Understanding Natural Language in Context

Avichai Levy, Erez Karpas

Technion

Abstract

Recent years have seen an increasing number of applications that have a natural language interface, either in the form of chatbots or via personal assistants such as Alexa (Amazon), Google Assistant, Siri (Apple), and Cortana (Microsoft). To use these applications, a basic dialog between the robot and the human is required.

While this kind of dialog exists today mainly within ”static” robots that do not make any movement in the household space, the challenge of reasoning about the information conveyed by the environment increases significantly when dealing with robots that can move and manipulate objects in our home environment.

In this paper, we focus on cognitive robots (Levesque and Lakemeyer 2008), which have some knowledge-based models of the world and operate by reasoning and planning with this model. Thus, when the robot and the human communicate, there is already some formalism they can use – the robot's knowledge representation formalism.

Our goal in this research is to translate natural language utterances into this robot's formalism, allowing much more complicated household tasks to be completed. We do so by combining off-the-shelf SOTA language models, planning tools, and the robot's knowledge-base for better communication. In addition, we analyze different directive types and illustrate the contribution of the world's context to the translation process.

Introduction

As time goes by, we see a significant increase in the use of virtual assistants such as Amazon's Alexa, Google Assistant, Apple's Siri, and many more. Although these virtual assistants are able to perform basic household tasks using online communication with smart home products, there are many more tasks that remain unsatisfied.

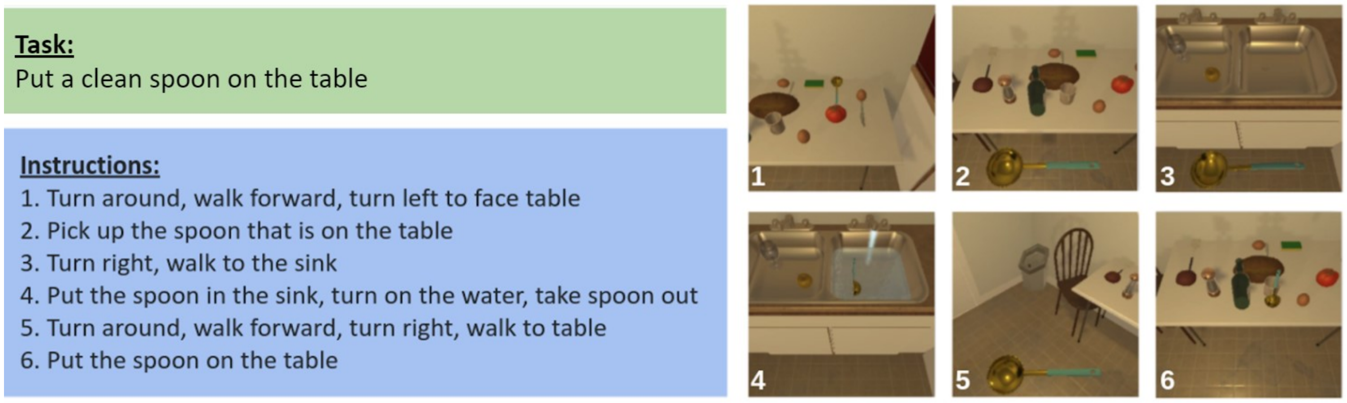
The next generation of these assistants are robots that operate in our houses, follow instructions given by us and perform much more complicated tasks. To make this dream come true, a robot must have the ability to plan a series of actions from an initial state until the required task is completed. Such a planning process could be accomplished via the PDDL formalism (McDermott et al. 1998).

Another basic requirement for household robots is the ability to reason about information that comes from the environment. This information can be visual, textual, audio, etc. As for the language reasoning part, in 2017, the Transformer neural network architecture appeared (Vaswani et al. 2017), reaching a great breakthrough in machine translation. Later on, transformer architectures evolved, solving a larger range of NLP tasks. Yet, training an agent to perform complicated actions in the real world is still hard, expensive, and time-consuming. Therefore, a lot of recent work has been dedicated to the development of realistic virtual simulators that mimic the behavior of the real world.

AI2-Thor (Kolve et al. 2017) is an example of this kind of simulator. The AI2-Thor simulator demonstrates realistic constraints from our real world. A good example of a real-world constraint is an **irreversible state** in which some actions change the world in a way that cannot be reversed (for example: the only tomato in the scene was sliced). In 2020, Shridhar et al. introduced the ALFRED (Action Learning from Realistic Environments and Directives) dataset, which is based on the AI2-Thor simulator. ALFRED consists of multi-modal data. It has a long sequence of instructions to achieve high-level tasks such as ”Put a slice of tomato in the fridge.” Those step-by-step instructions are combined with the egocentric vision of the robot at each time step (see 1 for example).

Our goal in this paper is to develop a system that incorporates natural language processing and planning to enable an agent to accomplish a real-world task given in natural language. By doing so, we are able to show how changes in the language input affect the agent's capability to accomplish the task.

In this work, we assume that the agent has some background knowledge about the environment it operates in (which is essential for any cognitive robot regardless of its natural language processing capabilities), such as the types of objects in the world, and their features and the possible relations between them, as well as the basic robot behaviors (modeled as actions with preconditions and effects). Moreover, we assume that at the beginning of every episode, the agent acquires complete scene information, including all the objects and their locations, using its vision tools. The last piece of the agent's input is a directive, in which a human asks the agent to perform a specific task in natural language. By combining all this information, our agent should generate a sequence of actions (in the robot's formalism) that will achieve the desired outcome of the human task. To do so,



Task:
Put a clean spoon on the table

Instructions:
1. Turn around, walk forward, turn left to face table
2. Pick up the spoon that is on the table
3. Turn right, walk to the sink
4. Put the spoon in the sink, turn on the water, take spoon out
5. Turn around, walk forward, turn right, walk to table
6. Put the spoon on the table

Figure 1: An example from the ALFRED dataset. The green text box contains the high level task and the blue text box contains the high level instructions needed for accomplishing that task. The images represent the egocentric vision input of the agent at each time step.

we have developed a system which combines large language models and PDDL planning. We evaluate this system on the textual part of the ALFRED dataset. We show how various inputs affect the outcome of the model and suggest that the context of the environment is vital for the agent's ability to succeed as well.

Our main contributions are as follows:

- We developed a novel translation approach that combines natural language and planning formalism for operating agents.
- We show that integrating the scene context (the world's semantic information) into the model's input leads to a significant improvement in the translation process, which indicates that the context of the world is vital for an operating agent.
- We imply that both the Transformer's encoder and the Transformer's decoder are essential for this translation task.

Background

In this section, we review some relevant background on language models and PDDL formalism.

Language Models

Nowadays, there are many Transformer-based natural language models. Among them are OpenAI's GPT, Google's BERT, and T5. In this work, we focus on the GPT-2 and T5 models.

Transformers Transformers were introduced in 2017 by a team at Google Brain (Vaswani et al. 2017) and are usually the model of choice for NLP problems. Transformers are designed to handle sequential input data, however, they do not necessarily process the data in order. That is, transformers allow parallelization and therefore reduce training time. The encoder and decoder are the basic components of the original Transformer's architecture. The encoder processes the input, while the decoder consists of decoding the encoder's output. The key component in the transformer architecture is the attention mechanism. In general, the attention mechanism focuses on certain parts of the input sequence when predicting a certain part of the output sequence. In other words, when generating the output, the attention unit will dynamically highlight relevant features of the input data and generate each output token according to its relative attention weights. The transformers had great success in natural language processing (NLP) tasks, and most of the latest NLP models are based on this neural network architecture.

GPT-2 The GPT-2 model was created by OpenAI in February 2019 (Radford et al. 2019). Unlike other transformer models, GPT-2 is a decoder-only transformer model, consists of 1.5 billion parameters and was trained on a dataset of 8 million web pages. GPT-2's goal is similar to other language models, which is to predict the next word, given all the words generated so far. Since GPT-2 was not trained on a specific task, it is a general-purpose learner, able to answer questions, translate sequences and summarize text. When published, the model achieved state-of-the-art results on 7 out of 8 tested language modeling datasets in a zero-shot setting. During its training process, GPT-2 was given different text sequences, separated by the $<|endoftext|>$ token. In this work, we also use a $<|startoftext|>$ token that indicates the beginning of the sentence.

T5 T5 was introduced by Google in 2019 (Raffel et al. 2019). Unlike the GPT architecture, T5 uses both the transformer's encoder and decoder. The idea behind the T5 model is to convert every language problem into a text-to-text format. This approach enables using the same model, objective, training procedure, and decoding process for different tasks, such as machine translation, sentiment analysis, summarization, and question answering. By using transfer learning techniques and training on a large corpus of web-scraped data, T5 achieves state-of-the-art results on a number of NLP tasks. Each training input consists of a prefix, source, and target. The prefix is task-specific and is paired with every input. For example, "translate English to German: text". Adding such a prefix enabled the model to tune its weights for a particular task, which helps to produce the expected output for that task alone by narrowing its scope of generation.

PDDL

The Planning Domain Definition Language (PDDL) (McDermott et al. 1998) is a language family that allows us to define a planning problem. Nowadays, there are many versions of PDDL. PDDL is an action-centred language, inspired by the well-known STRIPS formulations of planning problems (Fikes and Nilsson 1971). Mathematically, a STRIPS instance is a quadruple $\langle F, A, I, G\rangle$, in which each component has the following meaning:

- F – a set of facts describing the possible states of the world, 2^F.
- A – a set of actions. Each action $a \in A$ consists of a set of preconditions $pre(a)$, add effects $add(a)$, and delete effects $del(a)$. Applying a is possible in a state s where $pre(a) \subseteq s$, and results in the state $s[\langle a\rangle] = (s \backslash del(a)) \cup add(a)$.
- $I \subseteq F$ is the initial state of the world.
- G – the goal of the problem. The goal G is a set of facts $G \in F$. A state s satisfies a goal if $G \subseteq s$.

A plan π is a sequence of actions. $\pi = \langle a_0, a_1, \ldots, a_n\rangle$ is applicable from state s_0 if a_0 is applicable at s_0 and $\langle a_1, \ldots, a_n\rangle$ is applicable from $s_1 := s_0[\langle a_0\rangle]$. We denote the state reached by following plan π from state s by $s[\pi]$, and say that a plan π achieves a goal G if $G \subseteq s[\pi]$. The PDDL language generalizes the STRIPS setting into domain description and problem description.

The domain description contains the definitions of object types, predicates, as well as the actions' preconditions and effects. These elements are the aspects that do not change regardless of what specific situation we are trying to solve. On the other hand, the PDDL problem description is more specific and defines exactly what objects exist in the scene, what their current states are, and what the goal is.

For example, let us assume that the world contains apples, tomatoes, cucumbers, knives, and tables in three different colors: blue, yellow, and green. The actions that could be conducted on an object are: pickup, put, and slice. A problem file can be described as follows: The current scene contains two apples, three tomatoes, two knives, a yellow table, and a blue table. In the initial state, all the objects (besides the blue table) are on the yellow table. The goal is to put a slice of an apple on the blue table.

Many PDDL tools have been developed over the years. One major part is PDDL planners, which read PDDL files (domain and problem) and use them to find a sequence of actions that solves the problem. Another tool is a plan validator (Howey, Long, and Fox 2004), which checks if a given plan solves a specific PDDL problem.

Related work

Home service robots must have the ability to plan a sequence of actions to achieve their goals in the real world. This skill requires sophisticated reasoning at each time step, including interpreting multi-modal input types such as vision, language, and other sensor-type information. Thanks to environments like AI2-THOR(Kolve et al. 2017), Matterport 3D (Anderson et al. 2018), AI Habitat (Savva et al. 2019), and TDW(Gan et al. 2020), a dramatic improvement has been made in various real-world tasks. One of these is *visual semantic planning* (Zhu et al. 2017), which is the task of predicting a sequence of high-level actions from visual observations. The purpose of those actions, conducted by the agent, is to reach a goal state from an initial state. When addressing this kind of task, a robot operating in a human household space may need to overcome some challenges. For example, partially observable space or long-horizon tasks in which the decision-making at any step can depend on observations received far in the past. Hence, being able to properly memorize and utilize long-term history is crucial (Fang et al. 2019). In 2020, Shridhar et al. introduced the ALFRED dataset, which is based on the AI2-Thor simulator. ALFRED combines both egocentric vision and language directives to achieve everyday household tasks. Currently, the best model completed 39% of ALFRED's tasks, which still leaves a long way to go.

Recent papers have chosen to break this problem down into separate modalities instead of solving this difficult multi-modal problem. (Jansen 2020) explored this task on the ALFRED dataset, by using the GPT-2 language model to generate these plans from high-level task descriptions, without visual cues. In his work, Jansen showed that the GPT-2 model outperforms a baseline RNN model on this task, predicting successfully of 22.2% actions sequences, and 53.4% of the plans when ignoring the first action prediction in the sequence. Later on, (Wang et al. 2021) integrated a general domain knowledge graph of indoor environments with the BERT model (Devlin et al. 2018) to create better predictions, generating successfully 31.4% of the plans. While these previous works focused on language directive translation, they do not incorporate practical planning, and therefore are not sufficient for real-world intelligent agents.

On the other hand, (Wang, Tian, and Shao 2020) integrated Hierarchical Task Network and Probabilistic Inference to generate action sequences using multiple context types, but without natural language directives. These papers indicate that models can achieve surprising performances using information from only a single modality. In addition, a recent study (Thomason, Gordon, and Bisk 2018) has found that models using input from a single modality (either vision or language) often perform nearly as well as or even better than their multi-modal counterparts.

Methods

As described before, we assume that our agent already has background knowledge about the available actions, objects, and predicates in the world. Another assumption is the agent has complete scene information (typically acquired using vision). Additionally, the agent is given a language directive by a human, which instructs it to perform a particular task. The agent's goal is to output a sequence of actions that achieves the desired objective.

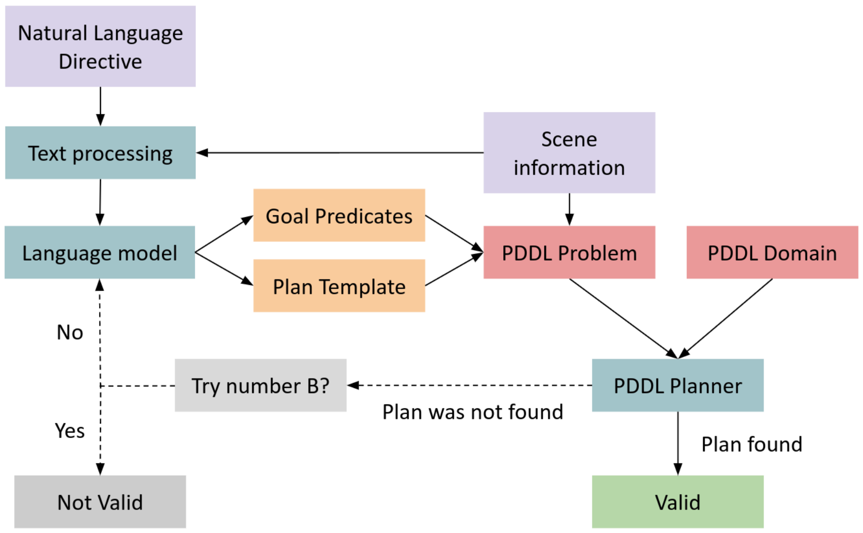


Figure 2: Our model architecture.

In general, a human can give language commands to a robot in multiple ways. The first option is to provide the agent with high-level task description, such as ”put a slice of tomato in the fridge”. On the other hand, more detailed instructions could be given also, for example, ”go to the kitchen, pick up the knife from the table, go to the tomato that is on the counter, slice the tomato, etc.”. While both of these types exist in each sample of the ALFRED dataset, we choose to focus on the high-level task description. Moreover, in this paper we incorporate the spatial relations between the objects in the scene, which reflects the semantic context of the environment. We term the high-level task description as ”task”, and the additional context as ”relations”.

Furthermore, we use the task and relations to train our model to produce two outputs: *goal predicates* and *plan templates*. The *goal predicates* represent the desired outcome of a given task description. In other words, the goal predicates are the main objects' state at the end of the plan execution (”sliced tomato, tomato on counter”). On the other hand, the *plan template* is the general structure of the robot's plan, which specify a sequence of actions and objects types that form the plan (”go to table, pickup knife table, go to tomato, slice tomato”).

By combining these outputs together, we are able to achieve our goal, which is to translate natural language into a **valid** robot-language plan. A valid plan is a sequence of actions in a robot's language that can be performed by a household agent to achieve given tasks.

In this work, we assume that the robot's language is the Planning Domain Definition Language (PDDL). As mentioned above, we assume that the agent has some background knowledge about the environment. We encode this knowledge in the PDDL domain, which include the objects in the world and the actions' definitions (preconditions and effects). Unlike the domain file, which is predetermined, the problem file varies between each task we are trying to solve. The main components of the problem file are the task's goal predicates and the current world state, which includes the objects in the scene and their predicates.

By adding predicates to the action's parameters and the world state, we restrain the problem, allowing the PDDL planner to terminate much faster. In our research, we add three types of constraints:

- **Length** - we add two predicates to the domain - $(next\ ?s_i\ ?s_j)$ and $(current\_step\ ?s_i)$. Each action in the domain gets the current step, s_i, and increases it to s_j, where s_j is the next step number in the $(next\ s_i\ s_j)$ predicate. By initializing the problem with the predicates:
  - $(current\_step\ s_0) = True$
  - $(next\ s_i\ s_{i+1}) = True\ \ \forall i:\ 0 \leq i < T$

  and adding the predicate $(current\_step\ s_T)$ to the goal, we force the PDDL planner to generate only plans of length T.
- **Action allowance** - for each time step $i \geq 0$, we add the predicate $(allowed\_action\ s_i)$, where *action* is some action type from the domain, making the planner to create only plans that their i'th action type is allowed.
- **Objects allowance** - This predicate is similar to the previous one. The predicate is $(allowed\_object_j\ obj\ s_i)$. Which indicates if the i'th action in the plan, $(action\_type\ obj_1 \ldots obj_k)$, may contain the object type obj at location j.

The combination of these constraints forces the PDDL planner to create a very specific PDDL plan. Therefore, these constraints reduce the search space, hence accelerating the process of finding a plan, if one exists.

An illustration of our model's architecture is available in Figure 2. The first component is the translation unit, which is responsible for translating a natural language directive into PDDL goal and plan template. The second part of the model will combine these elements into a PDDL problem and check for a PDDL plan that solves this problem. If such a plan is not found, the model will generate another PDDL plan template and re-check for a solution. Ultimately, this process ends when a valid plan is found or the number of plans generated exceeds a constant B. We now drill down into further details of each component.

Language-PDDL translation This unit consists of two translation channels. Both channels take the same natural language directive as input. The first channel and the primary meeting point between the language part and the PDDL part is the *plan templates*. A *plan template* (or *visual semantic plan* as in (Jansen 2020)) is a sequence of actions that achieve a general goal. Each action consists of an action type and parameter types. These actions are not object-specific, meaning that they cannot be used together as a plan for the PDDL problem and therefore not sufficient for an operating agent either. For instance, a plan template could be "go to dining table, pick up apple dining table, go to fridge, put apple fridge". Since there might be multiple apples and tables in our scene, the robot will not know which objects it should integrate with. In other words, a valid PDDL plan must contain the objects' unique ids as well as their types.

Even though the plan templates alone are not enough to solve each problem, they are still useful. In our method, we convert these plan templates into the PDDL constraints we defined above. To demonstrate this, consider the following plan template: *"go to table, pick up apple table"*.

The constraint predicates that derive from this plan template are:

- **Length** - since the length of the plan is 2, we add the predicates $(next\ s_0\ s_1)$, $(next\ s_1\ s_2)$, $(current\_step\ s_0)$ to the initial world state and $(current\_step\ s_2)$ to the goal predicates.
- **Action allowance** - for each element in the predicted sequence, we add the action allowance predicate of the element's action type and index - $(allowed\_goto\ s_0)$, $(allowed\_pickup\ s_1)$.
- **Objects allowance** - as in the previous case, we add the object allowance predicate for each object in the sequence. Each predicate consists of the object type, its location in the action, and its action's location in the sequence. - $(allowed\_arg_1\ table\ s_0)$, $(allowed\_arg_1\ apple\ s_1)$, $(allowed\_arg_2\ table\ s_1)$.

By integrating these constraints into the original PDDL problem, we force the planner to produce a plan that matches our template. This restriction significantly decreases the search space, thus accelerating the search process overall.

The second translation channel is the goal predicates unit. The goal predicates capture the user's desired outcome of a task. These predicates describe the world's state at the end of the plan's execution ("sliced tomato", "cold tomato"). In contrast to the PDDL plan, which should include specific objects, the goal predicates may be formulated in a more general way. In this paper, we use this general goal formalism rather than focusing on specific object ids. In other words, when the task's goal includes some predicates of a specific object, we accept any plan that reaches the same predicates on any instance of this object type. This is done due to the fact that there are many instances of each object type in our world, and we do not want to pick only one instance when we formulate the goal. We implement this attribute using the *exists* PDDL operator.

To illustrate this, assume the PDDL goal *"sliced tomato, on tomato countertop"*. This goal will be formulated as: *"(exists (?tomato0 - tomato ?countertop0 - countertop) (and (sliced ?toamto0) (on ?tomato0 ?countertop0)))"*

This encoding allows us to accept any plan that achieves a final world state in which there exists a sliced tomato on any countertop.

After generating these two PDDL elements, we combine them with the original PDDL problem, creating a new and more restricted problem to solve. As in earlier work, we model this translation process as a sequence-to-sequence task. Moreover, in our research, we focused on two language models, GPT-2 and T5. We have trained separate GPT-2 models for *goal predicates* prediction and for *plan template* prediction. However, since training a new task on T5 requires only changing the prefix of the input, we fine-tuned a single T5 model for both tasks.

PDDL consistency checking Once a new PDDL problem has been generated from the predicted goal and the constraint predicates, we will input both the domain of our world and the problem file into a PDDL planner. The planner will look for a valid plan that achieves our goal under the given constraints.

When the planner does find a solution, we count the plan template predictions as valid. On the contrary, when the planner does not find a solution, we will go back to our language model and "ask" it to generate another plan template. After a new template was generated, we convert it to PDDL constraints, update the PDDL problem, and check for a valid plan that fits the new template. This re-generation of the plan template is done by taking the next prediction in the model's beam search output. We repeat this procedure until a valid template is found or the number of templates generated exceeds a given number B.

Various Input types When a human approaches everyday tasks, he may have some preliminary knowledge about the world he operates in. We term this knowledge the "context of the environment" which includes, among other things, the objects in the scene, their spatial relations, and action-object pairs that commonly appear together. When providing only a task description to a robot that does not have this context knowledge, it may struggle to generate a successful plan.

In this paper, we suggest that adding the context of the environment to the task description as the input for the model (rather than using the task description alone), improves the quality of the model's output. We check this hypothesis by providing our model with various input types and tracking the changes in its performance.

Concretely, we perform multiple experiments, each having its own language input. The directives that were tested are the concatenation of the high-level task ("put two bowls on the dining table") with the relations between the objects in the scene ("on tomato table, on bowl countertop, etc."). In addition, to isolate the effect of each input type, we also analyze the performance of the model when the input contains only one type.

Evaluation

We evaluate our model on the language part of the ALFRED dataset and show that our model is able to achieve state-of-the-art results on the *visual semantic plan* generation task and *valid robot-plan* generation task.

Dataset

The ALFRED dataset consists of 8,055 visual samples, composed of an agent's egocentric visual observations of the environment. Each one corresponds to multiple language directives, annotated by mechanical turkers, adding up to a total number of 25,743 directives. ALFRED has 7 different task types parameterized by 84 object classes in 120 scenes. The tasks are Pick & Place, Stack & Place, Pick Two & Place, Clean & Place, Heat & Place, Cool & Place and Examine in Light. ALFRED is based on the AI2-Thor simulator, in which some actions may change the object's state in an irreversible way (a sliced potato will never be whole again). The evaluation data in ALFRED is divided into validation and test datasets. Each one is split also into seen and unseen environments. The purpose of the second split is to examine how well a model generalizes to entirely unseen new spaces with novel object class variations.

Pre-Processing In our work, we redivide the original ALFRED's training data into train, val, and test. Furthermore, we combine ALFRED's seen type validation set with our validation set and test our model both on our test data and ALFRED's validation unseen data. Since in our task we ignore the vision part of the data, we might encounter some duplicates between our datasets. Hence, we perform a cleaning process that deletes duplicate samples from the training and validation data with the same language directive as in our test datasets.

In the ALFRED dataset, there are several actions that the agent can execute. These actions are: $pickup$, put, $slice$, $heat$, $cool$, $clean$, $toggle$ and $goto$. By performing a single action or a sequence of actions, the agent may change the state of some objects. To track these changes, we model the state of each object by using the PDDL predicates formalism. The predicates that reflect the outcomes of these actions are $robot\_has\_obj$, on, $sliced$, hot, $cold$, $cleaned$, $toggled$ and can_reach. In addition, we added the predicate two_task which is an indicator for the "pick two" task.

In our work, we train language models to predict both the goal predicates, which express the desired final state of each object as derived from the language directive, and the plan template, A.K.A the *visual semantic plan*. To create the targets for the language models, we focused on the PDDL parameters and the high-level actions provided by the ALFRED samples. Each action in the plan template is in the form - $(action,\ arg1)$ if $action \in \{goto,\ toggle\}$, and $(action,\ arg1,\ arg2)$, otherwise. In the same way, each goal predicate is in the form - $(predicate,\ arg1,\ arg2)$, if $predicate = on$, and $(predicate,\ arg1)$, otherwise.

Models input format Since we use two different language models in our evaluation, GPT-2 and T5, we have to adjust the input to the correct form that these models accept. We fine-tune GPT-2 on the natural language directives and gold targets using the GPT's sos and eos tokens:

"<|*startoftext*|> **directive** *Task Type:* **target** <|*endoftext*|>"

Where *Task Type* is either "Goal" or "Actions" according to the prediction task we are performing. During evaluation and testing, we feed the model with the input "<|*startoftext*|> **directive** *Task Type:*", and let it generate tokens until a <|*endoftext*|> token is generated.

On the other hand, the T5 fine-tuning process on a new task is done by providing a unique prefix before the directive. In our work, the goal-predicates task's prefix is "translate task to goal" and the plan-template task's prefix is "translate plan to actions". Since T5 is an encoder-decoder model, at every training step we feed the model with source and target sequences. The source phrase is the prefix with the directive, and the target phrase is either the goal predicates or the plan template.

Data Validation A data sample will be considered *valid* for training if its original action sequence solves the sample's problem. To check if a given solution does solve a given task, we need a PDDL domain file and a PDDL problem file. Thus, we have created a PDDL domain file using our knowledge of the objects and actions in the ALFRED world and a PDDL problem file for each sample. The ALFRED domain file consists of the rules of the ALFRED world and its object types. While the same domain file is used across all samples, the problem file is different between samples. Moreover, creating a PDDL problem file requires knowing the world's current state, meaning, all the objects in our scene and their spatial relations. Although the ALFRED dataset samples provide some of the objects in the scene, it neither reveals all of the objects nor their spatial relations, but only their explicit coordinates in the space. To find the objects' relations, we used the initial location of each object and the scene type, taken from the ALFRED dataset, and loaded them into the AI2-Thor simulator. By doing so, we achieved the metadata of the scene, which provides more information about objects and their spatial relations. Concretely, we create relations in the form *on* obj_1 obj_2, where obj_1 is on top of obj_2 or inside it. Lastly, the goal predicates for each problem were generated from the "PDDL parameters" field of every data sample.

After creating the domain and problem files, we extracted the PDDL action sequence from the "high level plan" of each sample (which specifies the objects' ids) and used the VAL plan validator (Howey, Long, and Fox 2004) to check if this plan did solve the PDDL problem of this sample. Samples whose gold PDDL action sequence did not achieve the goal of the problem were marked as invalid samples and were removed from the data. Eventually, the train, val, test, and test_unseen datasets had 13893, 1650, 1010, and 682 samples, respectively. This division reflects an 80-10-10 (%) train-val-test partition.

Scoring Type	Model	Input	Predicate	Arg1	Arg2	F_Predicate	F_Seq
Strict	GPT-2	Task	0.80	0.77	0.79	0.76	0.66
		Relations	0.02	0.01	0.02	0.02	0.01
		Task + Relations	0.73	0.71	0.81	0.70	0.74
	T5	Task	0.89	0.86	0.84	0.85	0.78
		Relations	0.09	0.08	0.05	0.08	0.04
		Task + Relations	**0.92**	**0.89**	**0.88**	**0.89**	**0.85**
Permissive	GPT-2	Task	0.80	0.81	**0.89**	0.80	0.72
		Relations	0.02	0.01	0.02	0.02	0.01
		Task + Relations	0.73	0.74	0.83	0.73	0.77
	T5	Task	0.89	0.89	0.85	0.89	0.84
		Relations	0.09	0.09	0.05	0.09	0.04
		Task + Relations	**0.92**	**0.92**	**0.89**	**0.92**	**0.88**

Table 1: *Goal Predicates* Precision accuracy scores.

Metrics In our research, we implement both the evaluation measures used in (Jansen 2020) and some additional accuracy measures. Moreover, we extend these measures to the *goal predicate* task. We also use the same notation of permissive scoring, which accepts predictions of objects that are similar to the original ones. ("lamp - floor lamp", "knife - butter knife"). Both tasks have per-element accuracy measures ($predicate\backslash command,\ arg_1,\ arg_2$), permissive arguments ($p\_arg1,\ p\_arg2$), full triples and full sequence accuracy measures as defined in (Jansen 2020).

Notice, however, that while in the *visual semantic plan* task the order of the generated text does matter ("go to table, pick up tomato table" is not the same as "pick up tomato table, go to table"), in the *goal predicate* task we ignore the order of the generated predicates and measure the accuracy accordingly ("sliced tomato, cold tomato" is the same as "cold tomato, sliced tomato"). In the *goal predicate* task, we also look at permissive scoring in the full predicate and sequence level. The $f\_predicate\_sim$ and f_seq_sim measures indicate if a predicate or a sentence is wrong only in permissive objects ("cold butter knife, hot apple" and "cold knife, hot apple" are the same). We implemented two accuracy measures for the *valid robot-plan* task. The first measure is the $Valid\_Plans\_O$ measure, which indicates the ratio of samples that a valid PDDL plan (achieves the **original** goal predicates) was found, while following the plan template constraints. The second measure is similar to the first, except that it counts plans that achieve the **predicted** goal predicates. We term the second measure $Valid\_Plans\_P$.

Results

Goal Predicates In this section, we will analyze the results on both tasks, tested on two language models and three directive types. The accuracy measures in this section were calculated using the precision definition and were evaluated on the ALFERD's val_unseen dataset.

The models' accuracy scores on the goal predicate task are shown in Table 1. We have trained and tested the models on various inputs. While in the T5 model the accuracy scores were the highest on the *task + relations* input, in the GPT-2 model the *task + relations* input was better than the *task* input only on the full_sequence measure. In addition, T5 outperform GPT-2 on every input type, reaching almost 90% accuracy across all measures and predicting correctly 85% of exact full sequences. These results suggest that an encoder-decoder architecture might be more suitable for goal predicate prediction. Moreover, the additional information about the environment was captured better by the T5 model than by the GPT-2 model. Further examples of the goal prediction of our T5 model on new and unique examples are shown in 4. These directives are different from the common tasks of ALFRED, and their intention is to check the robustness of the model. In the second sample, we add the phrase "avoid using lettuce", and the model changes its original prediction to a potato instead of lettuce. Moreover, the model seems to recognize general types such as vegetables, cutlery, and baking tools as well.

Plan Template Table 2 contains the models' scores on the *plan template* task. In contrast to the *goal predicate* task results, both models achieve the highest score on the Task + Relations input. On the Task-only input, GPT-2 predicts correctly 32% of full original action sequences, which is better result from previous work (22%), but might be due to the training dataset changes. Furthermore, when adding the scene context to the model's input, T5 outperforms GPT-2 across almost all measures, predicting correctly 57% of the target plans in comparison to GPT's 46%. These results outperform recent work on *visual semantic plan* generation from natural language directives, which was also trained and evaluated on the ALFRED dataset.

Valid Robot Plan The models' scores on the *valid robot plan* task are also presented in Table 2. As shown in the table, T5 model generates valid plans for 91% of the samples, where 57% of them are the exact target plans. That is, 34% of the plan templates that T5 predicted were not the same as the original plans, but eventually solved the given task.

In addition, when the input is non-informative of the re-

Model	Input	Command	Arg1	Arg2	F_Action	F_Seq	Valid Plans O	Valid Plans P
GPT-2	Task	**0.93**	0.75	0.67	0.63	0.32	0.78	0.69
	Relations	0.54	0.14	0.16	0.10	0.00	0.00	0.00
	Task + Relations	**0.93**	0.78	0.74	0.69	0.46	0.89	0.83
T5	Task	0.91	0.73	0.63	0.60	0.29	0.72	0.77
	Relations	0.68	0.22	0.26	0.18	0.04	0.13	0.59
	Task + Relations	0.92	**0.82**	**0.76**	**0.75**	**0.57**	**0.91**	**0.97**

Table 2: *Plan Template* and *Valid Plans* Precision accuracy scores.

Model	GoTo	Pickup	Put	Cool	Heat	Clean	Slice	Toggle	Avg.
GPT_T	68	40	68	**85**	82	**78**	39	75	67
T5_T	66	36	66	79	83	74	47	77	66
GPT_{TR}	75	56	67	78	80	75	**55**	75	70
T5_{TR}	**79**	**65**	**72**	83	**84**	**78**	**55**	**97**	**77**

Table 3: Full actions triples accuracy (percentages), broken down to 8 actions types in ALFRED. The subscript stands for Task only input (T) or Task + Relations input (TR).

quired task, such as the objects' relations alone, GPT-2 is not able to generate any valid plan. On the other hand, T5 generated valid plans for 59% of the samples with respect to the predicted goal, but only 13% plans with respect to the original goal predicates. These results suggest that T5 might generate easier goal predicates when the input is non-informative rather than succeeding to predict valid plans.

Lastly, GPT-2 achieve better results on the original goals, in contrast to T5 which succeed more on the predicted goal predicates. This difference might be due to the fact that T5 succeed more on the *goal predicate* task than GPT-2.

Few shot learning In our setting, creating data samples for training is time-consuming and expensive. Hence, the ability of a model to perform successful few-shot learning is crucial. To evaluate this capability, we have created multiple training sets by downsampling the original data into smaller fractions and trained different T5 models on each set. As shown in figure 3, we see that with only 5% of the training data, our models are able to predict actions sequences and goal predicates nearly as well as models that were trained on the full dataset. These results suggest that our model is able to perform successful few-shot learning. We are planning to test this assumption in other domains in future work.

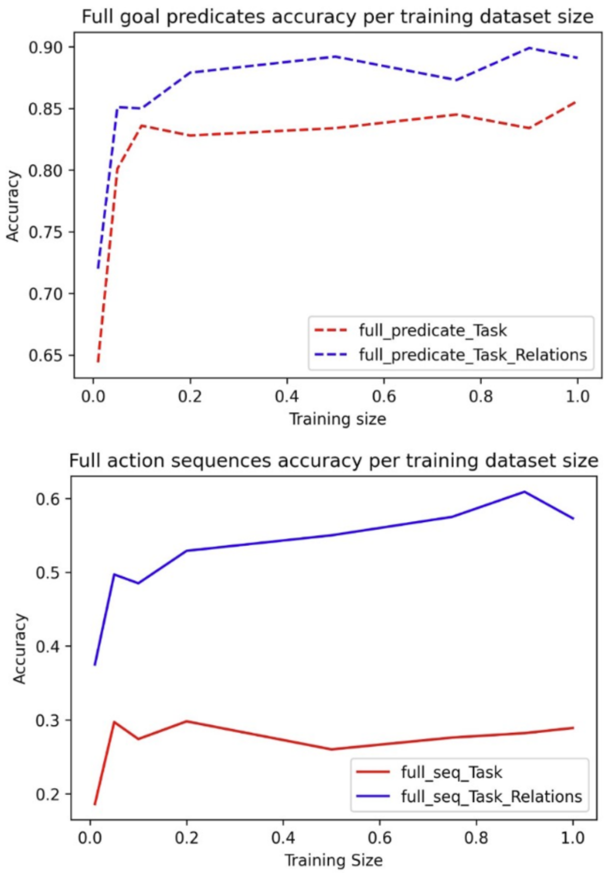


Figure 3: Few-shot accuracy scores.

Conclusions and Future Work

In this work, we have developed a novel approach for plan generation that solves everyday household tasks. When given a natural language high-level task description together with the world's context, our model predicts precisely 57% of the ALFRED dataset plans, and generates valid plans for another 34% of the samples. Reaching a total number of valid plans for 91% of unseen environments tasks. Our approach combines both language models and PDDL tools, working together as a whole to generate a valid PDDL plan that will achieve the language directive goal.

Looking forward, there are some future directions we would like to investigate. One of them is generating valid PDDL plans for object-specific goals. This task is more challenging since its goal state requires particular objects' predicates to be changed rather than any object of this type (for example, warming a **green** cup instead of any cup). Another

Input Text	Goal Predicates
Task: put either tomato or potato or lettuce on the counter.	on lettuce countertop, cold lettuce
Task: put either tomato or potato on the counter, **avoid using lettuce**.	on **potato** countertop, cold **potato**
Task: put a baking tool on the counter.	on spatula pan, on pan countertop
Task: place two vegetables in the drawer.	on potato drawer, two_task
Task: put any type of cutlery on the counter.	sliced spoon, on spoon cup, on cup countertop
Task: put some element in the fridge.	sliced potato, on potato fridge, cleaned potato

Table 4: Goal prediction examples of our T5 model, which was trained on the Task + Relations input. In the left column are the new task inputs for the model. Each task was paired with the objects' relations in the scene. These directives are different from the common tasks of ALFRED, and their intention is to check the robustness of the model.

challenge is defining a more conservative PDDL problem and domain that avoids changing basic attributes of objects (for example, put a **whole** tomato on the table). In addition, we would like to improve our zero-shot predictions on new and unseen object types.

References

Anderson, P.; Wu, Q.; Teney, D.; Bruce, J.; Johnson, M.; Sünderhauf, N.; Reid, I.; Gould, S.; and Van Den Hengel, A. 2018. Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 3674–3683.

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

Fang, K.; Toshev, A.; Fei-Fei, L.; and Savarese, S. 2019. Scene memory transformer for embodied agents in long-horizon tasks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 538–547.

Fikes, R. E.; and Nilsson, N. J. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3-4): 189–208.

Gan, C.; Schwartz, J.; Alter, S.; Schrimpf, M.; Traer, J.; De Freitas, J.; Kubilius, J.; Bhandwaldar, A.; Haber, N.; Sano, M.; et al. 2020. Threedworld: A platform for interactive multi-modal physical simulation. *arXiv preprint arXiv:2007.04954*.

Howey, R.; Long, D.; and Fox, M. 2004. VAL: automatic plan validation, continuous effects and mixed initiative planning using PDDL. In *16th IEEE International Conference on Tools with Artificial Intelligence*, 294–301.

Jansen, P. A. 2020. Visually-Grounded Planning without Vision: Language Models Infer Detailed Plans from High-level Instructions. *arXiv preprint arXiv:2009.14259*.

Kolve, E.; Mottaghi, R.; Han, W.; VanderBilt, E.; Weihs, L.; Herrasti, A.; Gordon, D.; Zhu, Y.; Gupta, A.; and Farhadi, A. 2017. Ai2-thor: An interactive 3d environment for visual ai. *arXiv preprint arXiv:1712.05474*.

Levesque, H.; and Lakemeyer, G. 2008. Cognitive robotics. *Foundations of artificial intelligence*, 3: 869–886.

McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL-the planning domain definition language.

Radford, A.; Wu, J.; Child, R.; Luan, D.; Amodei, D.; Sutskever, I.; et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8): 9.

Raffel, C.; Shazeer, N.; Roberts, A.; Lee, K.; Narang, S.; Matena, M.; Zhou, Y.; Li, W.; and Liu, P. J. 2019. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*.

Savva, M.; Kadian, A.; Maksymets, O.; Zhao, Y.; Wijmans, E.; Jain, B.; Straub, J.; Liu, J.; Koltun, V.; Malik, J.; et al. 2019. Habitat: A platform for embodied ai research. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 9339–9347.

Shridhar, M.; Thomason, J.; Gordon, D.; Bisk, Y.; Han, W.; Mottaghi, R.; Zettlemoyer, L.; and Fox, D. 2020. Alfred: A benchmark for interpreting grounded instructions for everyday tasks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 10740–10749.

Thomason, J.; Gordon, D.; and Bisk, Y. 2018. Shifting the baseline: Single modality performance on visual navigation & qa. *arXiv preprint arXiv:1811.00613*.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L.; and Polosukhin, I. 2017. Attention Is All You Need. *CoRR*, abs/1706.03762.

Wang, K.; Zhang, Y.; Jiang, C.; Luo, J.; Yang, X.; and Chen, S. 2021. Visual Semantic Planning for Service Robot via Natural Language Instructions. In *2021 China Automation Congress (CAC)*, 793–798. IEEE.

Wang, Z.; Tian, G.; and Shao, X. 2020. Home service robot task planning using semantic knowledge and probabilistic inference. *Knowledge-Based Systems*, 204: 106174.

Zhu, Y.; Gordon, D.; Kolve, E.; Fox, D.; Fei-Fei, L.; Gupta, A.; Mottaghi, R.; and Farhadi, A. 2017. Visual semantic planning using deep successor representations. In *Proceedings of the IEEE international conference on computer vision*, 483–492.