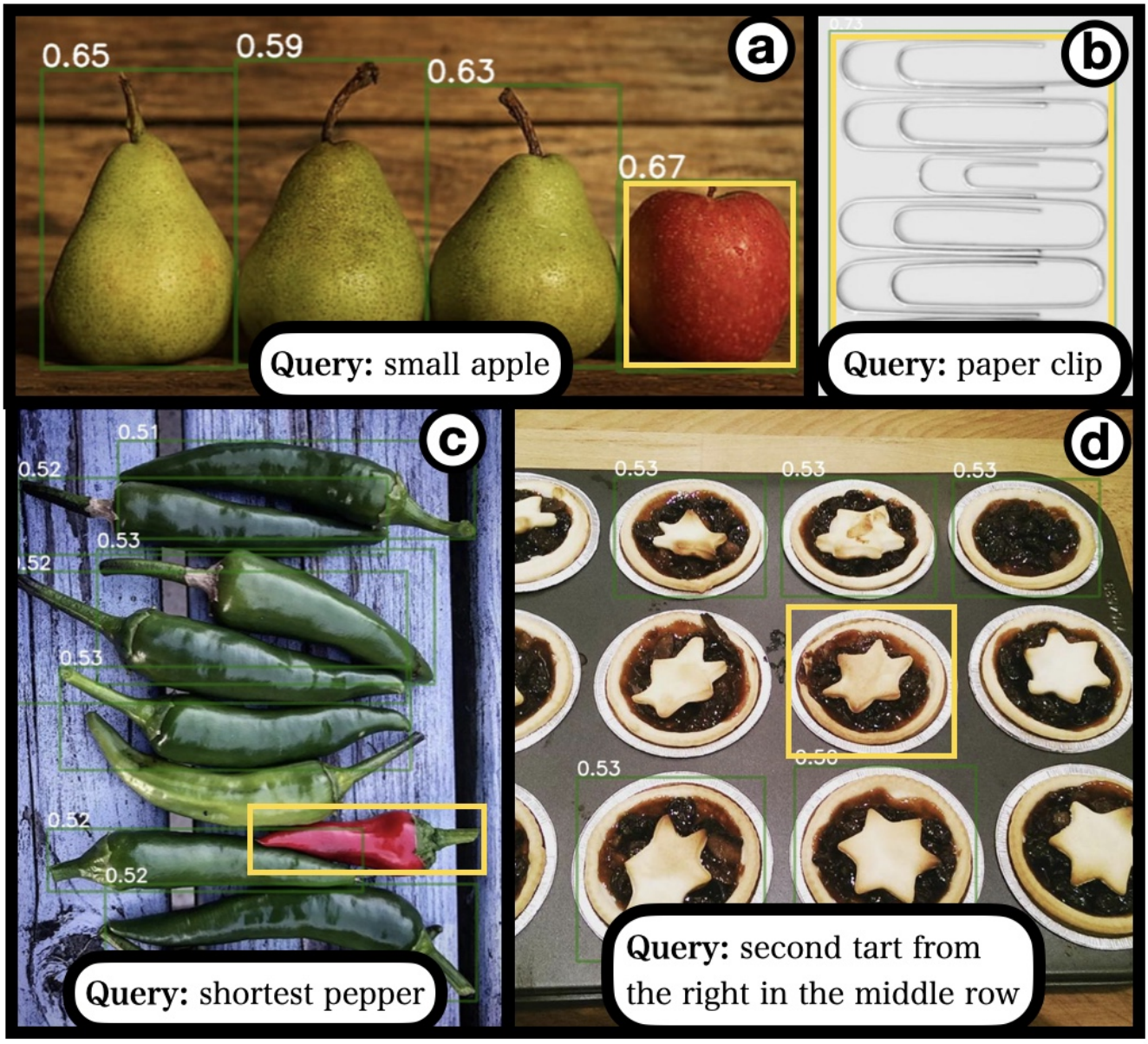


Figure 4.6: OVD (numbers in white denote the OVD confidence score) and OVD+GPT predictions are shown with green and yellow bounding boxes, respectively. *a* shows an example of agreement between OVD and OVD+GPT, *b* a mutual failure case, and *c*, *d* show cases where OVD+GPT exhibits superior performance compared to OVD.

.

Method	Task	
	Weight	Distance
OVD	0.14	0.64
VQA	0.64	0.56
Attribute Detection API	0.90	0.22
GPT-4o	0.88	0.70
Perception-Action API	**0.96**	**0.94**

Table 4.1: Performance of our perception-action API in 50 weight and 50 distance estimation queries in simulated AI2-THOR [7] household environments against baselines.

VQA, followed by a call to an LLM, significantly outperforms OVD-only (+121%) and VQA-only (+72%) solutions although we expected that all the models would be able to handle simple comparative attribute detection tasks. We believe that the step-by-step reasoning process followed by the prompt API leverages the strength of each model separately. On the contrary, burdening a model with additional reasoning tasks upon the ones that it was naturally tasked with in the first place (zero-shot language-conditioned object detection for GLIP, and zero-shot object recognition for BLIP-2) might be the reason we are missing out on its full task-specific potential. Another potential reason for this discrepancy in performance is that the pre-training objective of these models might not be aligned to our specific use case, which is identifying non-visually perceivable attributes.

H3: Our third hypothesis is rejected, mainly because the absolute form of an adjective yields better performance in the weight estimation task (See Fig. 4.3). In Fig. 4.3, Fig. 4.4, *sup* stands for superlative adjective form in the prompt. GPT-4 demonstrates the same performance for both forms, therefore we report a joint measurement for this model in Fig. 4.4.

H4: Confirming our hypothesis, our perception-action API solves both tasks and outperforms all baselines based on the results shown in Table 4.1. In distance estimation, the robot first identifies an object

patch with `find`, and then leverages its distance sensor by focusing on the detected image patch with `focus_on_patch` and calling `measure_distance` to get the measurement. In weight estimation, after locating the patch with `find`, the robot navigates (`go_to_object`) and proceeds to `pick_up` every object and measure its weight using the force/torque sensor by calling `measure_weight`. At every measurement, the generated programs compare the currently measured value with a previously stored minimum and update it if the current value is lower, finally yielding the minimum distance or weight. In some cases, `find` cannot locate the object patches leading to failures. We believe that this occurs because of the image quality of the simulated objects, which are not as realistic as the ones in real-world images and the OVD model that `find` uses struggles to locate them. In distance estimation, the generated code by the attribute detection API (OVD+GPT-4) incorrectly hardcodes the distance from the object to the robot camera to the distance of the object from the geometric center of the image frame, leading to a very low accuracy.

4.2.6 End-to-End Framework - Robot Demonstration

We integrate our perception-action API into a real robot by implementing a wrapper over a robot-specific API. We deploy the combined end-to-end framework on DJI® RoboMaster™ EP [105], an affordable ground robot with holonomic movement and pick-and-place capabilities. A demonstration of our framework in action is shown in Fig. 4.1. Picking-and-placing an object first requires navigating in front of it with the appropriate orientation (`go_to_object`). To this end, we leverage sensory information to design a control policy for implementing the `go_to_object` function to navigate to a detected object. The control policy is further divided into two sub-policies: i) a visual servoing-based control policy for the lateral movement that aligns the center of the patch of the detected object to the center of the image frame captured by the on-board robot camera (`focus_on_patch`), ii) a control policy for the longitudinal movement that steers the robot towards a proximal position to the object at hand based on an infrared distance sensor. The target distance from the gripper to an object is a pre-

computed functional gap, or in other words an experimentally determined *sweet spot* for picking and placing. Each of these policies is separately handled by a hand-tuned Proportional-Integral-Derivative (PID) controller. The robot is connected to a (*local*) computer via wifi connection and communicates with a (*remote*) computing cluster through a client-server architecture running on an SSH tunnel. To reduce latency due to the computational load of deploying a VLM on the cluster, we only run OVD on the first frame captured by the robot camera, and then track the corresponding position(s) with the Kanade–Lucas–Tomasi (KLT) feature tracker[1] [106].

4.3 Limitations & Future Work

Sensor Integration: Our experiments provide some insights on failure cases and emerging reasoning capabilities of VLMs in attribute detection. We demonstrate the applicability of our action-perception API on a robot in simulation and in the real world. In the future we plan to leverage the compositionality of our API and extend its sensing capabilities by incorporating more sensors (e.g. IMU, temperature sensor) via wrapper functions, supporting the discovery of additional attributes through active perception.

Error Propagation across Model Calls: In Sec. 4.2.5 we showed how the attribute API (VQA+GPT) outperforms calling an OVD or VQA model in isolation. However, if the first call yields an incorrect result, any downstream calls consume erroneous parameters and hence lead to an incorrect final result. In the future we plan to develop mechanisms that leverage additional feedback from the environment to catch such exceptions before errors propagate downstream.

[1] We follow the implementation in https://github.com/ZheyuanXie/KLT-Feature-Tracking.git.

Chapter 5: Proposed Work: Translating Cooking Instructions to Visually Grounded Action Programs using PDDL as an Interface

5.1 Background & Problem Statement

5.1.1 Planning Domain Definition Language (PDDL)

PDDL is a standardized means of encoding classical AI planning problems, developed by McDermott [9]. It clearly separates the representation of the planning problem into a domain file and a problem file. The domain file includes a lifted representation of the state space and action space. The state space defines the types of objects that are admissible in the environment and the allowable states in the form of boolean predicates that consume objects of the given types. The action space consists of STRIPS-style [107] actions with a set of objects as parameters, a set of preconditions that are prerequisite predicate values for action execution, and a set of effects that correspond to the resulting predicate values from successful action execution. The problem file enumerates the specific instances of the objects in the environment, the initial states, and the goal of the plan, or in other words, what state we want the world to be in at plan completion. The initial and goal states are both expressed in the form of predicates.

5.1.2 Problem Statement

Consider a mobile manipulation robot with an action space $\mathcal{A}$. The robot is tasked with executing a natural language instruction that involves completing a task in the kitchen. We adopt a STRIPS [107] action representation, where each action $a_i \in \mathcal{A}$ is described by a set of preconditions $Pre(a_i)$ and a set of postconditions or effects $Post(a_i)$. These sets are defined in the form of boolean predicates that represent the prerequisite states for action execution and the states resulting from successful action execution, respectively. The kitchen is equipped with a set of objects $\mathcal{O}$. We assume access to a robot control API that maps high-level actions to low-level control policies and a camera sensor that streams discrete visual observations of the scene before and after every attempt of performing an action. Our goal is to acquire a finite sequence of actions that correctly executes the given task by acquiring perceptual evidence that all preconditions and postconditions of every action are satisfied.

5.2 NL2PDDL2Prog

5.2.1 System Architecture

To solve this problem, we present NL2PDDL2Prog, a system with an architecture summarized in Fig. 5.1. Specifically, our system:

1. Visually acquires a list of ingredients $\mathcal{I}$, appliances A, utensils $\mathcal{U}$, and containers $\mathcal{C}$ from a scene observation.

2. Generates a PDDL problem file given a natural language instruction, a domain file, and a list of available objects in the environment.

3. Syntactically validates [8] the generated problem file.

4. Uses a classical planner [108] to acquire a feasible plan given the domain file and the generated

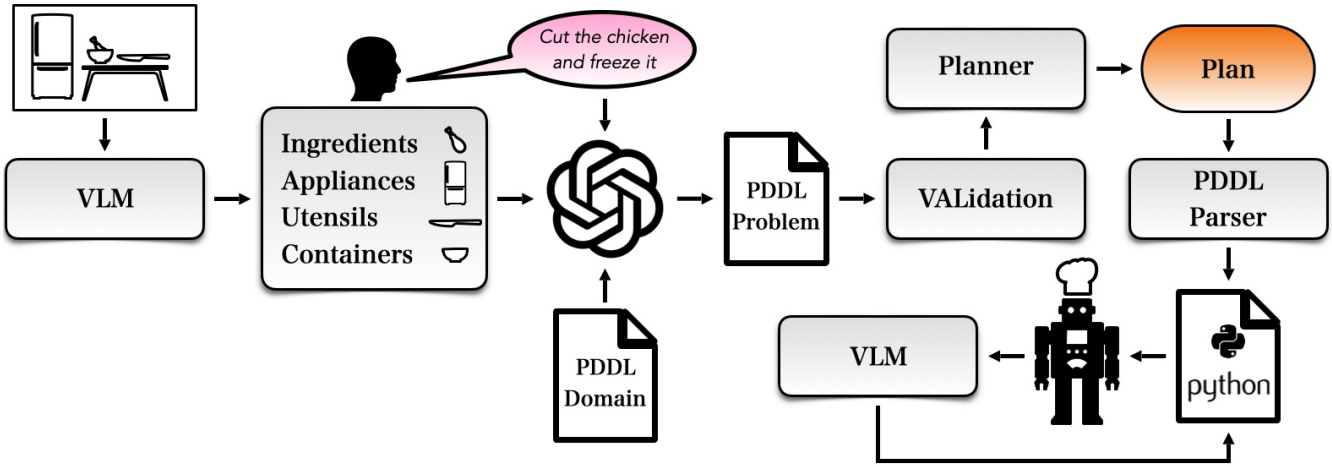


Figure 5.1: Schematic overview of our approach. A scene observation of the current state of the kitchen is passed to a VLM which returns a list with the ingredients, appliances, utensils, and containers, in accordance with the taxonomy of our PDDL domain. The LLM receives this list, the domain file including the admissible action space and state space, and a natural language instruction. It then returns a PDDL problem file which is syntactically validated using VAL [8] and passed along with the domain file to a planner [108]. The planner produces a plan that consists of a sequence of admissible actions to execute the instruction. The output plan is then parsed into a python program that calls a high-level skill API to execute actions and a VLM to ground predicate values corresponding to action preconditions and postconditions.

problem file.

5. Parses the generated plan into a robot-executable python program that visually grounds action preconditions and postconditions at runtime.

NL2PDDL2Prog is algorithmically summarized in Alg. 2. The following sections expand on its individual components.

Algorithm 2 NL2PDDL2Prog

Input: An image of the kitchen img, a PDDL domain file f^D, and a natural language instruction t
Output: A robot-executable action plan in the form of a python program

$\mathcal{O} = \{\mathcal{I}, A, \mathcal{U}, \mathcal{C}\} \leftarrow VQA(img)$ ▷ Object Recognition
$f^{Prob} \leftarrow LLM(f^{Dom}, \mathcal{O}, t)$ ▷ Problem File Generation
$f^{Prob} \leftarrow VAL(f^P)$ ▷ Problem File Validation
$f^{Plan} \leftarrow \mathcal{A}^* \leftarrow FastDownward(f^D, f^P)$ ▷ Planning
$f^{python} \leftarrow Parser(f^{Plan})$ ▷ Parsing
return f^{python}

function $Parser\big(\mathcal{A}^*, Pre(\mathcal{A}^*), Post(\mathcal{A}^*)\big)$

for $a_i^* \in \mathcal{A}^*$ **do**
  for $pre_i \in Pre(a_i^*)$ **do**
    if $VQA(pre_i)$ **then**
      `raisePreconditionError`(pre_i)
    end if
  end for
  `execute`(a_i^*)
  for $post_i \in Post(a_i^*)$ **do**
    if $VQA(post_i)$ **then**
      `raisePostconditionError`$(post_i)$
    end if
  end for
end for

5.2.2 PDDL Domain

The domain file is the foundation of the PDDL formulation, providing a human-readable model of the world. It consists of the types of allowable objects in the environment, the action space in the form of STRIPS-style actions, and the state space in the form of boolean predicates. In our object taxonomy, the set of available objects $\mathcal{O}$ in the kitchen can be further categorized into separate subsets of ingredients $\mathcal{I}$, appliances A, utensils $\mathcal{U}$, and containers $\mathcal{C}$. PDDL allows hierarchical object relationships and property inheritance. This translates to hypernymy or IS-A [109] hierarchical relationships. For example, a kitchen knife IS-A utensil and a utensil IS-An object, hence the knife inherits all properties of its parent object types. Our action space consists of some of the most frequent action verbs in the cooking domain [110] $\mathcal{A}$ ={`pick up`, `place`, `put-in`, `pour`, `cut`, `stir`, `open`, `close`, `turn on`, `turn off`} modeled as transitions from the predicates that must hold before action execution (preconditions) to the predicates that hold after the action (postconditions). The predicates include the location of an object `at ?o - object ?l - location`, a change of state of an ingredient (`sliced` or `stirred`), and `picked-up` denoting that an object has been picked up by the robot. `is-empty` denotes an empty container or appliance and `turned-on`, `is-open` indicates that an appliance has been turned on and is open, respectively. PDDL additionally supports axioms as a means of manually encoding commonsense reasoning rules that are not covered by the action definitions. In our domain, placing or pouring an ingredient i_1 onto another ingredient i_2 that is already in a container, implies that i_1 is also in the same container.

5.2.3 Visual Question Answering for Object Recognition

To acquire the set of available objects $\mathcal{O}$ in the kitchen, we employ a visual question answering (VQA) method using a VLM [101] capable of handling basic visual queries to recognize objects belonging to the admissible categories $\{\mathcal{I}, A, \mathcal{U}, \mathcal{C}\}$. Instead of using an open-vocabulary object detector which would assume a priori knowledge of the specific instances of the objects in the current kitchen

environment [10], we pose a question using the following template *"What are the {object_category} in this image?"*, where *object_category* is one of the following: {`ingredients, appliances, utensils, containers`}. This constrains the domain to a specific set of pre-defined object categories, but enables the system to recognize unseen objects of these categories in novel scenes.

5.2.4 Problem File Generation

The problem file consists of the instances of the available objects in the environment, the predicates that hold at the initial state, and the predicates that must hold at plan completion to signal a successful execution. We assume that the robot is available at the beginning of every plan execution and that the system has full observability of the state, which implies that all appliances are open and the items in them are visible without any significant occlusion. To determine the location of an ingredient or utensil we use the following template *"Where is the {object}?"*. To determine whether a container or appliance is empty we use the template *"Is the {object} empty?"*. Finally, we assume that sliceable ingredients are `not sliced` before plan execution. To generate the PDDL problem file, we prompt an LLM with the domain file, a list of all available objects and their categories, the initial state which corresponds to initial predicate values expressed in an expanded natural language form, and a natural language instruction. The main functionality of the LLM is to derive the initial predicates and the goal conditions based on the domain file and the information extracted through the VQA module. After acquiring the problem file, we use VAL [8] to syntactically validate it before passing it to the planner.

5.2.5 Plan Generation & Parsing

To generate a plan given the domain and problem specifications, we use Fast Downward[1] [108], a heuristic search-based classical planning system widely used in the LLM-PDDL literature [57–61]. The planner generates a finite sequence of admissible actions $\mathcal{A}^*$, if there is one, and produces an output

[1] We use the implementation in https://github.com/aibasel/downward.

file listing these actions and the specific parameter values used for solving the input planning problem. Based on the structure of this file, we synthesize a parser that matches the parameter values to lifted action definitions in the domain file and translates the generated plan into a python program that visually grounds preconditions $Pre(\mathcal{A}^*)$ and effects $Post(\mathcal{A}^*)$. An action is only executed if all preconditions are satisfied. This is achieved by nested conditional statements that query the VQA module with question templates corresponding to each predicate. If any of these preconditions does not hold, an exception is raised informing the user about the specific source of failure. Postconditions are similarly handled with conditional statements after action execution, halting program execution and displaying the error source if any of them is not satisfied. The parser is algorithmically summarized in Alg. 2.

5.3 Proposed Evaluation

5.3.1 Experiment Design

To evaluate NL2PDDL2Prog, we plan to investigate the performance of its individual components and analyze its functionality as an end-to-end system. We will begin by evaluating the VQA module. This is an essential part of the system used to acquire the initial list of objects in the kitchen and validate predicates corresponding to the preconditions and postconditions between and after every action in the generated plan. We plan to generate a set of images from simulated kitchen environments in AI2-THOR [7] and consider a set of images depicting real kitchens taken from the internet. The images will demonstrate different kitchen configurations corresponding to potential states based on the predicates in the domain file. We expect the VQA module to produce more accurate results on real kitchens, due to superior image quality and assuming a significantly higher exposure of the model to real images during pre-training. Additionally, we expect better performance when a smaller set of objects are present in the scene, compared to cluttered settings. For a quantitative evaluation we will compare the performance a set of top-performing open-source VLMs capable of handling basic visual queries, LLaVa [111], BLIP-

2 [101], and the proprietary GPT-4o [112].

Our next step is going to evaluate the ability of the LLM to combine a natural language instruction, a domain file, and a list of detected objects into producing a valid goal specification in the form of a PDDL problem file. We plan to investigate the performance of zero-shot, one-shot, and two-shot prompts. The numbers correspond to the number of example problem files included in the input prompt to bias the answer structure of the model towards producing programs of a similar pattern. We intend to measure the ratio of correct generated goal conditions compared to a ground truth problem file that we manually create based on the goal conditions of a recipe. We consider a goal condition incorrect if the action or any of its parameter values does not exactly match the respective condition in the ground truth file.

Finally, we aim to evaluate NL2PDDL2Prog as a planning system, comparing the output plan to the plan generated by an end-to-end LLM planner. The goal of this evaluation is to investigate whether the use of PDDL as an interface in LLM planning can produce more robust action plans for executing recipes in the kitchen. We will compare our system with an LLM prompted with the available objects, actions, a natural language instruction, and example output plans, and we will measure the executability and correctness of the plans. Executability is measured as the fraction of the generated actions that are admissible in the environment and correctness counts the fraction of generated actions that are also part of a ground truth plan. We formalize these insights into the following questions:

Q1: Is PDDL as an interface better than end-to-end LLM planning?

Q2: How accurate is the predicate grounding module?

Q3: What are the benefits of the visually grounded action programs?

Appendix A: Reading Lists

A.1 LLM Planning

1. Huang, Wenlong, et al. ”Language Models as Zero-Shot Planners: Extracting Actionable Knowledge for Embodied Agents.” International Conference on Machine Learning. PMLR, 2022. [38]

2. Huang, Wenlong, et al. ”Inner Monologue: Embodied Reasoning through Planning with Language Models.” arXiv preprint arXiv:2207.05608 (2022). [39]

3. Wu, Jimmy, et al. ”TidyBot: Personalized Robot Assistance with Large Language Models.” Autonomous Robots 47.8 (2023): 1087-1102. [113]

4. Ahn, Michael, et al. Do As I Can, Not As I Say: Grounding Language in Robotic Affordances. In Proceedings of the Conference on Robot Learning (CoRL), 2023. [37]

5. Singh, Ishika, et al. ”ProgPrompt: Generating Situated Robot Task Plans using Large Language Models.” 2023 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2023. [23]

6. Liang, Jacky, et al. ”Code as Policies: Language Model Programs for Embodied Control.” 2023 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2023. [40]

7. Vemprala, Sai H., et al. ”ChatGPT for Robotics: Design Principles and Model Abilities.” IEEE Access (2024). [41]

8. Valmeekam, Karthik, et al. ”On the Planning Abilities of Large Language Models-A Critical Investigation.” Advances in Neural Information Processing Systems 36 (2023): 75993-76005. [114]

9. Huang, Siyuan, et al. ”Instruct2act: Mapping Multi-Modality Instructions to Robotic Actions with Large Language Model.” arXiv preprint arXiv:2305.11176 (2023). [42]

10. Song, Chan Hee, et al. ”LLM-Planner: Few-Shot Grounded Planning for Embodied Agents with Large Language Models.” Proceedings of the IEEE/CVF International Conference on Computer Vision. 2023. [115]

A.2 Validating LLM Output with Classical Concepts

1. Kress-Gazit, Hadas, Georgios E. Fainekos, and George J. Pappas. ”Temporal-Logic-Based Reactive Mission and Motion Planning.” IEEE transactions on robotics 25.6 (2009): 1370-1381. [64]

2. Liu, Jason Xinyu, et al. ”Grounding Complex Natural Language Commands for Temporal Tasks in Unseen Environments.” Conference on Robot Learning. PMLR, 2023. [70]

3. Pan, Jiayi, Glen Chou, and Dmitry Berenson. ”Data-Efficient Learning of Natural Language to Linear Temporal Logic Translators for Robot Task Specification.” 2023 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2023. [51]

4. Yang, Ziyi, et al. ”Plug in the Safety Chip: Enforcing Constraints for LLM-Driven Robot Agents.” 2024 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2024. [55]

5. Liu, Bo, et al. ”LLM+P: Empowering Large Language Models with Optimal Planning Proficiency.” arXiv preprint arXiv:2304.11477 (2023). [57]

6. Xie, Yaqi, et al. ”Translating Natural Language to Planning Goals with Large-Language Models.” arXiv preprint arXiv:2302.05128 (2023).

7. Guan, Lin, et al. ”Leveraging Pre-trained Large Language Models to Construct and Utilize World Models for Model-based Task Planning.” Advances in Neural Information Processing Systems 36 (2023): 79081-79094.

8. Silver, Tom, et al. ”Generalized Planning in PDDL Domains with Pretrained Large Language Models.” Proceedings of the AAAI Conference on Artificial Intelligence. Vol. 38. No. 18. 2024. [60]

9. Ren, Allen Z., et al. ”Robots That Ask For Help: Uncertainty Alignment for Large Language Model Planners.” Conference on Robot Learning. PMLR, 2023. [116]

10. Chen, Yongchao, et al. ”AutoTAMP: Autoregressive Task and Motion Planning with LLMs as Translators and Checkers.” 2024 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2024. [71]

A.3 Robotic Cooking

1. Beetz, Michael, et al. ”Robotic Roommates Making Pancakes.” 2011 11th IEEE-RAS International Conference on Humanoid Robots. IEEE, 2011. [28]

2. Bollini, Mario, et al. ”Interpreting and Executing Recipes with a Cooking Robot.” Experimental Robotics: The 13th International Symposium on Experimental Robotics. Springer International Publishing, 2013. [1]

3. Malmaud, Jonathan, et al. ”Cooking with semantics.” Proceedings of the ACL 2014 Workshop on Semantic Parsing. 2014. [2]

4. Sakib, Md Sadman, David Paulius, and Yu Sun. ”Approximate Task Tree Retrieval in a Knowledge Network for Robotic Cooking.” IEEE Robotics and Automation Letters 7.4 (2022): 11492-11499. [117]

5. Papadopoulos, Dim P., et al. ”Learning Program Representations for Food Images and Cooking Recipes.” Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2022. [94]

6. Liu, Junjia, et al. ”Robot Cooking with Stir-Fry: Bimanual Non-Prehensile Manipulation of Semi-Fluid Objects.” IEEE Robotics and Automation Letters 7.2 (2022): 5159-5166. [29]

7. Wang, Huaxiaoyue, et al. ”Demo2code: From Summarizing Demonstrations to Synthesizing Code via Extended Chain-of-Thought.” Proceedings of the 37th International Conference on Neural Information Processing Systems. 2023. [43]

8. Wang, Huaxiaoyue, et al. ”MOSAIC: A Modular System for Assistive and Interactive Cooking.” arXiv preprint arXiv:2402.18796 (2024). [34]

9. Shirai, Keisuke, et al. ”Vision-Language Interpreter for Robot Task Planning.” 2024 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2024. [10]

10. Fu, Zipeng, Tony Z. Zhao, and Chelsea Finn. ”Mobile aloha: Learning bimanual mobile manipulation with low-cost whole-body teleoperation.” arXiv preprint arXiv:2401.02117 (2024). [33]

Appendix B: Timeline for Research Completion

The work [4] in Chapter 3 has been published at the 2024 IEEE International Conference on Robotics and Automation (ICRA 2024) and presented in Yokohama, Japan in May 2024. The work [24] in Chapter 4 is under review at the 2025 IEEE International Conference on Robotics and Automation (ICRA 2025) and was accepted and presented as a poster at ICRA@40, the 40th Anniversary of the IEEE International Conference on Robotics and Automation (ICRA) in Rotterdam, Netherlands in September 2024.

We plan to complete the research idea described in Chapter 5, perform real-robot experiments by the end of the year, and submit it to the IEEE Robotics and Automation Letters (RA-L, reviewed on a rolling basis) journal. In terms of future work, we plan to submit two conference papers to the 2025 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2025, deadline: March 2025) and the 2025 Conference on Robot Learning 2025 (CoRL 2025, deadline: June 2025), respectively, and hold the dissertation defense in the Summer of 2025.

Bibliography

[1] Mario Bollini, Stefanie Tellex, Tyler Thompson, Nicholas Roy, and Daniela Rus. Interpreting and executing recipes with a cooking robot. In *Experimental Robotics: The 13th International Symposium on Experimental Robotics*, pages 481–495. Springer, 2013.

[2] Jonathan Malmaud, Earl Wagner, Nancy Chang, and Kevin Murphy. Cooking with semantics. In *Proceedings of the ACL 2014 Workshop on Semantic Parsing*, pages 33–38, 2014.

[3] Yiwei Jiang, Klim Zaporojets, Johannes Deleu, Thomas Demeester, and Chris Develder. Recipe instruction semantics corpus (RISeC): Resolving semantic structure and zero anaphora in recipes. In *Proceedings of the Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the International Joint Conference on Natural Language Processing*, pages 821–826, 2020.

[4] Angelos Mavrogiannis, Christoforos Mavrogiannis, and Yiannis Aloimonos. Cook2ltl: Translating cooking recipes to ltl formulae using large language models, 2023.

[5] Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science*, pages 46–57. ieee, 1977.

[6] Javier Marin, Aritro Biswas, Ferda Ofli, Nicholas Hynes, Amaia Salvador, Yusuf Aytar, Ingmar Weber, and Antonio Torralba. Recipe1m+: A dataset for learning cross-modal embeddings for cooking recipes and food images. *IEEE Trans. Pattern Anal. Mach. Intell.*, 2019.

[7] Eric Kolve, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Matt Deitke, Kiana Ehsani, Daniel Gordon, Yuke Zhu, et al. Ai2-THOR: An interactive 3d environment for visual AI. *arXiv preprint arXiv:1712.05474*, 2017.

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

[9] Drew M McDermott. The 1998 ai planning systems competition. *AI magazine*, 21(2):35–35, 2000.

[10] Keisuke Shirai, Cristian C Beltran-Hernandez, Masashi Hamaya, Atsushi Hashimoto, Shohei Tanaka, Kento Kawaharazuka, Kazutoshi Tanaka, Yoshitaka Ushiku, and Shinsuke Mori. Vision-language interpreter for robot task planning. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2051–2058. IEEE, 2024.

[11] James J Gibson. *The ecological approach to visual perception: classic edition*. Psychology press, 2014.

[12] Devi Parikh and Kristen Grauman. Relative attributes. In *2011 International Conference on Computer Vision*, pages 503–510. IEEE, 2011.

[13] John Firth. A synopsis of linguistic theory, 1930-1955. *Studies in linguistic analysis*, pages 10–32, 1957.

[14] Xiaohan Zhang, Saeid Amiri, Jivko Sinapov, Jesse Thomason, Peter Stone, and Shiqi Zhang. Multimodal embodied attribute learning by robots for object-centric action policies. *Autonomous Robots*, pages 1–24, 2023.

[15] Khoi Pham, Kushal Kafle, Zhe Lin, Zhihong Ding, Scott Cohen, Quan Tran, and Abhinav Shrivastava. Learning to predict visual attributes in the wild. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 13018–13028, 2021.

[16] Maria A Bravo, Sudhanshu Mittal, Simon Ging, and Thomas Brox. Open-vocabulary attribute detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7041–7050, 2023.

[17] Keyan Chen, Xiaolong Jiang, Yao Hu, Xu Tang, Yan Gao, Jianqi Chen, and Weidi Xie. Ovarnet: Towards open-vocabulary object attribute recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 23518–23527, 2023.

[18] Xiaoyuan Guo, Kezhen Chen, Jinmeng Rao, Yawen Zhang, Baochen Sun, and Jie Yang. Lowa: Localize objects in the wild with attributes. In *R0-FoMo: Robustness of Few-shot and Zero-shot Learning in Large Foundation Models*, 2023.

[19] Gyan Tatiya, Jonathan Francis, Ho-Hsiang Wu, Yonatan Bisk, and Jivko Sinapov. Mosaic: Learning unified multi-sensory object property representations for robot perception. *arXiv preprint arXiv:2309.08508*, 2023.

[20] Lorenzo Bianchi, Fabio Carrara, Nicola Messina, Claudio Gennaro, and Fabrizio Falchi. The devil is in the fine-grained details: Evaluating open-vocabulary object detectors for fine-grained understanding. *arXiv preprint arXiv:2311.17518*, 2023.

[21] Olga Russakovsky and Li Fei-Fei. Attribute learning in large-scale datasets. In *Trends and Topics in Computer Vision: ECCV 2010 Workshops, Heraklion, Crete, Greece, September 10-11, 2010, Revised Selected Papers, Part I 11*, pages 1–14. Springer, 2012.

[22] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021.

[23] Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. Progprompt: Generating situated robot task plans using large language models. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11523–11530. IEEE, 2023.

[24] Angelos Mavrogiannis, Dehao Yuan, and Yiannis Aloimonos. Discovering object attributes by prompting large language models with perception-action apis, 2024. URL https://arxiv.org/abs/2409.15505.

[25] The "Poetics" of Everyday Life: Grounding Resources and Mechanisms for Artificial Agents. https://cordis.europa.eu/project/id/215843. Accessed: 2023-09-28.

[26] Yezhou Yang, Yi Li, Cornelia Fermuller, and Yiannis Aloimonos. Robot learning manipulation action plans by" watching" unconstrained videos from the world wide web. In *Proceedings of the AAAI conference on artificial intelligence*, volume 29, 2015.

[27] Katerina Pastra and Yiannis Aloimonos. The minimalist grammar of action. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 367(1585):103–117, 2012.

[28] Michael Beetz, Ulrich Klank, Ingo Kresse, Alexis Maldonado, Lorenz Mösenlechner, Dejan Pangercic, Thomas Rühr, and Moritz Tenorth. Robotic roommates making pancakes. In *2011 11th IEEE-RAS International Conference on Humanoid Robots*, pages 529–536. IEEE, 2011.

[29] Junjia Liu, Yiting Chen, Zhipeng Dong, Shixiong Wang, Sylvain Calinon, Miao Li, and Fei Chen. Robot cooking with stir-fry: Bimanual non-prehensile manipulation of semi-fluid objects. *IEEE Robotics and Automation Letters*, 7(2):5159–5166, 2022.

[30] Donghun Noh, Hyunwoo Nam, Kyle Gillespie, Yeting Liu, and Dennis Hong. Yori: Autonomous cooking system utilizing a modular robotic kitchen and a dual-arm proprioceptive manipulator. *arXiv preprint arXiv:2405.11094*, 2024.

[31] Moley Robotics. Moley kitchen. URL https://www.moley.com/moley-kitchen/. Accessed: 2023-05-29.

[32] Haochen Shi, Huazhe Xu, Samuel Clarke, Yunzhu Li, and Jiajun Wu. Robocook: Long-horizon elasto-plastic object manipulation with diverse tools. In *7th Annual Conference on Robot Learning*, 2023. URL https://openreview.net/forum?id=69y5fzvaAT.

[33] Zipeng Fu, Tony Z. Zhao, and Chelsea Finn. Mobile aloha: Learning bimanual mobile manipulation with low-cost whole-body teleoperation. In *arXiv*, 2024.

[34] Huaxiaoyue Wang, Kushal Kedia, Juntao Ren, Rahma Abdullah, Atiksh Bhardwaj, Angela Chao, Kelly Y Chen, Nathaniel Chin, Prithwish Dan, Xinyi Fan, et al. Mosaic: A modular system for assistive and interactive cooking. *arXiv preprint arXiv:2402.18796*, 2024.

[35] Zhe Huang, John Pohovey, Ananya Yammanuru, and Katherine Driggs-Campbell. Lit: Large language model driven intention tracking for proactive human-robot collaboration–a robot sous-chef application. *arXiv preprint arXiv:2406.13787*, 2024.

[36] David Paulius, Yongqiang Huang, Roger Milton, William D Buchanan, Jeanine Sam, and Yu Sun. Functional object-oriented network for manipulation learning. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2655–2662. IEEE, 2016.

[37] Brian Ichter, Anthony Brohan, Yevgen Chebotar, Chelsea Finn, Karol Hausman, Alexander Herzog, Daniel Ho, Julian Ibarz, Alex Irpan, Eric Jang, Ryan Julian, Dmitry Kalashnikov, Sergey Levine, Yao Lu, Carolina Parada, Kanishka Rao, Pierre Sermanet, Alexander T Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Mengyuan Yan, Noah Brown, Michael Ahn, Omar Cortes, Nicolas Sievers, Clayton Tan, Sichun Xu, Diego Reyes, Jarek Rettinghouse, Jornell Quiambao, Peter Pastor, Linda Luu, Kuang-Huei Lee, Yuheng Kuang, Sally Jesmonth, Nikhil J. Joshi, Kyle Jeffrey, Rosario Jauregui Ruano, Jasmine Hsu, Keerthana Gopalakrishnan, Byron David, Andy Zeng, and Chuyuan Kelly Fu. Do as i can, not as i say: Grounding language in robotic affordances. In *Proceedings of the Conference on Robot Learning (CoRL)*, volume 205, pages 287–318, 2023.

[38] Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International Conference on Machine Learning*, pages 9118–9147. PMLR, 2022.

[39] Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, et al. Inner monologue: Embodied reasoning through planning with language models. *arXiv preprint arXiv:2207.05608*, 2022.

[40] Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9493–9500. IEEE, 2023.

[41] Sai H Vemprala, Rogerio Bonatti, Arthur Bucker, and Ashish Kapoor. Chatgpt for robotics: Design principles and model abilities. *IEEE Access*, 2024.

[42] Siyuan Huang, Zhengkai Jiang, Hao Dong, Yu Qiao, Peng Gao, and Hongsheng Li. Instruct2act: Mapping multi-modality instructions to robotic actions with large language model. *arXiv preprint arXiv:2305.11176*, 2023.

[43] Huaxiaoyue Wang, Gonzalo Gonzalez-Pumariega, Yash Sharma, and Sanjiban Choudhury. Demo2code: From summarizing demonstrations to synthesizing code via extended chain-of-thought, 2023.

[44] Xiuye Gu, Tsung-Yi Lin, Weicheng Kuo, and Yin Cui. Open-vocabulary object detection via vision and language knowledge distillation. *International Conference on Learning Representations*, 2022.

[45] Aishwarya Kamath, Mannat Singh, Yann LeCun, Gabriel Synnaeve, Ishan Misra, and Nicolas Carion. Mdetr-modulated detection for end-to-end multi-modal understanding. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1780–1790, 2021.

[46] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C Berg, Wan-Yen Lo, et al. Segment anything. *arXiv preprint arXiv:2304.02643*, 2023.

[47] Harel Biggie, Ajay Narasimha Mopidevi, Dusty Woods, and Chris Heckman. Tell me where to go: A composable framework for context-aware embodied robot navigation. In *7th Annual Conference on Robot Learning*, 2023.

[48] Andy Zeng, Adrian Wong, Stefan Welker, Krzysztof Choromanski, Federico Tombari, Aveek Purohit, Michael Ryoo, Vikas Sindhwani, Johnny Lee, Vincent Vanhoucke, et al. Socratic models: Composing zero-shot multimodal reasoning with language. *arXiv preprint arXiv:2204.00598*, 2022.

[49] Karthik Valmeekam, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. Large language models still can't plan (a benchmark for llms on planning and reasoning about change). In *NeurIPS 2022 Foundation Models for Decision Making Workshop*, 2022.

[50] Jason Xinyu Liu, Ziyi Yang, Ifrah Idrees, Sam Liang, Benjamin Schornstein, Stefanie Tellex, and Ankit Shah. Grounding complex natural language commands for temporal tasks in unseen environments. In Jie Tan, Marc Toussaint, and Kourosh Darvish, editors, *Proceedings of The 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pages 1084–1110. PMLR, 06–09 Nov 2023. URL https://proceedings.mlr.press/v229/liu23d.html.

[51] Jiayi Pan, Glen Chou, and Dmitry Berenson. Data-efficient learning of natural language to linear temporal logic translators for robot task specification. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11554–11561. IEEE, 2023.

[52] Matthias Cosler, Christopher Hahn, Daniel Mendoza, Frederik Schmitt, and Caroline Trippel. nl2spec: interactively translating unstructured natural language to temporal logics with large language models. In *International Conference on Computer Aided Verification*, pages 383–396. Springer, 2023.

[53] Sara Mohammadinejad, Jesse Thomason, and Jyotirmoy V Deshmukh. Interactive learning from natural language and demonstrations using signal temporal logic. *arXiv preprint arXiv:2207.00627*, 2022.

[54] Yongchao Chen, Rujul Gandhi, Yang Zhang, and Chuchu Fan. Nl2tl: Transforming natural languages to temporal logics using large language models. *arXiv preprint arXiv:2305.07766*, 2023.

[55] Ziyi Yang, Shreyas S Raman, Ankit Shah, and Stefanie Tellex. Plug in the safety chip: Enforcing constraints for llm-driven robot agents. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 14435–14442. IEEE, 2024.

[56] Benedict Quartey, Eric Rosen, Stefanie Tellex, and George Konidaris. Verifiably following complex robot instructions with foundation models, 2024. URL https://arxiv.org/abs/2402.11498.

[57] 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, 2023.

[58] Yaqi Xie, Chen Yu, Tongyao Zhu, Jinbin Bai, Ze Gong, and Harold Soh. Translating natural language to planning goals with large-language models. *arXiv preprint arXiv:2302.05128*, 2023.

[59] Lin Guan, Karthik Valmeekam, Sarath Sreedharan, and Subbarao Kambhampati. Leveraging pretrained large language models to construct and utilize world models for model-based task planning. *Advances in Neural Information Processing Systems*, 36:79081–79094, 2023.

[60] Tom Silver, Soham Dan, Kavitha Srinivas, Joshua B Tenenbaum, Leslie Kaelbling, and Michael Katz. Generalized planning in pddl domains with pretrained large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 20256–20264, 2024.

[61] Ishika Singh, David Traum, and Jesse Thomason. Twostep: Multi-agent task planning using classical planners and large language models. *arXiv preprint arXiv:2403.17246*, 2024.

[62] Zhehua Zhou, Jiayang Song, Kunpeng Yao, Zhan Shu, and Lei Ma. Isr-llm: Iterative self-refined large language model for long-horizon sequential task planning. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2081–2088. IEEE, 2024.

[63] Georgios Fainekos, Hadas Kress-Gazit, and George Pappas. Temporal logic motion planning for mobile robots. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2020–2025, 2005.

[64] Hadas Kress-Gazit, Georgios E Fainekos, and George J Pappas. Temporal-logic-based reactive mission and motion planning. *IEEE transactions on robotics*, 25(6):1370–1381, 2009.

[65] Stephen L Smith, Jana Tumova, Calin Belta, and Daniela Rus. Optimal path planning for surveillance with temporal-logic constraints. *The International Journal of Robotics Research*, 30(14):1695–1708, 2011.

[66] Nakul Gopalan, Dilip Arumugam, Lawson LS Wong, and Stefanie Tellex. Sequence-to-sequence language grounding of non-markovian task specifications. In *Robotics: Science and Systems*, volume 2018, 2018.

[67] Roma Patel, Ellie Pavlick, and Stefanie Tellex. Grounding language to non-markovian tasks with no supervision of task specifications. In *Robotics: Science and Systems*, volume 2020, 2020.

[68] Christopher Wang, Candace Ross, Yen-Ling Kuo, Boris Katz, and Andrei Barbu. Learning a natural-language to ltl executable semantic parser for grounded robotics. In *Conference on Robot Learning*, pages 1706–1718, 2021.

[69] Matthew Berg, Deniz Bayazit, Rebecca Mathew, Ariel Rotter-Aboyoun, Ellie Pavlick, and Stefanie Tellex. Grounding language to landmarks in arbitrary outdoor environments. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 208–215. IEEE, 2020.

[70] Jason Xinyu Liu, Ziyi Yang, Ifrah Idrees, Sam Liang, Benjamin Schornstein, Stefanie Tellex, and Ankit Shah. Lang2ltl: Translating natural language commands to temporal robot task specification. *arXiv preprint arXiv:2302.11649*, 2023.

[71] Yongchao Chen, Jacob Arkin, Charles Dawson, Yang Zhang, Nicholas Roy, and Chuchu Fan. Autotamp: Autoregressive task and motion planning with llms as translators and checkers. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6695–6702. IEEE, 2024.

[72] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[73] Jiatao Gu, Zhengdong Lu, Hang Li, and Victor OK Li. Incorporating copying mechanism in sequence-to-sequence learning. *arXiv preprint arXiv:1603.06393*, 2016.

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

[75] Eric Hsiung, Hiloni Mehta, Junchi Chu, Xinyu Liu, Roma Patel, Stefanie Tellex, and George Konidaris. Generalizing to new domains by mapping natural language to lifted ltl. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 3624–3630. IEEE, 2022.

[76] Yu-qian Jiang, Shi-qi Zhang, Piyush Khandelwal, and Peter Stone. Task planning in robotics: an empirical comparison of pddl-and asp-based systems. *Frontiers of Information Technology & Electronic Engineering*, 20:363–373, 2019.

[77] Toki Migimatsu and Jeannette Bohg. Grounding predicates through actions. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 3498–3504. IEEE, 2022.

[78] Vittorio Ferrari and Andrew Zisserman. Learning visual attributes. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems*, volume 20. Curran Associates, Inc., 2007.

[79] Ali Farhadi, Ian Endres, Derek Hoiem, and David Forsyth. Describing objects by their attributes. In *2009 IEEE conference on computer vision and pattern recognition*, pages 1778–1785. IEEE, 2009.

[80] Christoph H Lampert, Hannes Nickisch, and Stefan Harmeling. Learning to detect unseen object classes by between-class attribute transfer. In *2009 IEEE conference on computer vision and pattern recognition*, pages 951–958. IEEE, 2009.

[81] Steven Chen and Kristen Grauman. Compare and contrast: Learning prominent visual differences. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1267–1276, 2018.

[82] Yi Wang, Jiafei Duan, Dieter Fox, and Siddhartha Srinivasa. NEWTON: Are large language models capable of physical reasoning? In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 9743–9758, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-emnlp.652. URL https://aclanthology.org/2023.findings-emnlp.652.

[83] Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Neural module networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 39–48, 2016.

[84] Ronghang Hu, Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Kate Saenko. Learning to reason: End-to-end module networks for visual question answering. In *Proceedings of the IEEE international conference on computer vision*, pages 804–813, 2017.

[85] Justin Johnson, Bharath Hariharan, Laurens Van Der Maaten, Judy Hoffman, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. Inferring and executing programs for visual reasoning. In *Proceedings of the IEEE international conference on computer vision*, pages 2989–2998, 2017.

[86] Tanmay Gupta and Aniruddha Kembhavi. Visual programming: Compositional visual reasoning without training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14953–14962, 2023.

[87] Dídac Surís, Sachit Menon, and Carl Vondrick. Vipergpt: Visual inference via python execution for reasoning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 11888–11898, 2023.

[88] Sanjay Subramanian, Medhini Narasimhan, Kushal Khangaonkar, Kevin Yang, Arsha Nagrani, Cordelia Schmid, Andy Zeng, Trevor Darrell, and Dan Klein. Modular visual question answering via code generation. *arXiv preprint arXiv:2306.05392*, 2023.

[89] Aleksandar Stanić, Sergi Caelles, and Michael Tschannen. Towards truly zero-shot compositional visual reasoning with llms as programmers. *arXiv preprint arXiv:2401.01974*, 2024.

[90] Yoshua Bengio, Réjean Ducharme, and Pascal Vincent. A neural probabilistic language model. *Advances in neural information processing systems*, 13, 2000.

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

[92] Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, Abu Awal Md Shoeb, Abubakar Abid, Adam Fisch, Adam R Brown, Adam Santoro, Aditya Gupta, Adrià Garriga-Alonso, et al. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *arXiv preprint arXiv:2206.04615*, 2022.

[93] Pontus Stenetorp, Sampo Pyysalo, Goran Topić, Tomoko Ohta, Sophia Ananiadou, and Jun'ichi Tsujii. brat: a web-based tool for NLP-assisted text annotation. In *Proceedings of the Demonstrations Session at EACL 2012*, Avignon, France, April 2012. Association for Computational Linguistics.

[94] Dim P Papadopoulos, Enrique Mora, Nadiia Chepurko, Kuan Wei Huang, Ferda Ofli, and Antonio Torralba. Learning program representations for food images and cooking recipes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16559–16569, 2022.

[95] Claudio Menghi, Christos Tsigkanos, Patrizio Pelliccione, Carlo Ghezzi, and Thorsten Berger. Specification patterns for robotic missions. *IEEE Transactions on Software Engineering*, 47(10): 2208–2224, 2019.

[96] Roma Patel, Roma Pavlick, and Stefanie Tellex. Learning to ground language to temporal logical form. In *NAACL*, 2019.

[97] Berk Calli, Arjun Singh, Aaron Walsman, Siddhartha Srinivasa, Pieter Abbeel, and Aaron M. Dollar. The ycb object and model set: Towards common benchmarks for manipulation research. In *2015 International Conference on Advanced Robotics (ICAR)*, pages 510–517, 2015.

[98] Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. Pddl-the planning domain definition language. 1998.

[99] Xiangyu Zhao, Yicheng Chen, Shilin Xu, Xiangtai Li, Xinjiang Wang, Yining Li, and Haian Huang. An open and comprehensive pipeline for unified object grounding and detection. *arXiv preprint arXiv:2401.02361*, 2024.

[100] Liunian Harold Li, Pengchuan Zhang, Haotian Zhang, Jianwei Yang, Chunyuan Li, Yiwu Zhong, Lijuan Wang, Lu Yuan, Lei Zhang, Jenq-Neng Hwang, et al. Grounded language-image pre-training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10965–10975, 2022.

[101] Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. *arXiv preprint arXiv:2301.12597*, 2023.

[102] Iuliia Kotseruba, Calden Wloka, Amir Rasouli, and John K. Tsotsos. Do Saliency Models Detect Odd-One-Out Targets? New Datasets and Evaluations. In *British Machine Vision Conference (BMVC)*, 2019.

[103] Sahar Kazemzadeh, Vicente Ordonez, Mark Matten, and Tamara Berg. Referitgame: Referring to objects in photographs of natural scenes. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 787–798, 2014.

[104] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. *arXiv preprint arXiv:2307.15818*, 2023.

[105] DJI RoboMaster EP. https://www.dji.com/robomaster-ep. Accessed: 2024-02-01.

[106] Carlo Tomasi and Takeo Kanade. Detection and tracking of point. *Int J Comput Vis*, 9(137-154): 3, 1991.

[107] Richard E Fikes and Nils J Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3-4):189–208, 1971.

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

[109] Dan Jurafsky. *Speech & language processing*. Pearson Education India, 2000.

[110] Daniel Nyga and Michael Beetz. Everything robots always wanted to know about housework (but were afraid to ask). In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 243–250. IEEE, 2012.

[111] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. *Advances in neural information processing systems*, 36, 2024.

[112] OpenAI. Hello gpt-4o. URL https://openai.com/index/hello-gpt-4o. Accessed: 2024-09-12.

[113] Jimmy Wu, Rika Antonova, Adam Kan, Marion Lepert, Andy Zeng, Shuran Song, Jeannette Bohg, Szymon Rusinkiewicz, and Thomas Funkhouser. Tidybot: Personalized robot assistance with large language models. *Autonomous Robots*, 47(8):1087–1102, 2023.

[114] Karthik Valmeekam, Matthew Marquez, Sarath Sreedharan, and Subbarao Kambhampati. On the planning abilities of large language models-a critical investigation. *Advances in Neural Information Processing Systems*, 36:75993–76005, 2023.

[115] Chan Hee Song, Jiaman Wu, Clayton Washington, Brian M Sadler, Wei-Lun Chao, and Yu Su. Llm-planner: Few-shot grounded planning for embodied agents with large language models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2998–3009, 2023.

[116] Allen Z Ren, Anushri Dixit, Alexandra Bodrova, Sumeet Singh, Stephen Tu, Noah Brown, Peng Xu, Leila Takayama, Fei Xia, Jake Varley, et al. Robots that ask for help: Uncertainty alignment for large language model planners. In *Conference on Robot Learning*, pages 661–682. PMLR, 2023.

[117] Md Sadman Sakib, David Paulius, and Yu Sun. Approximate task tree retrieval in a knowledge network for robotic cooking. *IEEE Robotics and Automation Letters*, 7(4):11492–11499, 2022.