Proceedings of the Thirty-Third International Conference on Automated Planning and Scheduling (ICAPS 2023)

Understanding Natural Language in Context

Avichai Levy, Erez Karpas

Faculty of Data and Decision Sciences, Technion – Israel Institute of Technology
avichai@campus.technion.ac.il, karpase@technion.ac.il

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 assistant 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, 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. In this paper we describe an approach for translating natural language directives into the robot's formalism, allowing much more complicated household tasks to be completed. We do so by combining off-the-shelf SoTA large language models, planning tools, and the robot knowledge of the state of the world and of its own model. This results in much more accurate interpretation of directives in natural language.

Introduction

Virtual assistants such as Amazon Alexa, Google Assistant, Apple Siri, and others are becoming more and more common. These virtual assistants are able to understand commands and, for example, control smart home products and perform simple household tasks. However, these assistants are very limited in both their understanding of the world around them, and in the actions they can perform. Importantly, these assistants are static – they do not move around the house, and they can not physically manipulate the world around them.

We are concerned with more advanced versions of these assistants, which might be installed on mobile manipulators – robots which drive around our homes, and can perform many various actions. We would like these robots to be able to follow directives given by human users and perform much more complicated tasks. We take the view that such robots must already be *cognitive* (Levesque and Lakemeyer 2008) – that is, they must have some internal model of the world and of their actions, and they must have some formalism to represent and reason about knowledge. This allows such robots to understand what they are sensing and to come up with long term courses of action.

In this paper, we describe a system which can take directives given to such a cognitive robot in natural language, and translate them into a formalism the robot can reason about and plan with. Our system contains two innovations. First, it combines state-of-the-art large language models, which translate the natural language directives into a formal language, with classical planning tools which validate and and verify the output from the language model. Second, it incorporates the state of the world (that is, the context in which the robot is situated) into the translation process.

To train our system for a specific application, we require examples which consist of (a) a high-level robot trajectory (that is, a sequence of high-level actions performed by the robot, together with the initial state of the world), (b) a natural language description of these actions, and (c) a natural language description of the goal this plan achieves. We start with a pretrained large language model, and fine tune it based on the given training examples. As the language model already has wide commonsense knowledge (for example, that a fork is a type of utensil), this allows our training to focus on the domain-specific aspects of the applications, and thus, as our empirical evaluation shows, our system achieves good accuracy after seeing only a relatively small number of training examples.

As is evident from the structure of the training examples we use, humans can give instructions at different levels of detail – they can give the robot a mission (a high-level goal), or they can give the robot more detailed instructions. Our system translates natural language directives into a formal representation which combines both a high-level goal and a plan template for the robot to follow. Of course, using off-the-shelf machine learning tools does not guarantee that both parts of the output will be consistent with each other. Therefore, we use automated planning to check that the output is consistent, and if the output is not consistent we try to find alternative predictions of the machine learning model which are consistent. We also use the automated planner to fill out some details in the detailed instructions, as humans rarely

give enough detail for robots to be able to follow their instructions directly.

Background

We now describe some necessary background to understand our paper. Our system relies on a pretrained large language model. These days, these are based on the Transformer neural network architecture (Vaswani et al. 2017).

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. Transformers had great success in natural language processing (NLP) tasks, and most of the latest NLP models are based on this neural network architecture.

In this paper, we use two specific large language models: GPT-2 (Radford et al. 2019) and T5 (Raffel et al. 2019). Unlike other transformer models, GPT-2 is a decoder-only transformer model, consisting of 1.5 billion parameters. It was trained on a dataset of 8 million web pages. 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. Both GPT-2 and T5 are general-purpose learners, which can perform different tasks.

As previously mentioned, our system uses large language models to translate its input into a formal representation, and then uses automated planning tools to check that the output of the system is consistent. Specifically, we rely on the PDDL, the Planning Domain Definition Language (McDermott et al. 1998). While a complete specification of PDDL is beyond the scope of this paper, we highlight the aspects of it which are important for this paper.

In PDDL, a planning task is described by a domain, which will typically be the same for a given application, and a problem, which varies according to the specific planning task at hand. The domain description specifies the possible object types and the possible relations between them (called predicates in PDDL). The domain also describes the possible actions that the robot can execute. Each action is parameterized by some objects, and has some preconditions and effects. We adopt the view that any cognitive robot (Levesque and Lakemeyer 2008) should have some domain model regardless of its ability to understand natural language, and thus assume such a domain is available (or, at least, that the effort required to develop this domain should be amortized among different components of the cognitive robot).

On the other hand, the PDDL problem typically changes every time we want to generate a new plan for the robot. The problem describes the set of objects in the world as well as the relations between them in the current state – known as the initial state. The problem also describes the goal we would like to achieve at the end. Our system assumes there is some perfect state estimator which gives us the initial state, and translates natural language directives into the goal, as well as into some constraints on the plan, as we explain in the next section.

Translation System

We now describe our system, which is illustrated in Figure 1, in more detail. First, we assume that we already have a cognitive robot, with a domain theory that describes the types of objects in the world, the possible relations between them, and the possible actions that the robot can execute – that is, a PDDL domain. We also assume that our robot is clairvoyant, and has complete and perfect information about the state of the world, including which objects exists in the world and the relations between them.

Clearly, this assumption is not realistic, as the information about the state of the world is typically obtained from vision, which could be imprecise, and might require the robot to perform some actions to actively gather information. However, as our focus in this paper is on translating natural language instructions, we assume our robot is clairvoyant, and leave incorporating more realistic models of sensing to future work.

The input to our translation system consists of two different modalities. First, the robot is given a directive in natural language by a human, which instructs it to perform a particular task. As previously mentioned, 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.". We call this part of the input the "directive". Second, the robot know the state of the world, including the objects that exist in the world and the relations between them. We call this part of the input the "state".

Ultimately, the output of our system is a robot plan – a sequence of high-level actions for the robot to follow. This plan should achieve the high-level task that the human gave the robot, in a way that is consistent with the detailed instructions the human gave. To compute the plan, our system has two phases. In the first phase, we feed both parts of the input into a large language model. This language model outputs two types of information: a *goal* which corresponds to the high-level task, and a *plan template* which corresponds to the detailed instructions. Both of these are encoded in PDDL and, together with the information about the state of the world, create a PDDL problem.

In the second phase, we call a PDDL planner on the generated problem (using the PDDL domain the robot already

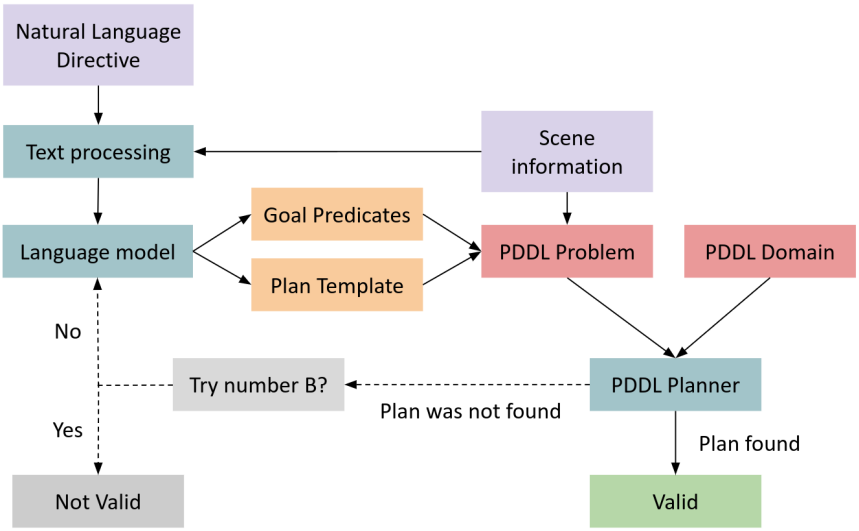

Figure 1: Diagram of Translation System

has), to find a plan. If a valid plan is found, we can output it. However, it could be the case that the prediction of the language model is not consistent, in which case the planner will not find a valid plan. In this case, we repeat this process using alternate (lower confidence) predictions of the language model, until either one of these returns a valid plan, or until some predefined number of tries B passes, in which case we do not return a plan.

Of course, the user has the option to only specify only a goal or only a plan template. If the user specifies a goal only, the planner is simply called with the goal that our system outputs, with no constraints on the plan. If the user specifies a plan template only, the planner is called with an empty goal, but must still obey the constraints from the plan template. We now explain how each of these phases work in more detail.

Phase I: From Natural Language to PDDL

In the first phase of our translation system we use a large language model to translate the input: a natural language directive together with state information, into the output we require: a predicted goal and plan template. As previously mentioned, we start with a pretrained large language model, which already contains commonsense knowledge. Thus, when we train the language model, we only need to fine tune it for the specific domain at hand. As our empirical evaluation will show, we require a small number of training examples to achieve very good accuracy.

As in earlier work, we model this translation process as a sequence-to-sequence task, and use on transformers (Vaswani et al. 2017) to solve it. Specifically, we used two language models, GPT-2 (Radford et al. 2019) and T5 (Raffel et al. 2019). Both of these are designed to handle textual input, so we can feed the natural language directive into these directly. To feed the state information into the transformer, we encode each fact in the state of the world as text. For example, $(\text{on } A\ B)$ would be encoded as "A on B", which the transformer can then process.

As we want to predict two types of output, a goal and a plan template, our transformer consists of two translation channels, which both take the same input mentioned above. With GPT-2, we trained separate models – one for each such channel. 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 channels.

Predicting Goals The *goal* channel captures the user's desired outcome of a task by describing the state of the world at the end of the plan's execution ("sliced tomato", "cold tomato"). One important point to note is that humans are unlikely to name specific objects in their natural language directive, as there is no agreed upon naming convention for objects in the real world. Therefore, when the goal includes some desired state of an object type, we accept any plan that reaches this state any instance of this object type.

For example, assume the user's directive is "put a sliced tomato on the countertop", and there are two tomatoes in the kitchen (assume the robot identifies these as tomato1 and tomato2, although the human does not know which is which). The human user does not care which tomato the robot uses, and thus out language model translates the goal into *"sliced tomato, on tomato countertop"*, without committing to a specific object.

Recall that the next step in our system is to encode this goal in PDDL. We do this by using existentially quantified goals, and thus we would translate the above goal into:

```
(exists (?tomato0 - tomato
         ?countertop0 - countertop)
    (and (sliced ?tomato0)
         (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.

Predicting Plan Templates The *plan template* channel outputs a sequence of actions that follows the detailed instructions in the input. As above, humans do not typically name specific objects, and thus an action occurrence in a plan template consists of an action type and parameter types. For example, a plan template could be "go to dining table, pick up apple dining table, go to fridge, put apple fridge". As there might be multiple apples and tables in our scene, this can not be translated directly into a sequence of actions. Therefore, we encode these into PDDL, and allow the planner to choose which objects to use for each action.

Our PDDL encoding enforces three types of constraints, according to the plan template: length of the plan, action type, and parameter type. These are encoded as follows:

- **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 planner to generate only plans of length T (where T is the number of actions in the predicted plan template).
- **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, forcing the planner to generate plans whose i'th action type is consistent with the plan template.
- **Object allowance** - This predicate is similar to the previous one. The predicate is $(allowed\_object_j\ objtype\ s_i)$, which indicates that the j'th parameter of i'th action in the plan can be of type $objtype$.

To demonstrate this, consider the following plan template: *"go to table, pick up apple table"* would be encoded in PDDL as follows:

- **Length** - since the length of the plan template 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. - $(allowed\_arg_1\ table\ \ s_0)$, $(allowed\_arg_1\ apple\ s_1)$, $(allowed\_arg_2\ table\ s_1)$.

Implementation Details 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 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 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 or the plan template.

Phase II: PDDL Consistency Checking

After generating the PDDL encoding for the predicted goal and plan template, we combine it with the initial state to generate a PDDL problem. We then use the robot's PDDL domain model, and call an automated planner to solve this planning task. The solution is thus constrained to achieve the predicted goal, while conforming to the constraints given by the plan template. If the planner finds a solution, it means the predicted goal and predicted plan template are consistent with each other, which gives us higher confidence that these predictions are correct. Thus, we output this plan for the robot to follow.

If the planner does not find a solution, it does not mean all is lost. Instead, we go back to our language model and "ask" it to generate another plan template. This is done by taking the next prediction in the model's beam search output. We then check if this new prediction is consistent (by calling the planner again). We repeat this procedure until a valid plan is found, or until the number of iterations exceeds a given number B – in which case we fail and do not return any plan. Note that w keep the same goal prediction throughout this process, as our goal prediction accuracy is typically higher than the plan template accuracy. However, in future work we could exploit the confidence of the goal and plan template predictions, to choose which one to keep and which one to change.

Implementation Details

We used the FF planner (Hoffmann and Nebel 2001), although any automated planner can be used. Interestingly, FF was used before for natural language generation (Koller and Hoffmann 2010). Additionally, we set the maximum number of predictions to test, B, to be 5.

Empirical Evaluation

Although the system we described above is general, there are very few datasets which can be used to evaluate it. Therefore, we train and evaluate our system on the ALFRED (Action Learning from Realistic Environments and Directives) dataset (Shridhar et al. 2020), which is built on top of the AI2-Thor simulator (Kolve et al. 2017). AI2-Thor simulates

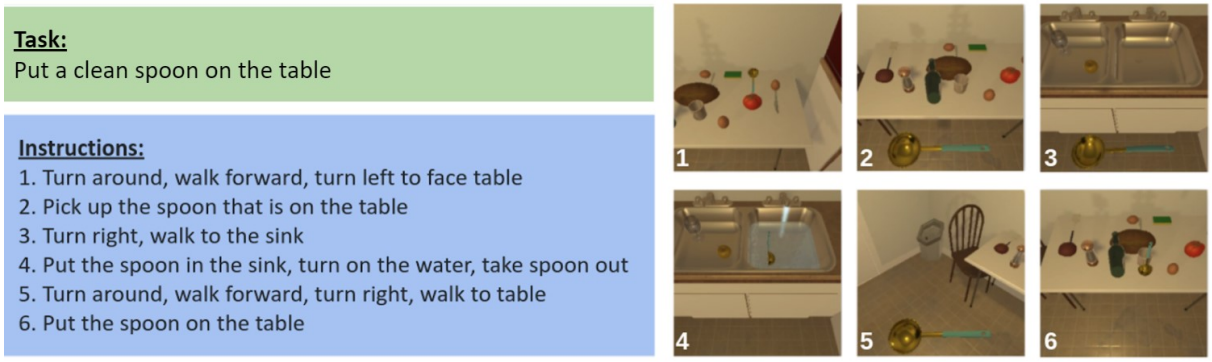

Figure 2: An example from the ALFRED dataset. The green text box contains the high level task and the blue text box contains the high level instructions needed for accomplishing that task. The images represent the egocentric vision input of the agent at each time step.

a robot which can perform high-level actions in a home environment. Some of these actions are irreversible (for example, slicing a tomato), which presents a challenge as misunderstanding an instruction can lead to a dead-end.

All of our code and data is available as supplementary material, and will be made publicly available once the paper is accepted.

Experimental Setup

The ALFRED dataset consists of 8,055 samples in which the robot executes a plan (a sequence of high-level actions). For each of these, a video of the robot's egocentric visual observations was shown to mechanical turkers, who were asked to formulate in natural language the high-level goal (called task in ALFRED terms) as well as the plan (called instructions in ALFRED terms). Figure 2 shows one example from this dataset. As each sample was annotated by multiple turkers, there are 25,743 examples in total.

Domain Model Examples in ALFRED come from 7 different task types, which take place in 120 different scenes. The tasks are Pick & Place, Stack & Place, Pick Two & Place, Clean & Place, Heat & Place, Cool & Place and Examine in Light. These involve 8 types of actions that the robot can execute: $pickup$, put, $slice$, $heat$, $cool$, $clean$, $toggle$ and $goto$ and 84 different object types.

We modeled the domain in PDDL, and used the following predicates to describe the state of the world: $robot\_has\_obj$, on, $sliced$, hot, $cold$, $cleaned$, $toggled$ and can_reach. One specific detail is that we created a special predicate (two_task) which is an indicator for the "pick two" task. For example, the natural language instruction "pick up two apples" would be translated by our language model into "pick up apple, two_task", which is then encoded into PDDL which forces the planner to pick up two different apples:

```
(exists (?apple0 - apple
         ?apple1 - apple)
    (and (not (= ?apple0 ?apple1))
         (robot_has_obj ?apple0)
         (robot_has_obj ?apple1)))
```

Pre-Processing In our work, we train language models to predict both the goal, which expresses the desired final state of each object as derived from the language directive, and the plan template. To create the targets for the language models, we focused on the PDDL parameters and the high-level actions provided by the ALFRED samples. The evaluation data in ALFRED is divided into validation and test datasets, where the labels of ALFRED's test set are hidden (as it is still an ongoing challenge). Each of these 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.

As we require the ground truth labels to evaluate our system, we evaluate only on ALFRED's training and validation sets, which we split into our own 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, which we term *test_unseen*. We might encounter duplicates between our datasets since we ignore the vision part of the data. Hence, the training and validation data are cleaned by removing duplicate samples with the same language directive as our test data.

Data Validation A data sample will be considered *valid* for training if its original action sequence solves the sample's problem. To obtain the initial state for the problem, 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 retrieve 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 for each problem was generated from the "PDDL parameters" field of the 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 solves the PDDL problem of this sample.

Model	Input	Consistency
GPT-2	Directive	0.69
	State	0.00
	Directive + State	0.83
T5	Directive	0.77
	State	0.59
	Directive + State	**0.97**

Table 1: Consistency Results

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.

Results

In this section we present the results of our empirical evaluation on ALFERD's val_unseen set. Due to the fact that these instances contain novel environments and objects, they provide the most accurate representation of the model's performance and generalization. We present results for both the GPT-2 and T5 large language models, with three different input modalities (natural language directive only, state only, and both).

Consistency As a first sanity check, we would like to measure how often our translation system produces consistent plans. Thus, we predict both the goal (from the task description), and the plan template (from the detailed instructions), and measure how often the plan that is produced by our system achieves the goal it predicted. Note that this does not mean the goal or the plan have anything to do with what the user said, merely that the output is consistent. Recall that our system guarantees that any plan that is output is consistent, and so the failures here mean that the first B plan templates produced by the language model were not consistent with the goal.

Table 1 presents the results for this measure. As the results show, using T5 with the directive and the state results in consistent plans 97% of the time. Not surprisingly, using GPT-2, a less powerful language model, yields less accurate results. Also of note is the observation that GPT-2 is unable to produce any consistent plans with the state only, while T5 is able to produce consistent plans from the state only 59% of the time. This suggests that T5 might predict easier goals when the input is non-informative.

Goal Prediction Next, we present the accuracy for our system's goal prediction. The models' accuracy scores on the goal prediction task are shown in Table 2.

For each goal prediction, we measure accuracy at several different levels, to be compatible with previous work (Jansen 2020). First, for each predicted fact in the goal we check whether the *predicate* is correct – that is, the predicate name only, without regard to the other arguments. We also check whether the first argument (*arg1*) and the second argument (*arg2*, if it exists) are correct, both of these independently of the other measures. Finally, we also check whether the entire goal *fact* (that is, the predicate and all arguments are correct).

Additionally, we check whether the entire *goal* is correct. As the goal is an unordered set of facts, we do not care about the order here, but simply check whether the predicted set of goal facts is correct. We also measure these using *permissive scoring* (Jansen 2020), which accepts predictions of objects that are similar to the original ones, e.g., "lamp - floor lamp", "knife - butter knife". These are indicated with $_p$.

As expected, accuracy based on state alone is almost 0 for both GPT-2 and T5. When using the natural language directive alone, without taking the context (that is, the state) into account, the better model (T5) is able to achieve 78% accuracy. However, incorporating the context into the model's input (Directive + State, denoted by D+S) increases accuracy to 85%, showing that there is relevant information for the goal prediction task in the state information. The results from adding the context for GPT-2 are very similar, albeit with lower accuracy. These findings suggest that encoder-decoder architectures might work better for goal prediction.

Table 3 shows some examples we selected, which show interesting mistakes of our system to predict the correct goals. These directives are different from the common tasks of ALFRED, and their intention is to check the robustness of the model. These examples show that the model displays some commonsense knowledge, such as that a potato is a vegetable, which comes from the pretrained language model.

Plan Templates Table 4 contains the models' scores on the *plan template* task. As for the goal prediction task, we measure accuracy at several levels. For each predicted action in the plan template, we check if the operator type (op) is correct, as well as if the arguments (*arg1* and *arg2*) are correct. We also check whether the entire *action* is correct. We also check whether the entire *plan* template is correct, that is, that all actions are correct, and (unlike for the goal prediction) in the correct order.

Similarly to the goal prediction task, both models achieve the highest score on the directive + state (D+S) input, showing that context is important for generating plans as well. In fact, the accuracy of the T5 model improves by 15% from including the context, showing that it is arguably even more important for generating plans than for predicting goals.

On the directive only input, GPT-2 predicts correctly 32% of full original action sequences. Previous work (Jansen 2020) which also used GPT-2 on the directive only achieved 22% accuracy, but the difference might be due to the training dataset changes. In any case, in this paper we improve upon the previous work by (a) incorporating context (state) into the input, (b) validating the resulting plans, and (c) using T5 instead of GPT-2. These changes improve accuracy to 57% – a significant improvement over the baseline of 32%.

Table 5 breaks down per-action accuracy scores by operator type. As the results indicate, some operator types are harder to predict accurately than others. This is likely due to a combination of how often each operator type appears in

Model	Input	predicate	arg1	$arg1_p$	arg2	$arg2_p$	fact	$fact_p$	goal	$goal_p$
GPT-2	Directive	0.80	0.77	0.81	0.79	0.89	0.76	0.80	0.66	0.72
	State	0.02	0.01	0.01	0.02	0.02	0.02	0.02	0.01	0.01
	D+S	0.73	0.71	0.74	0.81	0.83	0.70	0.73	0.74	0.77
T5	Directive	0.89	0.86	0.89	0.84	0.85	0.85	0.89	0.78	0.84
	State	0.09	0.08	0.09	0.05	0.05	0.08	0.09	0.04	0.04
	D+S	**0.92**	**0.89**	**0.92**	**0.88**	**0.89**	**0.89**	**0.92**	**0.85**	**0.88**

Table 2: *Goal* prediction accuracy scores. A permissive measure is indicated by the subscript p.

Input: Put a **baking tool** on the counter
Output: on **spatula** pan, on pan countertop
Input: Place two **vegetables** in the drawer
Output: on **potato** drawer, two_task
Input: Put any type of **cutlery** on the counter
Output: sliced **spoon**, on **spoon** cup, on cup countertop

Table 3: Goal prediction examples of our T5 model. The inputs are task descriptions and the outputs are the predicted goals

Model	Input	op	arg1	arg2	action	plan
GPT-2	Directive	**0.93**	0.75	0.67	0.63	0.32
	State	0.54	0.14	0.16	0.10	0.00
	D+S	**0.93**	0.78	0.74	0.69	0.46
T5	Directive	0.91	0.73	0.63	0.60	0.29
	State	0.68	0.22	0.26	0.18	0.04
	D+S	0.92	**0.82**	**0.76**	**0.75**	**0.57**

Table 4: *Plan Template* Accuracy Scores

the training data, together with the number of possible arguments for each operator type. For example, the pickup action can have many different object types in its arguments, while the argument for toggle is almost always a lamp.

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 plan templates and goals nearly as well as models that were trained on the full dataset. This means that instead of requiring more than 10K samples for training, several hundred are enough.

These results suggest that our model is able to perform successful few-shot learning. Of course, this result heavily relies on the knowledge that is already encoded in the pre-trained large language models. Thus, the training we perform for our system can be seen as fine tuning the large language models for the specific application (or domain) at hand.

Model	GoTo	Pickup	Put	Cool
GPT_D	0.68	0.40	0.68	**0.85**
$T5_D$	0.66	0.36	0.66	0.79
GPT_{DS}	0.75	0.56	0.67	0.78
$T5_{DS}$	**0.79**	**0.65**	**0.72**	0.83
Model	Heat	Clean	Slice	Toggle
GPT_D	0.82	**0.78**	0.39	0.75
$T5_D$	0.83	0.74	0.47	0.77
GPT_{DS}	0.80	0.75	**0.55**	0.75
$T5_{DS}$	**0.84**	**0.78**	**0.55**	**0.97**

Table 5: Action Accuracy by Operator Type. The subscript stands for Directive only input (D) or Directive + State input (DS).

Related Work

Many researchers have realized that 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 tasks is *visual semantic planning* (Zhu et al. 2017), which is the task of generating a sequence of high-level actions from visual observations. 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.

Recent papers have chosen to break this problem down into separate modalities instead of solving this difficult multi-modal problem. The most closely related work to ours (Jansen 2020) explored this task on the ALFRED dataset, by using the GPT-2 language model to generate plans from high-level task descriptions, without visual cues. In this work, GPT-2 was shown to outperform 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. As mentioned in the empirical evaluation, our results improve over these even further.

Other work has integrated Hierarchical Task Networks

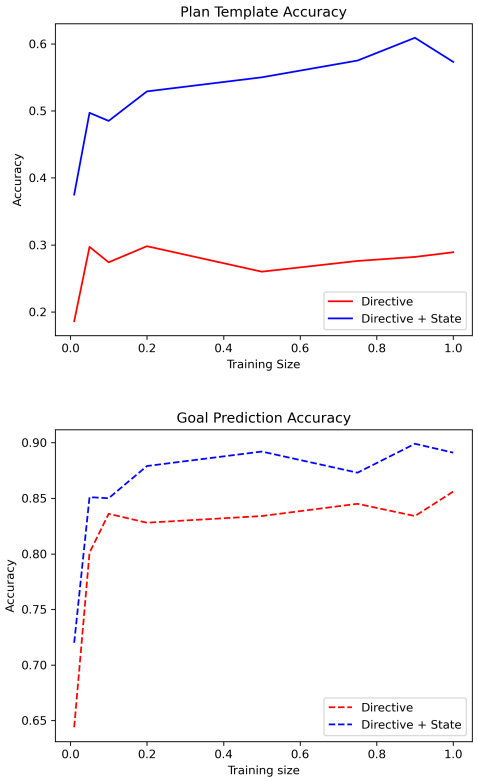

Figure 3: Few-shot Accuracy Scores

and Probabilistic Inference to generate action sequences using multiple context types, but without natural language directives (Wang, Tian, and Shao 2020). Later work 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 (Wang et al. 2021). 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.

Summary and Future Work

We have developed a novel approach for natural language understanding for commanding robots that are situated in the real world. The two main innovations we make in this paper are (a) incorporating the context (state of the world) into the input of our system, together with the natural language directive, and (b) incorporating automated planning into the system to verify that the plans it produces are valid. Combined with state-of-the-art pretrained large language models, our system achieves 85% accuracy in translating tasks expressed in natural language to goals, and 57% accuracy in following plans expressed in natural language on the ALFRED dataset.

In future work, we intend relax the assumption of having a clairvoyant robot. This would require integrating a computer vision module into our system, as well as incorporating partially observable planning with sensing capabilities into the planner. We believe the framework laid out in this paper for combining large language models with automated planning tools serves as a good starting point for this.

Additionally, we would like to explore the ability of natural language to express richer directives. For example, a natural language directive like "go to the bed room through the kitchen" combines a goal with a partial specification of the plan. Such a directive could be translated into a partial plan template, which could be expressed in PDDL similarly to the way our system works.

Humans can also use complex natural language expressions to refer to specific objects. For example "pick up the rightmost tomato on the kitchen countertop". In this case, we might use semantic parsing (Zettlemoyer and Collins 2005) to translate the object-reference to a query which can be run on the knowledge base representing the state of the world. Once this query is evaluated, its result can be incorporated into the planning process. Alternatively, the expression could be translated to PDDL directly, but this might require introducing new predicates, such as those expressing geometric relations between the objects, to be able to express "rightmost" in PDDL.

Another challenge is ensuring that the system follows common sense rules in both interpreting directives and in planning. For example, if the human asks to put a tomato on the table, then the robot should probably not slice the tomato before putting it on the able, even though in the PDDL representation the goal would be achieved by having a sliced tomato on the table. On the planning side, if the human asks for the robot to take a beer out of the refrigerator, the robot should probably close the refrigerator door after taking out the beer, even though the user did *not* ask for it. At an even higher level of difficulty, the robot need to understand that the directive "get me a beer" should not be translated to the goal of the user holding a beer, but rather to the goal of the user drinking a beer, which might also require getting a bottle opener and a glass.

We believe the key to addressing all of these challenges is in combining machine learning with knowledge representation, reasoning, and automated planning. This paper takes a small step in this direction.

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*.

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*.

Hoffmann, J.; and Nebel, B. 2001. The FF Planning System: Fast Plan Generation Through Heuristic Search. *J. Artif. Intell. Res.*, 14: 253–302.

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*.

Koller, A.; and Hoffmann, J. 2010. Waking Up a Sleeping Rabbit: On Natural-Language Sentence Generation with FF. In Brafman, R. I.; Geffner, H.; Hoffmann, J.; and Kautz, H. A., eds., *Proceedings of the 20th International Conference on Automated Planning and Scheduling, ICAPS 2010, Toronto, Ontario, Canada, May 12-16, 2010*, 238–241. AAAI.

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. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control.

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.

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.

Zettlemoyer, L. S.; and Collins, M. 2005. Learning to Map Sentences to Logical Form: Structured Classification with Probabilistic Categorial Grammars. In *UAI '05, Proceedings of the 21st Conference in Uncertainty in Artificial Intelligence, Edinburgh, Scotland, July 26-29, 2005*, 658–666. AUAI Press.

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.