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

Angelos Mavrogiannis[1], Christoforos Mavrogiannis[2], and Yiannis Aloimonos[1]

***Abstract*— Cooking recipes are especially challenging to translate to robot plans as they feature rich linguistic complexity, temporally-extended interconnected tasks, and an almost infinite space of possible actions. Our key insight is that combining a source of background cooking domain knowledge with a formalism capable of handling the temporal richness of cooking recipes could enable the extraction of unambiguous, robot-executable plans. In this work, we use Linear Temporal Logic (LTL) as a formal language expressible enough to model the temporal nature of cooking recipes. Leveraging pre-trained Large Language Models (LLMs), we present a system that translates instruction steps from an arbitrary cooking recipe found on the internet to a series of LTL formulae, grounding high-level cooking actions to a set of primitive actions that are executable by a manipulator in a kitchen environment. Our approach makes use of a caching scheme, dynamically building a queryable action library at runtime, significantly decreasing LLM API calls (-51%), latency (-59%) and cost (-42%) compared to a baseline that queries the LLM for every newly encountered action at runtime. We demonstrate the transferability of our system in a realistic simulation platform through showcasing a set of simple cooking tasks.**

I. INTRODUCTION

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, or performing a delicate assembly task.

In this paper, we focus on translating cooking recipes into executable robot plans. Cooking is one of the most common household activities and poses a unique set of challenges to robots [5]. 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 [25], such as omitting arguments that are easily inferred from context (the known "Zero Anaphora" problem [18]; see Fig. 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.

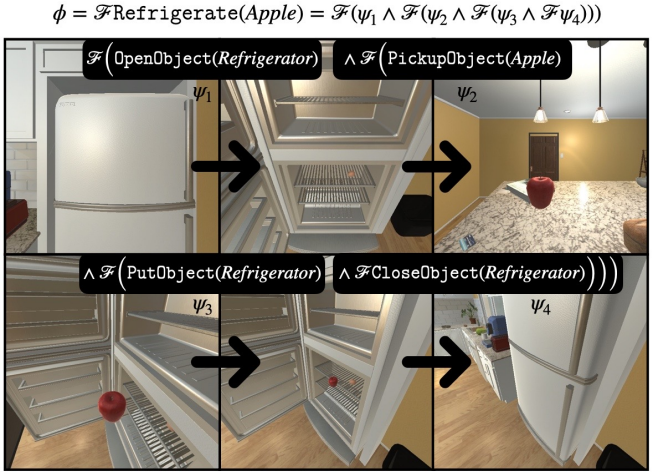


Fig. 1: **Cook2LTL in AI2-THOR** [19]: The robot is given the instruction *Refrigerate the apple*. Cook2LTL produces an initial LTL formula ϕ (*top left*); then it queries an LLM to retrieve the low-level admissible primitives for executing the action; finally it generates a formula consisting of 4 atomic propositions $(\psi_1, \psi_2, \psi_3, \psi_4)$ that provide the required task specification and yield these consecutive scenes.

Motivated by these observations, our key insight is that combining a source of background cooking domain knowledge with a formalism capable of handling the temporal richness of cooking recipes could enable the extraction of unambiguous, robot-executable plans. Our **main contribution** is 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 LTL formulae (See Fig. 1). These plans are then suitable for use in downstream robotic tasks. We cache the action reduction policy, incrementally building a queryable action library and limiting 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 build and evaluate our method based on a subset of recipes from the Recipe1M+ corpus [26], and demonstrate its transferability to an embodied robotic platform through experiments in a simulated kitchen in AI2-THOR [19].

[1]Department of Computer Science, University of Maryland, College Park, 8125 Paint Branch Dr, College Park, MD 20742, USA. angelosm@cs.umd.edu, jyaloimo@cs.umd.edu

[2]Department of Robotics, University of Michigan, Ann Arbor, MI, 48105. cmavro@umich.edu.

Our code can be found at this link. A video with an example simulation rollout can be found at this link.

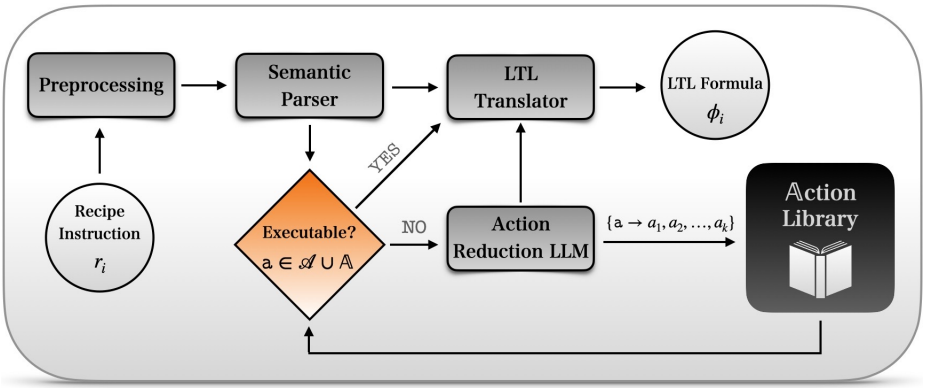


Fig. 2: **Cook2LTL System:** The input instruction r_i is first preprocessed and then passed to the semantic parser, which extracts meaningful chunks corresponding to the categories $\mathcal{C}$ and constructs a function representation $\mathtt{a}$ for each detected action. If $\mathtt{a}$ is part of the action library $\mathbb{A}$, then the LTL translator infers the final LTL formula ϕ. Otherwise, the action is reduced to a lower-level of admissible actions $\{a_1, a_2, \ldots a_k\}$ from $\mathcal{A}$, saving the reduction policy to $\mathbb{A}$ for future use. The LTL translator yields the final LTL formulae from the derived actions.

II. RELATED WORK

Robotic Cooking: Cooking has been an important means of studying action understanding [1, 3, 32, 45]. The EU project POETICON [1] viewed cognitive systems as a set of languages {natural, vision, motor} and integrated these languages towards understanding cooking actions. Following this point of view, Yang et al. [45] processed YouTube videos using Convolutional Neural Networks (CNNs) and a grammatical approach [32] to produce parse trees that could be used for generating cooking actions. A few works have built end-to-end cooking systems that implement textual recipes on real robots [3, 5], but are restricted to completing a roughly specific task (e.g., baking [5], and making pancakes [3]) and hence can only deal with a limited subset of recipes. On the other hand, more versatile commercial solutions (e.g., the Moley kitchen [29]) are expensive and to the best of our knowledge cannot handle unseen recipes in real time.

Although our approach has not been applied on a real-world hardware platform, our AI2-THOR simulation [19] in Sec. V-B demonstrates its transferability to a real robot while allowing the system to adapt to new recipes.

LLM planning: Several works have grounded high-level actions to a well-defined set of actions for task planning using textual LLM- [15–17, 21, 24, 37, 41] or multimodal LLM-based [10, 44, 46] approaches. Our interest lies in the former category given the unimodal nature of our text-to-robot action approach. These textual LLM-based works have shown great performance but come with certain limitations. For instance, the framework of Ichter et al. [17] cannot handle open-vocabulary or combinatorial tasks, the one by Huang et al. [16] might produce action plans including items that are not present in the current environment, and the model of Huang et al. [15] does not guarantee that the returned actions are admissible in the current context. Some of these works [24, 37, 41, 43] have leveraged programming language structures as an expressive tool for efficiently representing a rich set of task procedures in the LLM prompts. 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 to generate the lower-level task plan.

In our work, we adapt the methodology proposed by Singh et al. [37], 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, restricting the output plan to the constrained set of primitive actions and objects available in the current environment.

Natural Language to LTL: LTL was proposed as a method of formal verification for computer programs [35] but has been extensively used in robotics [11, 20, 38] as a paradigm that provides 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 to LTL formulae. Most of the proposed approaches try to address the main bottleneck which is the cost of obtaining annotations of natural language with their equivalent LTL logical forms. Gopalan et al. [12] orchestrate a data collection and augmentation pipeline to build a synthetic domain and translate natural language to LTL formulae using Seq2Seq models [2]. Alternatively, Patel et al. [34], Wang et al. [42] 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. [4], Liu et al. [23] ground referring expressions to a known set of atomic propositions and translate to LTL formulae using Seq2Seq models [13] and LLMs [6], respectively. Similarly, Chen et al. [8], Pan et al. [30] are using the paraphrasing abilities of LLMs to generate synthetic datasets tackling the scarcity of labeled LTL data.

Our approach is more similar to the work of Chen et al.

[8] and Hsiung et al. [14], 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, while we deal with unstructured free-form recipe text scraped from the internet. Moreover, most of these works assuming an embodied agent executing actions have mainly been applied to navigation and simple pick-and-place tasks or combinations of these, while our web-scraped cooking recipe corpus offers a richer and more diverse action space.

III. PRELIMINARIES

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

A. Linear Temporal Logic

LTL is a temporal logic that was developed for formal verification of computer programs through model checking [35]. It is suitable for expressing task specifications and verifying system performance in safety-critical applications. These task specifications are expressed through the use of the following grammar:

$$\phi ::= p \,|\, \neg p \,|\, \phi_1 \wedge \phi_2 \,|\, \phi_1 \vee \phi_2 \,|\, \mathcal{G}\,\phi \,|\, \mathcal{F}\,\phi \,|\, \phi_1\,\mathcal{U}\,\phi_2 \qquad (1)$$

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 $\mathcal{G}\,\phi$ which denotes that ϕ holds globally, $\mathcal{F}\,\phi$ which denotes that ϕ must eventually hold, and $\phi_1\,\mathcal{U}\,\phi_2$ indicating 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 cooking tasks.

B. 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:

$$p(W) = p(w_1, w_2, \ldots, w_n) = \prod_{i=1}^{n} p(w_i | w_1, \ldots, w_i - 1) \quad (2)$$

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 [9] and GPT-3 [6] are pre-trained on large-scale internet corpora and have dominated across a series of downstream natural language processing (NLP) tasks [39]. 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.

IV. TRANSLATING COOKING RECIPES TO LTL FORMULAE

A. 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 categories as its parameters:

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

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.

B. System Architecture

To solve this problem, we propose Cook2LTL, the system architecture summarized in Fig. 2. Given an instruction r_i and a set of actions $\mathcal{A}$, Cook2LTL:

1) Semantically parses r_i into a function representation `a` for every detected high-level action.
2) Reduces each high-level action $\texttt{a} \notin \mathcal{A}$ 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.

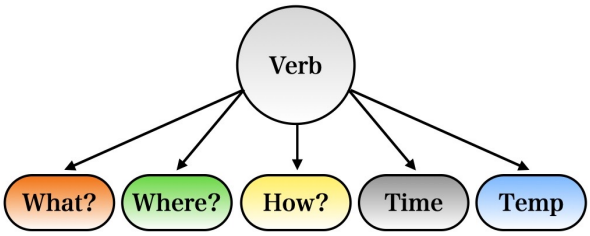
C. Semantic Parsing and Data Annotation

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 a combination of part-of-speech 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.

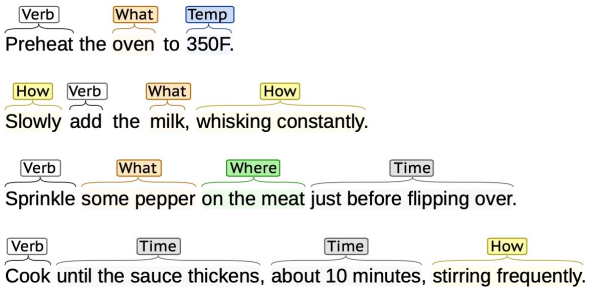
Algorithm 1 Cook2LTL

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



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



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

Fig. 3: We annotate Recipe1M+ [26] 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 these categories.

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+ [26]. Specifically, we consider a subset of 100 recipes from Recipe1M+, leading to 1000 recipe instruction steps and use brat [40] 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 [31].

Fig. 3 shows these categories and a set of example recipe steps taken from Recipe1M+ [26]. `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 to. `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 (Fahrenheit or Celsius, 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.

D. Reduction to Primitive Actions

Some of the function representations captured in the previous step contain high-level actions that might not be supported and 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 [37] 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 [37] and in our example in Fig. 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 changing the primitive actions in the pythonic import.

Action Library: Extending the work of [37], every time that 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 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 $\mathtt{a}$ in the pythonic import part of the prompt, allowing the model to invoke it when generating future policies (e.g. the action `boil` in 4). The key benefit comes from avoiding to

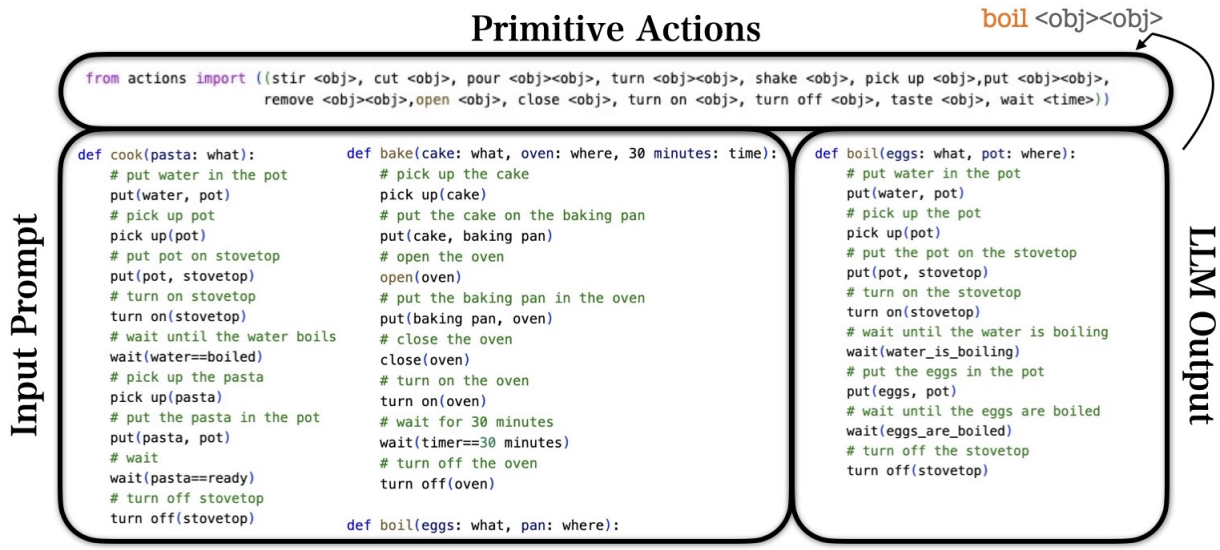


Fig. 4: Inspired by ProgPrompt [37], Cook2LTL uses an LLM prompting scheme to reduce a high-level cooking action (e.g. `boil eggs`) to a series of primitive manipulation actions. The prompt consists of an import statement of the primitive action set and example function definitions of similar cooking 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 prompt, and the policy is now considered given to the system.

continuously query an LLM for action reduction, 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 such as the OpenAI API.

E. 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

$$F(l_1 \wedge \mathcal{F}(l_2 \wedge \ldots \mathcal{F}l_n))) \tag{3}$$

defined in [28] and used in [23, 33, 34] 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, and 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. 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. 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 $\mathcal{A}$:

$$\phi = F(\mathtt{a}_1 \wedge \mathcal{F}(\mathtt{a}_2 \wedge \ldots \mathcal{F}\mathtt{a}_n))) = F(\psi_1 \wedge \mathcal{F}(\psi_2 \wedge \ldots \mathcal{F}\psi_n))) \tag{4}$$

where:

$$\begin{cases} \psi_i = a_i & , a_i \in \mathcal{A} \text{ ,or} \\ \psi_i = f(a_1, a_2, \ldots, a_k, \mathbb{O}) & , \mathbb{O} = \{\mathcal{F}, \wedge, \vee, \neg, \mathcal{U}\} \end{cases} \tag{5}$$

V. EVALUATION

A. Ablation Study

To investigate the performance of Cook2LTL, we conduct an ablation study comparing variants of Cook2LTL. 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, and 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

Fig. 5: Tasks we tested Cook2LTL on 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 I: Performance of Cook2LTL against baselines across 50 Recipe1M+ [26] recipes (10 runs per recipe).

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.

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)* 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*.

B. Results & Discussion

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

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+𝔸) outperforms the baseline (AR*) and primitive action-focused system version (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 of our system, and cost by 42% and 31%, respectively.

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

C. Demonstration in AI2-THOR

We demonstrate the performance of Cook2LTL in a simulated AI2-THOR [19] kitchen environment (See Fig. 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 primitive action concept 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. 5 contains screenshots from our experiments. We run 5 set 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+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 II).

VI. Limitations & Future Work

System: We annotated a small part of the Recipe1M+ dataset [26] 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 [7]

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 the Planning Domain Definition Language (PDDL) [27] which incorporates action preconditions and postconditions in the problem setting and has recently been explored with LLMs [22, 36].

REFERENCES

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

[2] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[3] M. Beetz, U. Klank, I. Kresse, A. Maldonado, L. Mösenlechner, D. Pangercic, T. Rühr, and M. Tenorth. Robotic roommates making pancakes. In *2011 11th IEEE-RAS International Conference on Humanoid Robots*, pages 529–536. IEEE, 2011.

[4] M. Berg, D. Bayazit, R. Mathew, A. Rotter-Aboyoun, E. Pavlick, and S. Tellex. Grounding language to landmarks in arbitrary outdoor environments. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 208–215. IEEE, 2020.

[5] M. Bollini, S. Tellex, T. Thompson, N. Roy, and D. Rus. Interpreting and executing recipes with a cooking robot. In *Experimental Robotics: The 13th International Symposium on Experimental Robotics*, pages 481–495. Springer, 2013.

[6] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[7] B. Calli, A. Singh, A. Walsman, S. Srinivasa, P. Abbeel, and A. 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.

[8] Y. Chen, R. Gandhi, Y. Zhang, and C. Fan. Nl2tl: Transforming natural languages to temporal logics using large language models. *arXiv preprint arXiv:2305.07766*, 2023.

[9] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[10] D. Driess, F. Xia, M. S. Sajjadi, C. Lynch, A. Chowdhery, B. Ichter, A. Wahid, J. Tompson, Q. Vuong, T. Yu, et al. Palm-e: An embodied multimodal language model. *arXiv preprint arXiv:2303.03378*, 2023.

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

[12] N. Gopalan, D. Arumugam, L. L. Wong, and S. Tellex. Sequence-to-sequence language grounding of non-markovian task specifications. In *Robotics: Science and Systems*, volume 2018, 2018.

[13] J. Gu, Z. Lu, H. Li, and V. O. Li. Incorporating copying mechanism in sequence-to-sequence learning. *arXiv preprint arXiv:1603.06393*, 2016.

[14] E. Hsiung, H. Mehta, J. Chu, X. Liu, R. Patel, S. Tellex, and G. 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.

[15] W. Huang, P. Abbeel, D. Pathak, and I. Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International Conference on Machine Learning*, pages 9118–9147. PMLR, 2022.

[16] W. Huang, F. Xia, T. Xiao, H. Chan, J. Liang, P. Florence, A. Zeng, J. Tompson, I. Mordatch, Y. Chebotar, et al. Inner monologue: Embodied reasoning through planning with language models. *arXiv preprint arXiv:2207.05608*, 2022.

[17] B. Ichter, A. Brohan, Y. Chebotar, C. Finn, K. Hausman, A. Herzog, D. Ho, J. Ibarz, A. Irpan, E. Jang, R. Julian, D. Kalashnikov, S. Levine, Y. Lu, C. Parada, K. Rao, P. Sermanet, A. T. Toshev, V. Vanhoucke, F. Xia, T. Xiao, P. Xu, M. Yan, N. Brown, M. Ahn, O. Cortes, N. Sievers, C. Tan, S. Xu, D. Reyes, J. Rettinghouse, J. Quiambao, P. Pastor, L. Luu, K.-H. Lee, Y. Kuang, S. Jesmonth, N. J. Joshi, K. Jeffrey, R. J. Ruano, J. Hsu, K. Gopalakrishnan, B. David, A. Zeng, and C. K. 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.

[18] Y. Jiang, K. Zaporojets, J. Deleu, T. Demeester, and C. 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.

[19] E. Kolve, R. Mottaghi, W. Han, E. VanderBilt, L. Weihs, A. Herrasti, M. Deitke, K. Ehsani, D. Gordon, Y. Zhu, et al. Ai2-THOR: An interactive 3d environment for visual AI. *arXiv preprint arXiv:1712.05474*, 2017.

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

[21] K. Lin, C. Agia, T. Migimatsu, M. Pavone, and J. Bohg. Text2motion: From natural language instructions to feasible plans. *arXiv preprint arXiv:2303.12153*, 2023.

[22] B. Liu, Y. Jiang, X. Zhang, Q. Liu, S. Zhang, J. Biswas, and P. Stone. Llm+p: Empowering large language models with optimal planning proficiency, 2023.

[23] J. X. Liu, Z. Yang, I. Idrees, S. Liang, B. Schornstein, S. Tellex, and A. Shah. Lang2ltl: Translating natural language commands to temporal robot task specification. *arXiv preprint arXiv:2302.11649*, 2023.

[24] A. Madaan, S. Zhou, U. Alon, Y. Yang, and G. Neubig. Language models of code are few-shot commonsense learners. *arXiv preprint arXiv:2210.07128*, 2022.

[25] J. Malmaud, E. Wagner, N. Chang, and K. Murphy. Cooking with semantics. In *Proceedings of the ACL 2014 Workshop on Semantic Parsing*, pages 33–38, 2014.

[26] J. Marin, A. Biswas, F. Ofli, N. Hynes, A. Salvador, Y. Aytar, I. Weber, and A. Torralba. Recipe1m+: A dataset for learning cross-modal embeddings for cooking recipes and food images. *IEEE Trans. Pattern Anal. Mach. Intell.*, 2019.

[27] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins. Pddl-the planning domain definition language. 1998.

[28] C. Menghi, C. Tsigkanos, P. Pelliccione, C. Ghezzi, and T. Berger. Specification patterns for robotic missions. *IEEE Transactions on Software Engineering*, 47(10):2208–2224, 2019.

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

[30] J. Pan, G. Chou, and D. Berenson. Data-efficient learning of natural language to linear temporal logic translators for robot task specification. *arXiv preprint arXiv:2303.08006*, 2023.

[31] D. P. Papadopoulos, E. Mora, N. Chepurko, K. W. Huang, F. Ofli, and A. 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.

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

[33] R. Patel, R. Pavlick, and S. Tellex. Learning to ground language to temporal logical form. In *NAACL*, 2019.

[34] R. Patel, E. Pavlick, and S. Tellex. Grounding language to non-markovian tasks with no supervision of task specifications. In *Robotics: Science and Systems*, volume 2020, 2020.

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

[36] T. Silver, S. Dan, K. Srinivas, J. B. Tenenbaum, L. P. Kaelbling, and M. Katz. Generalized planning in pddl domains with pretrained large language models. *arXiv preprint arXiv:2305.11014*, 2023.

[37] I. Singh, V. Blukis, A. Mousavian, A. Goyal, D. Xu, J. Tremblay, D. Fox, J. Thomason, and A. Garg. Progprompt: Generating situated robot task plans using large language models, 2022.

[38] S. L. Smith, J. Tumova, C. Belta, and D. Rus. Optimal path planning for surveillance with temporal-logic constraints. *The International Journal of Robotics Research*, 30(14):1695–1708, 2011.

[39] A. Srivastava, A. Rastogi, A. Rao, A. A. M. Shoeb, A. Abid, A. Fisch, A. R. Brown, A. Santoro, A. Gupta, A. Garriga-Alonso, et al. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *arXiv preprint arXiv:2206.04615*, 2022.

[40] P. Stenetorp, S. Pyysalo, G. Topić, T. Ohta, S. Ananiadou, and J. 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.

[41] S. Vemprala, R. Bonatti, A. Bucker, and A. Kapoor. Chatgpt for robotics: Design principles and model abilities. *Microsoft Auton. Syst. Robot. Res*, 2:20, 2023.

[42] C. Wang, C. Ross, Y.-L. Kuo, B. Katz, and A. Barbu. Learning a natural-language to ltl executable semantic parser for grounded robotics. In *Conference on Robot Learning*, pages 1706–1718, 2021.

[43] H. Wang, G. Gonzalez-Pumariega, Y. Sharma, and S. Choudhury. Demo2code: From summarizing demonstrations to synthesizing code via extended chain-of-thought, 2023.

[44] J. Wu, R. Antonova, A. Kan, M. Lepert, A. Zeng, S. Song, J. Bohg, S. Rusinkiewicz, and T. Funkhouser. Tidybot: Personalized robot assistance with large language models. *arXiv preprint arXiv:2305.05658*, 2023.

[45] Y. Yang, Y. Li, C. Fermuller, and Y. 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.

[46] A. Zeng, A. Wong, S. Welker, K. Choromanski, F. Tombari, A. Purohit, M. Ryoo, V. Sindhwani, J. Lee, V. Vanhoucke, et al. Socratic models: Composing zero-shot multimodal reasoning with language. *arXiv preprint arXiv:2204.00598*, 2022.