Using Planning to Improve Semantic Parsing of Instructional Texts

Vanya Cohen and **Raymond Mooney**
Department of Computer Science, University of Texas at Austin
Austin, TX 78712, USA
{vanya, mooney}@utexas.edu

Abstract

We develop a symbolic planning-based decoder to improve the few-shot semantic parsing of instructional texts. The system takes long-form instructional texts as input and produces sequences of actions in a formal language that enable execution of the instructions. This task poses unique challenges since input texts may contain long context dependencies and ambiguous and domain-specific language. Valid semantic parses also require sequences of steps that constitute an executable plan. We build on recent progress in semantic parsing by leveraging large language models to learn parsers from small amounts of training data. During decoding, our method employs planning methods and domain information to rank and correct candidate parses. To validate our method, we evaluate on four domains: two household instruction-following domains and two cooking recipe interpretation domains. We present results for few-shot semantic parsing using leave-one-out cross-validation. We show that utilizing planning domain information improves the quality of generated plans. Through ablations we also explore the effects of our decoder design choices.

1 Introduction

Recent advancements in natural language processing (NLP) have successfully combined large language models with external symbolic reasoning capabilities. Toolformer (Schick et al., 2023) enables the use of external tools to perform tasks such as arithmetic and factual lookup, which are currently challenging for large language models. Liu et al. (2023) created a system that performs multi-step reasoning in planning tasks specified in natural language. This is achieved by using an external symbolic planner for domain-specific reasoning, providing guarantees that plans are logical and satisfy environmental constraints. By integrating external symbolic reasoning capabilities, many of the shortcomings of large language models can be addressed while still capitalizing on their strengths. Motivated by this line of work, we develop a novel decoding procedure that uses symbolic planning to improve the few-shot semantic parsing of long-form instructional texts. We map instructional texts to formal action sequences and validate our method on two recipe semantic-parsing datasets (Bollini et al., 2013; Tasse and Smith, 2008). Semantic parsing of instructional texts with few-shot learning poses several challenges to natural language processing (NLP) techniques. Current NLP methods are dominated by large language models. These are based on Transformer (Vaswani et al., 2017) architectures and leverage large scale pretraining to enable high, few-shot and zero-shot task performance (Brown et al., 2020).

Building on work from (Shin et al., 2021), we investigate semantic parsing in the few-shot setting using OpenAI's Codex language and code LLM (Chen et al., 2021; Shin and Van Durme, 2021). Learning occurs in-context, by prompting the LLM with a few input-output task examples. Pretrained LLM representations allow for more sample efficient learning than non-pretrained methods; but data scarcity still introduces performance limitations (Brown et al., 2020). Data efficiency is advantageous when working with long-form instructional texts. The datasets we consider are small and the cost of annotating long texts with ground-truth semantic parses is high.

In many semantic parsing tasks, context dependencies, input natural language strings, and output parses are relatively short. These tasks fit easily within the available context size of LLM models and consist of at most several input and output statements. Modeling long input-output dependencies poses a number of challenges for current models, as shown by their degraded performance on tasks designed to leverage long contexts (Tay et al., 2020). Semantic parsing of long-form instructional texts,

Proceedings of the 1st Workshop on Natural Language Reasoning and Structured Explanations (@ACL 2023), pages 47–58

like the recipe datasets we evaluate, can require learning representations for hundreds of words, and outputting tens of steps, each with multiple arguments and complex syntax.

Instructional texts also exist within an implicit planning domain. These texts describe plans for achieving a goal by manipulating objects that comprise a world-state. Executable semantic parse-plans must consist of valid transitions within this world-state. For example, to bake an item the oven must be on and preheated. These requirements constitute preconditions for the bake action. Long-context dependencies pose challenges to generating executable sequences of actions that are also relevant to the task instructions. To form a valid parse-plan, instructions must be translated into a sequence of executable actions. All requisite actions must be represented in the plan, potentially including actions not explicitly mentioned. Complicating matters, the common-sense knowledge needed to reason about valid plan sequences in a domain is only very implicitly represented within the LLM and few-shot examples.

To address these challenges, we propose *Planning Augmented Semantic Parsing*. Our method leverages a formal symbolic planning representation to rank and correct candidate plans. Plans are corrected by searching for sequences of actions that satisfy the preconditions of all output actions. Ranking selects plans which best meet the domain's planning and syntactic constraints by ranking plans highly if they have fewer inadmissible actions, require fewer additional actions to correct, and have fewer steps with invalid syntax. After ranking, planning errors are fixed using symbolic planning methods. The result is an effective neuro-symbolic approach that combines the strengths of deep-learned LLMs and classical AI planning.

We validate our approach using leave-one-out cross validation across each dataset and provide ablations for various aspects of our model choices. Results show that using *Planning Augmented Semantic Parsing* results in more valid plan sequences that still maintain high relevance to the natural language task instructions.

Overall we make the following contributions:

- Develop a novel method for using symbolic planning to improve semantic parsing with large language models.
- Demonstrate improvements in the executability of generated plans on two datasets, in a low-data, few-shot setting.

2 Background

2.1 Planning

We consider these instructional text semantic parses to be plans in a symbolic task planning setting. A task planning domain defines a world-state, actions that modify the world-state, and transition function specifying the effect of actions on the world-state (Ghallab et al., 2016). The world state is composed of a collection of Boolean values defining the existence and state of various kitchen objects and planning actions are implemented as STRIPs-style operators (Fikes and Nilsson, 1971). Each action has logical preconditions that must be satisfied for its execution. For any state, the admissible actions are all actions with satisfied preconditions. Upon execution the action changes the values of the variables which define the world-state. A planning task is specified by an initial state and a goal state. The resulting plan (if it exists) comprises a sequence of actions that can be taken to reach the goal state.

2.2 In-Context Learning

In-context learning allows LLMs to perform novel tasks specified in terms of a small number of sample input-output pairs (Brown et al., 2020). These examples are provided as part of the generation context to condition the language model. Typically these examples are drawn from the training split of a dataset and prepended to a test example. Few-shot prompted learning differs from other LLM learning paradigms including the zero-shot inference utilized in the evaluations of GPT-2 (Radford et al., 2019) and fine-tuning used to transfer pre-trained model weights to novel tasks as in (Radford et al., 2018), (Peters et al., 2018) and (Devlin et al., 2018).

3 Related Work

(Branavan et al., 2009) develops a reinforcement learning-based method for semantic parsing of instructional texts and (Branavan et al., 2010) additionally learns to fill in low-level steps from high-level instructions using environment interaction. Previous work also formulates semantic parsing as a text-to-text machine translation task (Wong and Mooney, 2006). Our work builds on the few-shot semantic parsing of (Shin et al., 2021) and (Shin and Van Durme, 2021) that establishes OpenAI's

Codex model (Chen et al., 2021) as a high performing LLM for few-shot semantic parsing. (Bollini et al., 2013) and (Tasse and Smith, 2008) introduce the recipe-semantic parsing datasets we use for evaluation but learn semantic parsers using shallow features and classification-based approaches. Other work investigates semantic parsing of recipes (Malmaud et al., 2014) including using modern deep-learning methods (Papadopoulos et al., 2022). Recent work uses few-shot learning and large language models to map single commands (Huang et al., 2022) and short sequences of commands (Brohan et al., 2022) to executable plans. However, to our knowledge, ours is the first work to use symbolic planning to improve semantic parsing of instructional texts.

4 Methods

For the LLM we use Davinci Codex (Chen et al., 2021) based on the GPT-3 architecture (Brown et al., 2020). Like GPT-3, the model was trained on a large web-sourced text corpus, but includes code in the dataset. While OpenAI does not publish the sizes of the Codex models available through their API, (Gao, 2021) empirically estimate the size of text-only Davinci at 175B parameters.

4.1 Datasets

We evaluate our method on two recipe semantic parsing datasets from (Tasse and Smith, 2008) and (Bollini et al., 2013). (Tasse and Smith, 2008) contains 260 recipes with corresponding semantic parses in the Minimal Instruction Language for the Kitchen (MILK) syntax. Each statement in the language corresponds to a plan step with an action and arguments. Some plan steps produce new variables (ingredients or tools) which are consumed by subsequent steps. The recipes in this dataset cover a wide range of cuisines, ingredients, techniques, and tools. Each step also contains an optional human-readable description of the step. The original dataset uses variables as arguments to action steps. We replace these with their literal values to make the generation problem easier for the LLM. The 60 recipes of (Bollini et al., 2013) were selected to be executed by a cooking robot. The recipes are mainly limited to baking and contain a small fixed set of tools and actions. This dataset also contains planning domain definitions in the form of STRIPs-style operator actions (Fikes and Nilsson, 1971). Recipe steps that require tools or techniques outside of this fixed vocabulary are mapped to a NO-OP. Examples from both datasets are in the Appendix: with in Table 5 and action signatures are defined in Table 6.

4.2 Planning Domain

(Bollini et al., 2013) contains planning domain definitions that specify the state of the kitchen, ingredients, and tools used in each recipe. These were developed to facilitate recipe execution on a real-world cooking robot. We utilize these definitions for planning in this domain. The domain also provides a successor state function that given a starting state and a search depth, returns all valid sequences of actions up to the search depth. This is used in Algorithm 1. The (Tasse and Smith, 2008) dataset does not provide planning definitions. We construct planning definitions where the existence of ingredients and tools are the only predicates and transitions in the environment involve either creating or destroying these objects. Therefore the only preconditions in this domain involve the existence of objects. Actions are considered valid if their objects have been instantiated by prior steps, otherwise the their preconditions are considered unsatisfied.

4.3 Prompt Design

To generate a plan for a test example using few-shot learning, we prompt the model with sample recipes and parses taken from the held-out examples. Following (Liu et al., 2021), prompt examples are selected using nearest-neighbor search using the cosine distance between their embeddings as computed by a text embedding model, specifically the "all-mpnet-base-v2" model from the SentenceTransformers library (Reimers and Gurevych, 2019) based on the MPNet model (Song et al., 2020). Due to the limited length of the input context for the Codex models ($8,000$ and $2,048$ tokens) and API request limits, the number of training examples is limited to a maximum of five for the recipes of (Bollini et al., 2013) and one for (Tasse and Smith, 2008). The selection of few-shot training examples reduces to Equation 1, where a is a training example, X denotes the set of held-out examples, and E represents the recipe embedding function.

$$P(a) = \arg\min_{x \in X} \{cos\_sim(E(x), E(a))\} \quad (1)$$

The full prompt is formed by concatenating the training example instructions with their semantic parses, and the instructions for the test example. Each component of the prompt is separated by new line characters, and a special delimiter "###" which is also appended to the end of the prompt. Because of the in-context learning, the model learns to append the delimiter to the end of the generated parse. Thereby, the delimiter is used to identify the end of the model's sequence completion.

4.4 Planning-Augmented Decoding

Given a prompt sequence, the LLM defines a distribution over next token continuations. Due to utilization limits of the Codex API, our method samples a fixed number of plan completions for each recipe. For sampling next tokens, we use nucleus sampling introduced by (Holtzman et al., 2020), which offers improvements over other sampling methods. These are then ranked using a scoring function based on the generation probability and a *planning score* which factors in the number of precondition errors and syntax errors in the plan and the sequence probability of the plan.

The planning score is calculated by combining measures of plan executabilty: the number of precondition errors, syntax errors, and additional planning steps. Precondition errors occur when a plan step's preconditions are not satisfied. For example when the step references non-existent ingredients or the world state does not allow the action to occur. Syntax errors occur when the plan contains malformed steps that cannot be parsed by the plan interpreter. The syntax error score (SE) is the number of plan steps which contain syntax violations. The precondition error score (PE) is the number of plan steps which cannot be executed because their preconditions are not satisfied. Finally, (AS) is the number of steps added to the plan by planning in order to maximize the number of plan steps with valid preconditions. Steps with errors are counted as opposed to counting all errors in each step. This allows for computation of a score in the interval $[0, 1]$ to match the sequence probability score. In general identifying multiple syntax errors in a given step is not possible, as the presence of even a single syntax error may result in an undefined grammatical context.

These counts are normalized by the plan length (N) so as not to penalize longer plans. The natural log is taken to re-scale the planning score for addition with the sequence log-probability, adding ϵ to avoid taking the logarithm of zero in cases where the plan contains no errors and requires no additional steps to be valid. The planning score is added to the mean log-probability of the token sequence representing the plan. This scoring function results in plans with a higher sequence probability and fewer planning errors being selected.

$$score = \ln(1.0 - \frac{SE + PE + AS}{N} + \epsilon) + \frac{1}{T}\sum_{t=1}^{T} \ln P_t \quad (2)$$

The plan that maximizes the $score$ function is passed to a planning module. For each inadmissible step where the preconditions are not satisfied, it searches for sequences of admissible actions to insert into the plan, such that those actions lead to valid preconditions for that step. To limit the search space, the planning module only searches in the space of plans that can be inserted before an existing inadmissible plan step.

4.5 Correcting Plans

While the ranking procedure ensures that high probability and low-error plans will be surfaced, these plans may still contain precondition errors. The planning domain information and a planning algorithm together form a planning module that can attempt to correct these precondition errors. The ranking procedure incorporates this planning module to calculate the additional steps AS that can be inserted into a plan to fix precondition errors. These steps are also included in the total number of steps N. Therefore fixable plans will receive a better planning score.

Algorithm 1 describes the procedure for finding steps to insert into a plan to ensure that each step's preconditions are satisfied. As input, it takes the world state after executing some number of plan steps, and computes the sequence of actions needed to ensure the preconditions of the next plan step A are met. The algorithm returns the shortest number of actions required to satisfy the step's preconditions. Aside from producing more valid plans, this method should produce plans which correspond more closely to the ground truth semantic parse annotations. However because the planning module only inserts steps into a plan before an inadmissible step, and does not change the existing steps,

Algorithm 1: The planning algorithm inserts steps before actions with unmet preconditions.

Input: A starting state S, desired action A, transition function T, and search depth N.
Output: The shortest sequence of actions P which ends in action A or $null$ if none exists.

```
sequences = successors(S, T, depth=N);
best = null;
for s in sequences do
    if A in s then
        /* truncate the plan until
           the action A or false if
           no such prefix exists   */
        plan = prefix(s, A);
        if plan && length(plan) <
          length(best) then
            best = plan;
        end
    end
end
```

it cannot necessarily fix all precondition errors in a plan. Utilizing planning and likelihood based decoding balances the desire for plans with valid preconditions while ensuring that plans contain relevant steps to the recipe. These two requirements may compete in some cases. In the simplest case, an empty plan, there are no potential precondition errors, but the plan also contains no relevant plan steps. In practice there exists a trade-off between plan executability and correctness as noted by (Huang et al., 2022).

5 Experiments

We use leave-one-out cross-validation to evaluate performance of various models on our two recipe datasets. For the shorter recipes in the (Bollini et al., 2013) dataset, we evaluate using both one and five training plans in the prompt. We evaluate the longer recipes of the (Tasse and Smith, 2008) dataset using only a single prompt example due to context length limitations.

To evaluate the correctness of each output plan we compare the generated plan to the ground truth annotation from the dataset. We use metrics that measure the similarity between the output and ground truth plans. However for each recipe there are potentially many admissible plans and subjective judgements about the level of detail of the annotation and about which attributes to include. To address these potential ambiguities, we evaluate models using several diverse metrics to capture different aspects of plan accuracy.

5.1 Baselines

We evaluate three baseline methods for ranking the generated plans: *Random*, *Rank (PPL)*, and *Rank*. Our full ranking method with partial planning is denoted *Rank+Plan*. *No Rank* simply selects a random plan from the set of generated completions. *Rank (PPL)* selects the plan with the lowest perplexity (PPL) (the highest sequence probability), providing a baseline where no planning domain information is utilized. Finally, *Rank* ranks the plans by the scoring function in Equation 2, but does not correct precondition errors through planning like *Rank + Plan* does.

5.2 Longest Common Subsequence (LCS)

Prior work evaluates plan correctness in terms of the LCS between generated and ground truth plans (Puig et al., 2018). We normalized LCS by the length of the longer plan. LCS evaluates the textual overlap between plans; computing common subsequences which may contain interwoven unequal sequences. It therefore does not strongly penalize erroneous injected subsequences. This metric ranges from $[0.0, 1.0]$ where 0.0 indicates no sequence overlap and 1.0 indicates identical plans.

5.3 Plan Steps F1

LCS reflects the order and content of the generated plan steps compared to the ground truth. We also report an F1 measure (the harmonic mean of precision and recall) that quantifies the quality of the individual plan steps without regard to their sequencing. Steps in the generated and ground truth plans are compared based on string equality. In many plans, steps are often repeated. For example a recipe from (Bollini et al., 2013) may have many $mix()$ steps after pouring different ingredients. We choose to treat each of these repetitions as a unique step when computing the precision and recall. We also exclude NO-OP steps from these calculations for the (Bollini et al., 2013) dataset as they do not change the plan's results.

(Bollini et al., 2013)		
Rank	**Rank + Plan**	**Ground Truth**
pour(nuts) mix() scrape() bake(25)	pour(nuts) mix() scrape() **preheat(350)** bake(25)	pour(nuts) mix() scrape() **preheat(350)** bake(25)
(Tasse and Smith, 2008)		
Rank	**Rank + Plan**	**Ground Truth**
combine("1/2 cup all-purpose flour", "1 egg", "1 tablespoon chopped fresh parsley", "1/2 teaspoon salt", "1/4 teaspoon freshly ground nutmeg", "mashed potatoes", "dough", "stir in")	**create_ing("1/2 cup all-purpose flour")** ... combine(**"1/2 cup all-purpose flour"**, "1 egg", "1 tablespoon chopped fresh parsley", "1/2 teaspoon salt", "1/4 teaspoon freshly ground nutmeg", "mashed potatoes", "dough", "stir in")	**create_ing("1/2 cup all-purpose flour")** ... combine(**"1/2 cup all-purpose flour"**, "1 egg", "1 tablespoon chopped fresh parsley", "1/2 teaspoon salt", "1/4 teaspoon freshly ground nutmeg", "mashed potatoes", "mashed potato mixture", "stir in")

Table 1: Excerpted examples of improved parsing for recipes using the *Rank + Plan* method. The parses were selected by randomly selecting recipe where the *Rank + Plan* method resulted in an improvement in the number of precondition errors over the baseline methods. NO-OP actions are omitted for brevity.

5.4 Precondition and Syntax Errors (PE & SE)

The previous metrics assess similarity to the ground truth plan and do not explicitly reflect the executability of generated plans. Therefore, we also measures the frequency of precondition and syntax errors in plans. Errors are counted on a per-step basis. If a step contains more than one error or more than one type of error these are quantified as a single error and type for the step.

5.5 Implementation Details

We utilize Davinci Codex[1] for all experiments due to its large context size of $8,000$ tokens which is sufficient for all prompts and completions across both datasets. Our method samples ten completions up to a fixed length of $1,500$ tokens or until the special delimiter sequence is reached. The decoding length was chosen to be longer than the longest recipe parse in either of the datasets. We generate ten completions for each recipe to offer a diversity of plans to rank and correct through planning. We also utilize a nucleus sampling top-p value of 0.5. This value was selected because it maximizes the performance of the *No Rank* baseline with respect to LCS. We perform ten trials to compute means and confidence intervals.

[1]The OpenAI API name for the model is "code-davinci-002".

6 Results

We report results for the (Bollini et al., 2013) dataset in Table 2 and the (Tasse and Smith, 2008) dataset in Table 3. Mean cross-validation results are reported with 95% confidence intervals computed using a t-distribution for ten trials. For both datasets, the number of precondition errors are reduced by ranking using our scoring metric and corrective planning. The *Rank* method results in a decrease in the precondition error rate, and by adding corrective planning (*Rank + Plan*), the error rate is again reduced significantly. This results in more valid, executable plans. Even as the precondition error rate is reduced, the LCS remains constant for the (Bollini et al., 2013) dataset and only slightly reduced for the (Tasse and Smith, 2008) dataset. This indicates that the plans maintain high agreement with the ground truth plans while steps are

Models	(Bollini et al., 2013)			
	LCS↑	PE↓	SE↓	F1 ↑
No Rank				
Davinci Codex, E=1	0.908 ± 0.007	0.737 ± 0.067	0.065 ± 0.025	0.784 ± 0.002
Davinci Codex, E=5	0.950 ± 0.001	0.277 ± 0.020	0.000 ± 0.000	0.859 ± 0.002
Rank (PPL)				
Davinci Codex, E=1	0.897 ± 0.008	0.962 ± 0.685	0.042 ± 0.008	0.784 ± 0.004
Davinci Codex, E=5	0.949 ± 0.005	0.198 ± 0.009	0.002 ± 0.004	0.863 ± 0.003
Rank				
Davinci Codex, E=1	0.901 ± 0.008	0.382 ± 0.037	0.025 ± 0.008	0.798 ± 0.002
Davinci Codex, E=5	0.952 ± 0.005	0.120 ± 0.015	0.002 ± 0.004	$\mathbf{0.868 \pm 0.002}$
Rank + Plan				
Davinci Codex, E=1	0.903 ± 0.008	0.143 ± 0.033	0.025 ± 0.008	0.807 ± 0.002
Davinci Codex, E=5	0.952 ± 0.005	$\mathbf{0.033 \pm 0.000}$	0.002 ± 0.004	$\mathbf{0.870 \pm 0.002}$

Table 2: Results and ablations for the (Bollini et al., 2013) dataset, reported as means over the leave-one-out cross validation. 95% confidence intervals are computed using a t-distribution over ten trials. Results using one (E=1) and five (E=5) training examples in each prompt are shown. All plans are generated using a nucleus sampling top-p value of 0.5.

Models	(Tasse and Smith, 2008)			
	LCS↑	PE↓	SE↓	F1 ↑
No Rank				
Davinci Codex, E=1	$\mathbf{0.707 \pm 0.002}$	0.805 ± 0.029	0.940 ± 0.134	$\mathbf{0.448 \pm 0.001}$
Rank (PPL)				
Davinci Codex, E=1	0.692 ± 0.003	0.827 ± 0.086	0.875 ± 0.199	0.443 ± 0.002
Rank				
Davinci Codex, E=1	0.695 ± 0.004	0.293 ± 0.016	$\mathbf{0.226 \pm 0.024}$	0.446 ± 0.001
Rank + Plan				
Davinci Codex, E=1	0.695 ± 0.003	$\mathbf{0.000 \pm 0.000}$	$\mathbf{0.237 \pm 0.018}$	0.446 ± 0.001

Table 3: Results and ablations for the (Tasse and Smith, 2008) dataset, reported as means over the leave-one-out cross validation. 95% confidence intervals are computed using a t-distribution over ten trials. Results using one (E=1) training example in each prompt are shown. All plans are generated using a nucleus sampling top-p value of 0.5.

added. For the (Bollini et al., 2013) dataset the F1 score also improves through ranking and corrective planning indicating that highly ranked plans contain more accurate selections of recipe steps. The F1 score does not improve for the (Tasse and Smith, 2008) dataset and the *Rank + Plan* method results in no precondition errors. However, these results are not surprising because the for this dataset the planning module can only insert ingredient and tool definitions into the plan. These inserted actions could result in lower LCS by lengthening the generated plan and may not match ingredient definitions in the ground truth plan. Because of the presence of free-text descriptions and specifications in the (Tasse and Smith, 2008) dataset and the difficulty of parsing longer plans, both the LCS and F1 are lower than for the (Bollini et al., 2013) dataset. Finally providing more in-context examples for the (Bollini et al., 2013) dataset improves performance for all measured metrics.

Table 1 contains qualitative examples of the *Rank + Plan* performance. For the example from the (Bollini et al., 2013) dataset, in the generated plan the oven is not preheated before baking. The corrected plan adds a preheat() action to satisfy the preconditions of the bake() action, which requires

a heated oven. In the example from the (Tasse and Smith, 2008) dataset, a recipe that uses certain ingredients to make dough. In the generated plan these ingredients are not instantiated. However the planning module inserts actions to instantiate the ingredients which improves the validity of the generated plan. An additional example for the (Bollini et al., 2013) dataset is included in Table 4

7 Discussion

Evaluations show that our method improves plan validity as measured by the mean number of precondition errors, syntax errors, and accuracy of steps returned (F1) in each plan. LCS remains fairly constant across our evaluations and ablations. The LCS metric reflects both the content of planning steps and their sequencing. By contrast, F1 only assesses the accuracy of steps in the generated plans. Perhaps there exists a trade-off wherein the process of inserting corrective plan steps reduces the amount of alignment of the generated and ground truth plans (lowering LCS), but increases the accuracy of included steps (raising F1). Of all the metrics considered, our method results in largest reduction in the number of precondition errors (PE). We achieve these improvements without singificant reductions in LCS and with an increase in F1. This is an important validation for our method, as (Huang et al., 2022) finds that there exists a trade-off between the executability and semantic correctness (measure by LCS) of generated plans. It is straightforward to increase executability (fewer precondition errors) by ignoring the instructional text content and only outputting valid actions. For any downstream applications, plans must be executable and while also reflecting the content of the instructions. Therefore is is important to reduce the number of precondition errors while maintaining content similarity to ground-truth plans.

7.1 Limitations and Future Work

Our approach requires access to planning information for each instructional text domain. In general, creating this information requires programming and domain knowledge to formally specify the planning constraints. However for high-value applications the effort associated with generating these planning domain definitions may be justified by their potential to help in generating more valid plan-based semantic parses. Having this knowledge is also crucial to allowing an agent or robot to execute the resulting plan and may be naturally available in many domains as part of the execution component. In the course of developing our semantic parsing model, we discovered that Codex could generate valid planning domain definitions in a variety of output formats including the Planning Domain Definition Language (Fox and Long, 2003). This may provide a path towards automatically generating planning domain definitions for novel environments or reducing the need for human annotators. Future work could also evaluate our method in other planning domains that contain tasks beyond cooking such as VirtualHome (Puig et al., 2018) or ALFRED (Shridhar et al., 2020).

8 Conclusion

We develop an approach to semantic parsing for long-form instructional texts that leverages planning domain information to generate more valid plans in a low-data, few-shot setting. Our method significantly reduces the number of precondition errors present in semantically parsed plans for two recipe datasets. These results highlight the benefit of a neuro-symbolic approach that utilizes the state-of-the-art code-generation LLM Codex to produce relevant steps for recipe execution and refines these plans using classical symbolic planning. In quantitative and qualitative evaluations, our approach generates plans that reflect the relevant steps of the natural language recipe. The symbolic planning component corrects precondition errors that arise from omitted or implied instructional steps and the challenges of learning with long context-dependencies from limited examples.

Acknowledgements

This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. HR001122C0007. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Defense Advanced Research Projects Agency (DARPA).

References

Mario Bollini, Stefanie Tellex, Tyler Thompson, Nicholas Roy, and Daniela Rus. 2013. Interpreting and executing recipes with a cooking robot. In *Experimental Robotics*, pages 481–495. Springer.

Satchuthananthavale RK Branavan, Harr Chen, Luke Zettlemoyer, and Regina Barzilay. 2009. Reinforcement learning for mapping instructions to actions. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 82–90.

SRK Branavan, Luke Zettlemoyer, and Regina Barzilay. 2010. Reading between the lines: Learning to map high-level instructions to commands. In *Proceedings of the 48th annual meeting of the association for computational linguistics*, pages 1268–1277.

Anthony Brohan, Yevgen Chebotar, Chelsea Finn, Karol Hausman, Alexander Herzog, Daniel Ho, Julian Ibarz, Alex Irpan, Eric Jang, Ryan Julian, et al. 2022. Do as i can, not as i say: Grounding language in robotic affordances. In *6th Annual Conference on Robot Learning*.

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

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

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

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

Maria Fox and Derek Long. 2003. Pddl2. 1: An extension to pddl for expressing temporal planning domains. *Journal of artificial intelligence research*, 20:61–124.

Leo Gao. 2021. On the sizes of openai api models. `https://blog.eleuther.ai/gpt3-model-sizes/`. Accessed: 2022-08-11.

Malik Ghallab, Dana Nau, and Paolo Traverso. 2016. *Automated planning and acting*. Cambridge University Press.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. 2020. The curious case of neural text degeneration. In *International Conference on Learning Representations*.

Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. 2022. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. *arXiv preprint arXiv:2201.07207*.

Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone. 2023. Llm+p: Empowering large language models with optimal planning proficiency.

Jiachang Liu, Dinghan Shen, Yizhe Zhang, Bill Dolan, Lawrence Carin, and Weizhu Chen. 2021. What makes good in-context examples for gpt-3? *arXiv preprint arXiv:2101.06804*.

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

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

Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*.

Xavier Puig, Kevin Ra, Marko Boben, Jiaman Li, Tingwu Wang, Sanja Fidler, and Antonio Torralba. 2018. Virtualhome: Simulating household activities via programs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8494–8502.

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. 2018. Improving language understanding by generative pre-training. *OpenAI blog*.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools.

Richard Shin, Christopher H. Lin, Sam Thomson, Charles Chen, Subhro Roy, Emmanouil Antonios Platanios, Adam Pauls, Dan Klein, Jason Eisner, and Ben Van Durme. 2021. Constrained language models yield few-shot semantic parsers. In *2021 Empirical Methods in Natural Language Processing*.

Richard Shin and Benjamin Van Durme. 2021. Few-shot semantic parsing with language models trained on code. *arXiv preprint arXiv:2112.08696*.

Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. 2020. ALFRED: A Benchmark for Interpreting Grounded Instructions for Everyday Tasks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. 2020. Mpnet: Masked and permuted pre-training for language understanding. *arXiv preprint arXiv:2004.09297*.

Dan Tasse and Noah A Smith. 2008. Sour cream: Toward semantic processing of recipes. *Carnegie Mellon University, Pittsburgh, Tech. Rep. CMU-LTI-08-005*.

Yi Tay, Mostafa Dehghani, Samira Abnar, Yikang Shen, Dara Bahri, Philip Pham, Jinfeng Rao, Liu Yang, Sebastian Ruder, and Donald Metzler. 2020. Long range arena: A benchmark for efficient transformers. *arXiv preprint arXiv:2011.04006*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Yuk Wah Wong and Raymond Mooney. 2006. Learning for semantic parsing with statistical machine translation. In *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference*, pages 439–446.

A Examples & Action Definitions

(Bollini et al., 2013)		
Rank	**Rank + Plan**	**Ground Truth**
pour(flour) mix() mix() preheat(350) bake(20)	pour(flour) mix() mix() preheat(350) **scrape()** bake(20)	pour(flour) mix() **scrape()** preheat(350) bake(20)

Table 4: Additional excerpted generation examples for the (Bollini et al., 2013) dataset.

(Tasse and Smith, 2008)	
A Very Intense Fruit Smoothie	
1 (10 ounce) package frozen mixed berries 1 (15 ounce) can sliced peaches, drained 2 tablespoons honey In a blender, combine frozen fruit, canned fruit and honey. Blend until smooth.	create_ing("1 (10 ounce) package frozen mixed berries") create_ing("1 (15 ounce) can sliced peaches, drained") create_ing("2 tablespoons honey") create_tool("blender") combine("1 (10 ounce) package frozen mixed berries", "1 (15 ounce) can sliced peaches, drained", "2 tablespoons honey", "fruit and honey", "") put("fruit and honey", "blender") mix("fruit and honey", "blender", "smoothie", "blend") chefcheck("smoothie", "smooth")
(Bollini et al., 2013)	
Easy Cake Mix Cookies	
1 (18 1/4 ounce) box chocolate cake mix 1/3 cup vegetable oil 2 eggs Combine cake mix, oil and eggs. Mix well. Bake at 350F for about 10 minutes. Remove from oven and let cool on pan for several minutes before removing to rack to finish cooling.	ingredient(["cake_mix"], "1 (18 1/4 ounce) box", homogenous=True) ingredient(["oil"], "1/3 cup", homogenous=True) ingredient(["eggs"], "2", homogenous=True) pour(cake_mix), pour(oil), pour(eggs), mix() mix() scrape(), preheat(350), bake(10) noop()

Table 5: Example recipes from the (Tasse and Smith, 2008) and (Bollini et al., 2013) datasets. These examples are the second shortest and shortest for each dataset respectively.

(Bollini et al., 2013)
ingredient(contains : string, amount : string, homogenous : bool)
pour(ingredient : string)
scrape()
preheat(temperature : string)
bake(time : string)
noop()

(Tasse and Smith, 2008)
create_tool(name : string)
create_ing(name : string)
chefcheck(name : string, description : string)
cut(item : string, tool : string, result : string, description : string)
combine(item : string, tool : string, result : string, description : string)
cook(item : string, tool : string, result : string, description : string)
do(item : string, tool : string, result : string, description : string)
leave(item : string, description : string)
mix(item : string, tool : string, result : string, description : string)
put(item : string, tool : string)
remove(item : string, tool : string)
separate(item : string, result1 : string, result2 : string, description : string)
serve(item : string, description : string)
set(item : string, description : string)

Table 6: Action definitions for the cooking recipe domains. The actions of (Bollini et al., 2013) operate on a world-state definition that includes the state of the ingredients, a mixing bowl, a baking pan, and the oven. In contrast (Tasse and Smith, 2008) provides no planning domain definitions and employs qualitative descriptions of state transformations in the action annotations. We build a simple planning domain where the state consists only the existence of objects as state variables. The preconditions for an action are met if the items used in the action have been instantiated.