LEMON: Language-Based Environment Manipulation via Execution-Guided Pre-training

Qi Shi[†*], Qian Liu[◊*], Bei Chen[§], Yu Zhang[†], Ting Liu[†], Jian-Guang Lou[§]

[†]Research Center for Social Computing and Information Retrieval,
Harbin Institute of Technology, Harbin, China

[◊]Beihang University, Beijing, China; [§]Microsoft Research Asia, Beijing, China

{qshi, zhangyu, tliu}@ir.hit.edu.cn

qian.liu@buaa.edu.cn; {beichen, jlou}@microsoft.com

Abstract

Language-based environment manipulation requires agents to manipulate the environment following natural language instructions, which is challenging due to the huge space of the environments. To address this challenge, various approaches have been proposed in recent work. Although these approaches work well for their intended environments, they are difficult to generalize across environments. In this work, we propose LEMON, a general framework for language-based environment manipulation tasks. Specifically, we first specify a task-agnostic approach for language-based environment manipulation tasks, which can deal with various environments using the same generative language model. Then we propose an execution-guided pre-training strategy to inject prior knowledge of environments to the language model with a pure synthetic pre-training corpus. Experimental results on tasks including ALCHEMY, SCENE, TANGRAMS, PROPARA and RECIPES demonstrate the effectiveness of LEMON: it achieves new state-of-the-art results on four of the tasks, and the execution-guided pre-training strategy brings remarkable improvements on all experimental tasks[1].

1 Introduction

Building agents that can understand human language and accordingly manipulate the environment around them has been a long-standing goal of artificial intelligence (Winograd, 1971). Various tasks focus on this scene, including collaborative building (Narayan-Chen et al., 2019), state tracking (Dalvi et al., 2018; Tandon et al., 2020) and instruction following (Andreas and Klein, 2015; Long et al., 2016; Suhr et al., 2019). What these tasks have in common is that the agents are required to manipulate the environment based on the natural language. To seize the commonality of existing tasks, we define such tasks as language-based environment manipulation (LEM) tasks. Generally, these tasks are challenging due to the large exploration space of the environment itself and the complexity of human-agent interactions. For example, in the environment shown in Figure 1, the agent needs to manipulate seven beakers with various colored liquids correctly according to the long instruction.

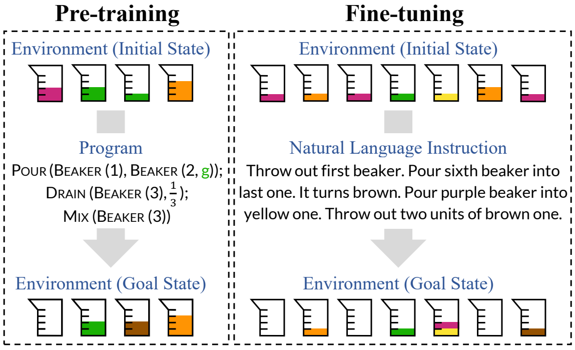

Pre-training
Fine-tuning
Environment (Initial State)
Program
POUR (BEAKER (1), BEAKER (2, g));
DRAIN (BEAKER (3), 1/3);
MIX (BEAKER (3))
Environment (Goal State)
Environment (Initial State)
Natural Language Instruction
Throw out first beaker. Pour sixth beaker into last one. It turns brown. Pour purple beaker into yellow one. Throw out two units of brown one.
Environment (Goal State)

Figure 1: The schematic illustration of the pre-training (**left**) and fine-tuning (**right**) procedure of LEMON. The environment is from ALCHEMY (Long et al., 2016). In the pre-training stage, the input of LEMON includes an initial environment state and a program, and the goal environment state is served as the supervision. The fine-tuning stage is similar to the pre-training stage, except that the program in the model input is replaced by the natural language instruction.

To address these challenges, recent work have proposed various specialized models to deal with different environments (Suhr and Artzi, 2018; Dalvi et al., 2018; Gupta and Durrett, 2019b; Tang et al., 2020). Although these models work well, they are difficult to generalize across environments since they contain environment-specific modules. For example, Suhr and Artzi (2018) design different encoder modules for different environments.

Different from previous work focusing on specialized models, we argue that with formulating

[*] Work done during internship at Microsoft Research Asia.
[1] Our code is available at: https://github.com/microsoft/ContextualSP

Findings of the Association for Computational Linguistics: EMNLP 2022, pages 471 - 485
December 7-11, 2022 ©2022 Association for Computational Linguistics

LEM tasks as sequence generation problems, the family of generative language models (GLMs), such as BART (Lewis et al., 2020), can be an environment-generic agent for various environments. Taking advantage of GLMs, such a task-agnostic solution greatly reduces the difficulty of modeling different environments. However, GLMs generally lack prior knowledge of downstream environments since they have not seen even similar ones during pre-training. To unleash the power of GLMs in downstream environments, we argue that GLMs should be continually pre-trained to understand these environments, and the pre-training should engage GLMs to explore as much of the environment space as possible. We believe if GLMs can understand the environment well, they will more easily manipulate the environment with respect to human language.

Inspired by the above, in this paper, we propose LEMON (for **L**anguage-based **E**nvironment **M**anipulati**on** via Execution-guided Pre-training), a general framework for LEM tasks. As shown in Figure 1, LEMON consists of two parts: 1) A task-agnostic approach that uses the same protocol to tackle different LEM tasks (right). 2) An execution-guided pre-training strategy, which injects prior knowledge about environments into the GLM (left). For the first part, we employ the popular BART (Lewis et al., 2020) as the model backbone, and take five representative tasks ALCHEMY, SCENE, TANGRAMS (Long et al., 2016), PROPARA (Dalvi et al., 2018) and RECIPES (Bosselut et al., 2018) as the testbed. For the pre-training part, it is to engage our model to explore the environment space. Considering that the environment space mainly consists of the state space (i.e., valid environment states) and the action space (i.e., possible actions to manipulate the environment), we suggest pre-training the model via synthesizing data involving these two spaces. Specifically, given an environment, we begin with randomly sampling its relevant initial states and programs[2]. With feeding the random initial state and the random program as input for LEMON, we leverage the goal state after executing the program as supervision for LEMON. Since the program execution is easy to carry out in symbolic environments[3], our execution-guided pre-training is suitable for various symbolic environments. Meanwhile, since the random initial states and the programs can be sampled systematically, we can readily obtain a large-scale high-quality pre-training corpus without human labeling or data cleaning. To the best of our knowledge, LEMON is the first work to explore pre-training in language-based environment manipulation. In summary, the main contributions of our framework LEMON are three-fold:

- We suggest a task-agnostic approach that can be tailored to various environments. By formulating LEM tasks as sequence generation problems, our approach leverages one architecture to tackle them.

- We propose a novel execution-guided pre-training strategy, which can inject prior knowledge of environments by continually pre-training with only synthetic data.

- Experimental results on five tasks demonstrate that our task-agnostic approach is comparable or prior to previous systems, and our pre-training strategy further improves the performance by a significant margin (e.g., +4.1% on ALCHEMY). Finally, our approach achieves new state-of-the-art results on ALCHEMY, SCENE, PROPARA, and RECIPES.

2 LEMON Framework

We now discuss the LEMON framework (Figure 1) in more detail. Specifically, we introduce the task-agnostic approach for language-based environment manipulation (§2.1) and the execution-guided pre-training (§2.2). As for LEMON instantiations for different tasks, we leave the descriptions to §3.

2.1 Task-Agnostic Approach for LEM

As mentioned in §1, the existence of environment-specific modules makes previous models difficult to generalize across environments. To eliminate this issue, we propose a task-agnostic approach to tackle different environments.

Task Formulation An environment space consists of a state space and an action space. And a state can be further decomposed into a set of entities (e.g., beakers in ALCHEMY) and properties (e.g., colors in ALCHEMY). Generally, the goal of LEM tasks is to manipulate the environment state with natural language. Formally, given

[2] A program consists of a sequence of functions, of which each function is an action or a composition of actions.

[3] Symbolic environments stand for the environments that can be represented by semantic symbols.

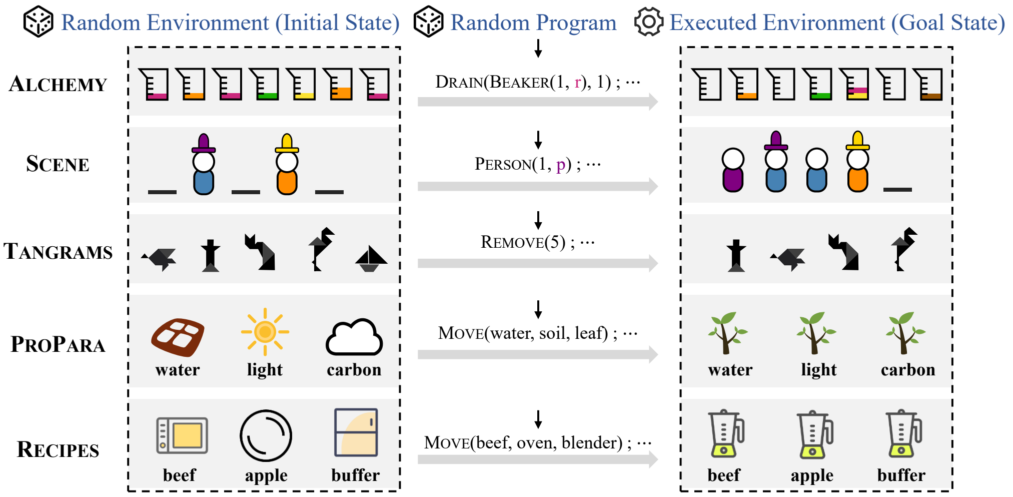

Random Environment (Initial State)
Random Program
Executed Environment (Goal State)
ALCHEMY
SCENE
TANGRAMS
PROPARA
RECIPES
DRAIN(BEAKER(1, r), 1) ; ···
PERSON(1, p) ; ···
REMOVE(5) ; ···
MOVE(water, soil, leaf) ; ···
MOVE(beef, oven, blender) ; ···
water
light
carbon
beef
apple
buffer
water
light
carbon
beef
apple
buffer

Figure 2: The illustration of the pre-training procedure of LEMON framework on five tasks including ALCHEMY, SCENE, TANGRAMS, PROPARA and RECIPES.

an initial environment state S_0, the goal of LEM tasks is to predict the goal environment state S based on the human language instruction I. In most cases, the LEM task is performed in an interactive manner, and there would be a sequence of context-dependent instructions. Again, given an initial environment state S_0 and a sequence of natural language instructions $\mathbf{I} = (I_1, I_2, ..., I_T)$, where T stands for the total number of instructions in one conversation, the goal turns to predict the goal environment states at each step as $\mathbf{S} = (S_1, S_2, ..., S_T)$. In the following, we use the conversational formulation to illustrate.

Model Architecture With formulating LEM tasks as sequence generation problems, we leverage BART (Lewis et al., 2020), a powerful encoder-decoder language model, to generate the goal environment state token-by-token. Formally, at t-th step, the input to our model consists of three parts, namely the initial environment state S_0, the history $(I_1, I_2, ..., I_{t-1})$ and the current instruction I_t. Following previous work (Liu et al., 2020), we directly concatenate the history and the current instruction to form $\mathbf{I}_t = (I_1, I_2, ..., I_t)$, which contains all historical instructions. The final input to our model is the concatenation of S_0 and $\mathbf{I}_t$ with a `[SEP]` token as a separator between them. The output is the corresponding goal environment state S_t.

2.2 Execution-Guided Pre-training

We propose an execution-guided pre-training strategy to explore the environment space as much as possible through synthetic data. In the following, we will introduce the pre-training task and the pre-training corpus generation procedure in turn.

Pre-training Task As described in §1, to encourage the model to understand and explore the environment, LEMON adopts the *program execution* as the pre-training task. Formally, given a randomly sampled initial environment state S_0 and a randomly sampled program A, the model is pre-trained to predict the goal environment state S, as shown in Figure 2. Such a pre-training task fulfills our expectations of both environment exploration and environment understanding, which can be explained from two aspects. From the input perspective, such a task involves all essential elements of an environment (i.e., state and action). Together with large-scale random sampling, it allows the model to fully explore the environment space. From the output perspective, such a task is challenging – the model must understand the environment to predict S correctly. Meanwhile, the program execution as a pre-training task is highly flexible. As shown in Figure 2, it works well for five different tasks. In the implementation of pre-training, we first concatenate S_0 and A using the same `[SEP]` token as a separator and then feed the concatenated sequence into LEMON. The pre-training supervision, i.e., the goal environment state S, is obtained from a task-dependent executor. In LEM tasks, the executor is designed to interpret each task-dependent program and change the current environment state to another state accordingly. In practice, the executor can be easily implemented since the environments are symbolic.

⟨state⟩	→	⟨action⟩; ⟨state⟩ | ⟨action⟩
⟨action⟩	→	⟨mix⟩ | ⟨pour⟩ | ⟨drain⟩
⟨mix⟩	→	MIX (⟨beaker⟩)
⟨pour⟩	→	POUR (⟨beaker⟩, ⟨beaker⟩)
⟨drain⟩	→	DRAIN (⟨beaker⟩, ⟨integer⟩) |
		DRAIN (⟨beaker⟩, ⟨fraction⟩)
⟨beaker⟩	→	BEAKER (⟨index⟩) |
		BEAKER (⟨index⟩, ⟨color⟩)
⟨index⟩	→	1 | 2 | ··· | 7 | −1 | −2 | ··· | −7
⟨color⟩	→	r | g | o | p | y | b
⟨integer⟩	→	1 | 2 | 3 | 4
⟨fraction⟩	→	$\frac{1}{2}$ | $\frac{1}{3}$ | $\frac{1}{4}$ | $\frac{2}{3}$ | $\frac{2}{4}$ | $\frac{3}{4}$

Table 1: Grammar rules of the program used in ALCHEMY. The grammar rules of other domains and the descriptions can be found in Appendix A.

Pre-training Corpus Generation Unlike most pre-training work that employs web crawling to collect pre-training corpus, we synthesize the pre-training corpus directly by randomly sampling the environment states and programs. Compared to human language, high-quality environment states and programs are easier to sample since they are highly structured. As introduced above, each pre-training example contains a sampled initial environment state, a sampled executable program, and a goal environment state obtained from the executor. One by one, the pre-training corpus can be generated by repeating the sampling process. Concretely, for the initial environment state sampling, it can be achieved by randomly selecting a valid value for each property defined in the corresponding environment. As for the program sampling, a valid program can be generated by randomly selecting a valid function and then randomly sampling from all suitable parameters of the selected function. The valid values for each property and function will be discussed later.

3 LEMON Instantiations

To demonstrate the capabilities of LEMON, we apply our framework on five exemplary tasks, namely, ALCHEMY, SCENE, TANGRAMS, PROPARA, and RECIPES. Examples of each task are shown in Figure 2, including visualizations of the initial environment state and the goal environment state, as well as a schematic representation of the program. In this section, for each task, we elaborate the definition of the environment and the applied program to instantiate LEMON.

3.1 ALCHEMY

Environment State Definition The environment state in ALCHEMY contains seven beakers, each containing up to four units of colored chemicals. Each environment state contains three properties, including beaker IDs (from 1 to 7), liquid colors (brown, green, orange, purple, red, and yellow), and liquid amounts (from 0 to 4). Figure 2 shows an example, and the example initial environment state can be represented as `1:r|2:o|3:r|4:g|5:y|6:oo|7:r` in text, where different letters represent different colors. Note that if a beaker does not contain any liquid, it can be represented by `_`. And `|` stands for the delimiter that splits the state of each beaker, which is also applicable for the following tasks.

Program Definition The action space of ALCHEMY contains three kinds of actions to manipulate the environment, namely, POUR, DRAIN and MIX. We use the program proposed by Guu et al. (2017), where the functions are the same as the actions defined in the environment. The detail program grammar is shown in Table 1.

3.2 SCENE

Environment State Definition The environment state in SCENE contains ten positions, with up to one person in each position. A person is defined by a shirt color and optionally a hat color. Formally, each environment state contains three properties, including position IDs (from 1 to 10), shirt colors (brown, green, orange, purple, red, and yellow), and hat colors (the same as shirt colors). As shown in Figure 2, the example initial environment state can be represented as `1:__|2:bp|3:__|4:oy|5:__` (only five positions are shown in Figure 2 for brevity) in text, where the first character in each position represents the shirt color and the second one represents the hat color. `_` indicates either an empty position or a person without a hat. Note that the hat can only appear when the position is occupied.

Program Definition Four actions are defined in the SCENE environment to manipulate the environment, namely, ENTER, LEAVE, MOVE and TRADE-HATS. For the program, we use the one proposed by Suhr and Artzi (2018). The functions include PERSON, HAT, RMPERSON and RMHAT, which represent inserting / removing a person / hat in the state. The permutations of the defined functions in the program are sufficient to represent all actions defined in the environment.

Models	Alchemy			Scene			Tangrams		
	Inst	3utts	5utts	Inst	3utts	5utts	Inst	3utts	5utts
Fully Supervised Approaches									
(Fried et al., 2018)	–	–	72.0	–	–	72.7	–	–	69.6
(Huang et al., 2019)	–	–	76.4	–	–	74.5	–	–	72.3
(Yeh and Chen, 2019)	–	–	76.1	–	–	75.1	–	–	72.5
Weakly Supervised Approaches									
(Long et al., 2016)	–	56.8	52.3	–	23.2	14.7	–	64.9	27.6
(Guu et al., 2017)	–	66.9	52.9	–	64.8	46.2	–	65.8	37.1
(Suhr and Artzi, 2018) *w.* Reinforce	89.1	74.2	62.7	87.1	73.9	62.0	86.6	80.8	**62.4**
(Suhr and Artzi, 2018) *w.* Heuristic	89.4	73.3	62.3	88.8	78.9	66.4	86.6	81.4	60.1
LEMon	**97.1**	**85.3**	**75.4**	**92.7**	**85.8**	**72.3**	92.3	82.4	60.0
w.o. pre-training	96.9	84.0	71.3	91.6	83.1	68.9	**92.8**	**83.4**	56.7

Table 2: Experimental results on the test set of Alchemy, Scene and Tangrams. Fully supervised approaches (in grey background) are the approaches that use **annotated** programs as labels, while weakly supervised approaches are the approaches that no golden program is provided. Although the comparisons are **not fair**, we report the results of fully supervised approaches for reference. Note that our ablation *w.o.* pre-training is identical to fine-tuning BART on the downstream task, and the same for Table 3 and Table 4.

3.3 Tangrams

Environment State Definition The environment state in Tangrams contains a list of up to five unique objects. Similarly, the environment state can be represented by the object indexes (from 1 to 5) and the object names (A, B, C, D, and E). For example, the initial environment state in Figure 2 can be represented as 1:A|2:B|3:C|4:D|5:E. The same object cannot appear in one environment state. If the number of objects is less than 5, we fill the sequence with _ to make it 5 in length.

Program Definition Three actions are involved to manipulate the Tangrams environment, namely, Add, Remove and Swap. And we use the program proposed by Suhr and Artzi (2018), which defines the functions including Insert and Remove. Similar to the two kinds of programs mentioned above, permuting these two functions can achieve the goal of representing all actions defined in the environment.

3.4 ProPara & Recipes

Environment State Definition The ProPara environment describes real-world scientific processes such as photosynthesis, erosion, etc. Each environment state in ProPara contains a set of entity participants and their corresponding locations, and the locations vary with the natural language procedural text being described. Unlike the three environments mentioned above, the properties of an environment state in ProPara are not fixed, but are dynamically constructed from the natural language text. Figure 2 shows an example, where the initial state stands for participants water, light, carbon are located in locations soil, sun, cloud respectively. The environment state in ProPara can be naturally represented in key-value format. For example, the initial state in Figure 2 can be represented as `ent:water|light|carbon loc:soil|sun|cloud`, here `ent:` and `loc:` are special tokens that indicate the boundaries of entity participants and locations, respectively.

Program Definition In the ProPara environment, the procedural text describes four actions, namely, Create, Move, Destroy and None. In practice, we use the program proposed by Dalvi et al. (2019), in which the functions also contain Create, Move and Destroy, which are aligned with the action space of ProPara. As for Recipes, the environment describes the state tracking process in the cooking domain. And the definition of the environment states and the programs are similar with ProPara.

4 Experiments

In this section, we compare LEMon with baseline methods on the tasks discussed in §3 to demonstrate its effectiveness. Due to space limitation, we do not introduce these baselines below.

4.1 Data and Evaluation

Alchemy, Scene & Tangrams These three tasks are introduced with different environments in the SCoNE corpus (Long et al., 2016). Each human-agent interaction has 5 instructions. Following Long et al. (2016), we evaluate LEMon

Models	Sentence-Level					Document-Level		
	Cat-1	Cat-2	Cat-3	Macro-Avg	Micro-Avg	Precision	Recall	F1
EntNet (Henaff et al., 2017)	51.6	18.8	7.8	26.1	26.0	54.7	30.7	39.4
QRN (Seo et al., 2017)	52.4	15.5	10.9	26.3	26.5	60.9	31.1	41.1
ProLocal (Dalvi et al., 2018)	62.7	30.5	10.4	34.5	34.0	81.7	36.8	50.7
ProGlobal (Dalvi et al., 2018)	63.0	36.4	35.9	45.1	45.4	61.7	48.8	51.9
AQA (Ribeiro et al., 2019)	61.6	40.1	18.6	39.4	40.1	62.0	45.1	52.3
ProStruct (Tandon et al., 2018)	–	–	–	–	–	74.3	43.0	54.5
XPAD (Dalvi et al., 2019)	–	–	–	–	–	70.5	45.3	55.2
LACE (Du et al., 2019)	–	–	–	–	–	75.3	45.4	56.6
KG-MRC (Das et al., 2019)	62.9	40.0	38.2	47.0	46.6	69.3	49.3	57.6
ProGraph (Zhong et al., 2020)	67.8	44.6	41.8	51.4	51.5	67.3	55.8	61.0
IEN (Tang et al., 2020)	71.8	47.6	40.5	53.3	53.0	69.8	56.3	62.3
NCET (Gupta and Durrett, 2019b)	73.7	47.1	41.0	53.9	54.0	67.1	58.5	62.5
ET$_{\text{BERT}}$ (Gupta and Durrett, 2019a)	73.6	52.6	–	–	–	–	–	–
DynaPro (Amini et al., 2020)	72.4	49.3	**44.5**	55.4	55.5	75.2	58.0	65.5
TSLM (Rajaby Faghihi and Kordjamshidi, 2021)	78.8	56.8	40.9	58.8	58.4	68.4	68.9	68.6
KoaLa (Zhang et al., 2021)	78.5	53.3	41.3	57.7	57.5	77.7	64.4	70.4
REAL (Huang et al., 2021)	78.4	53.7	42.4	58.2	57.9	**81.9**	61.9	70.5
LEMon	**81.7**	**58.3**	43.3	**61.1**	**60.7**	74.8	**69.8**	**72.2**
w.o. pre-training	78.8	57.2	42.9	59.6	59.2	69.9	68.1	69.0

Table 3: Experimental results of our method LEMon and baselines on the test set of ProPara.

with denotation accuracy. In addition, the evaluation metrics can be divided into the denotation accuracy of a single instruction (Inst), of the first three instructions (3utts), and of the complete interactions (5utts).

ProPara & Recipes These two tasks is introduced in two procedural text understanding datasets (Dalvi et al., 2018; Bosselut et al., 2018), and are designed to track entity states through natural language paragraphs. For ProPara, the evaluation metrics are composed of two levels: the sentence-level and the document-level. The **sentence-level** evaluates the model based on its prediction for the following three questions: Is entity Created, Moved or Destroyed in the process? When is entity Created, Moved or Destroyed? Where is entity Created, Moved or Destroyed? The sentence-level metrics include the accuracy of the above questions (Cat-1, Cat-2, Cat-3), and their micro / macro-average. The **document-level** evaluates the model based on its prediction on four document-level questions: What are the inputs? What are the outputs? What are the conversions? What are the moves? The document-level metrics report the average precision, recall, and F1 score of the four questions. For Recipes, following previous work (Zhang et al., 2021; Huang et al., 2021), we report the location changes of each entity, and take precision, recall and F1 scores as the evaluation metrics. The statistics of 5 datasets can be found in Appendix B.

Models	Precision	Recall	F1
NCET (re-implementation)	56.5	46.4	50.9
IEN (re-implementation)	58.5	47.0	52.2
KoaLa (Zhang et al., 2021)	**60.1**	52.6	56.1
REAL (Huang et al., 2021)	55.2	52.9	54.1
LEMon	56.0	**67.1**	**61.1**
w.o. pre-training	53.9	63.6	58.4

Table 4: Experimental results of our method LEMon and baselines on the test set of Recipes.

4.2 Experimental Setup

We use BART-Large in fairseq (Ott et al., 2019) to implement LEMon. During pre-training, we synthesize 1 million pre-training examples for each experimental task. The learning rate is set to 3×10^{-5} in all experiments of pre-training and fine-tuning. During pre-training, the maximum training step is set to $10,000$ for Alchemy, Scene, Tangrams and $2,000$ for ProPara and Recipes, while the batch size is set to around $1,000$ for all tasks. During fine-tuning, the maximum training step is set to $10,000$ for all tasks, while the batch size is set to 64 for Alchemy, Scene, Tangrams and 32 for ProPara and Recipes, respectively.

4.3 Experimental Results

Alchemy & Scene From Table 2, we can observe that LEMon outperforms previous best-performing systems under weak supervision on both Alchemy and Scene, with significant improvements of 13.1% and 5.9% in the 5utts denotation accuracy, respectively. Notably, LEMon

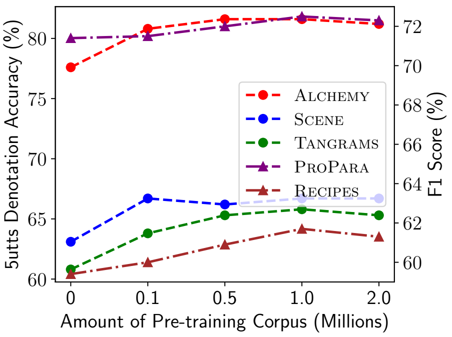


Figure 3: The performance of downstream tasks with respect to the amount of pre-training corpus. We plot the 5utts denotation accuracy on ALCHEMY, SCENE and TANGRAMS (circle), and plot the F1 score on PROPARA and RECIPES (triangle).

not only achieves new state-of-the-art performance among weakly supervised approaches, but also comes close to the performance of fully supervised approaches that leverage extra annotated programs. Moreover, the results also show that the execution-guided pre-training brings significant improvements (e.g., 4.1% of ALCHEMY in the 5utts denotation accuracy), which demonstrates that our pre-training strategy provides considerable prior knowledge for LEMON.

TANGRAMS Similarly, the results on TANGRAMS in Table 2 show that our execution-guided pre-training strategy improves LEMON by 3.3% in the 5utts denotation accuracy, further illustrating the effectiveness of our approach. Nevertheless, LEMON does not perform as well compared to previous state-of-the-art method (Suhr and Artzi, 2018). We suppose this is because Suhr and Artzi (2018) carefully model the historical instructions, while LEMON directly concatenates them. We leave the fine-grained context modeling of our approach for future work.

PROPARA Table 3 summarizes the results of the PROPARA task, in which LEMON achieves new state-of-the-art performance based on both the sentence-level and the document-level evaluation. On the sentence-level evaluation, LEMON shows stable improvements in most metrics compared to both previous approaches and LEMON *w.o.* execution-guided pre-training, which demonstrates that LEMON achieves an overall improvement in the understanding of procedural texts with respect to the environment. On the document-level evaluation, LEMON achieves an F1 score of 72.2%, which is 1.7% higher than the previous best-performing system REAL (Huang et al., 2021) and 1.8% higher than KOALA (Zhang et al., 2021). Note the improvement is highly non-trivial since KOALA leverages external knowledge, which indicates that the prior knowledge LEMON learns during pre-training is more effective than external knowledge. Similarly, the execution-guided pre-training brings a 3.2% improvement, which again demonstrates that the pre-training in LEMON can significantly facilitate the interaction procedure between natural language and environments.

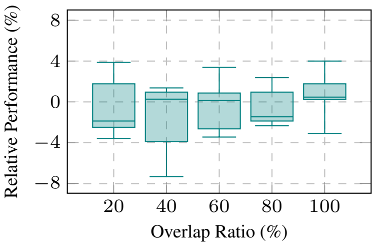


Figure 4: The relative performance of downstream tasks with respect to the overlap ratio.

RECIPES Table 4 shows the experimental results of the RECIPES task that LEMON reach state-of-the-art performance and surpass previous best-performing systems (Huang et al., 2021) with a large margin by 7.0%. Besides, the proposed execution-guided pre-training also brings a 2.7% improvement. These results further illustrate the effectiveness of LEMON.

4.4 Pre-training Analysis

Scaling up pre-training has a positive impact Previous work (Lewis et al., 2020) has shown that the scale of the pre-training corpus is an important factor in pre-training, and thus we analyze the effect of our pre-training scale on downstream tasks. Figure 3 shows the performance of downstream tasks with respect to the size of the pre-training corpus, which are obtained from the validation set of each task. As seen, the performance of the model generally improves by scaling up the pre-training corpus, consistent with previous observations on pre-training (Liu et al., 2021).

Improvements do not come from data leakage Since the pre-training corpus of LEMON contains various randomly sampled environment states, this may raise the doubt that the improvements of LEMON is due to the data leakage, that is, LEMON has seen some environments in the downstream val-

Type (Percent)	Example	Environment (Goal State) Comparison
Operation Correctness (68.3%)	**Environment (Initial State)**: **Instruction**: Empty out the first beaker, add the orange chemical to the red.	The difference lies in the third beaker, (*w.o.* pre-training) versus (*w.* pre-training). Without pre-training, the model does not correctly understand the semantics of "add", i.e., removing the liquid from the third beaker.
Instruction Completeness (18.3%)	**Environment (Initial State)**: **Instruction**: Throw out one unit of the second beaker, pour the second beaker into first one.	The difference lies in the first beaker, (*w.o.* pre-training) versus (*w.* pre-training). As observed, without pre-training, the model seems to ignore the instruction "throw out one unit of the second beaker".
Grounding Correctness (8.5%)	**Environment (Initial State)**: **Instruction**: Pour out one part of the second yellow beaker.	The whole states are (*w.o.* pre-training) versus (*w.* pre-training). Without pre-training, the model does not find the correct beaker according to the instructions.

Table 5: The main types of the improvements by the execution-guided pre-training in the validation set of ALCHEMY. For the all domains, please refer to Appendix D.

idation sets. Although we have already ensure that the pre-training corpus does not contain the environment states in the validation set and the test set, it is still interesting to investigate the potential impact of data leakage on LEMON. To analyze the effect, we create a validation corpus of size $40,000$ for each task, which contains only the environment states in the validation set, and then merge the cases from the validation corpus into the pre-training corpus with certain ratios (denoted as **overlap ratio**). Figure 4 shows the box plot of the relative performance to the reported performance (vertical axis) with respect to the overlap ratio (horizontal axis). We can observe that from the perspective of the vertical axis, the vertical axis of the densest area is near 0, indicating that the two variables are irrelevant, thus proving that the effectiveness of LEMON hardly relies on the overlap between corpus size and validation set. For the detailed downstream performance with respect to the overlap ratio, please refer to Appendix C.

Improvements come from prior knowledge acquirement To show what LEMON obtain from the execution-guided pre-training procedure, we manually analyze examples in the validation set where predictions are wrong before pre-training and correct after pre-training. Table 5 shows the main types of the improvements caused by the execution-guided pre-training. We can see that with the execution-guided pre-training, LEMON successfully masters the prior knowledge of different environments. Specifically, LEMON can manipulate the environments better, as reflected in the correctness of operations, the completeness of instructions, and the correctness of grounding.

5 Related Work

Language-based Environment Manipulation The first line of our related work is the previous work on LEM tasks. According to the output, existing methods on LEM tasks can be mainly divided into two categories: program prediction and state prediction. Prior work always treat the LEM task as a program prediction problem (Long et al., 2016; Guu et al., 2017; Suhr and Artzi, 2018; Fried et al., 2018; Huang et al., 2019; Yeh and Chen, 2019; Dalvi et al., 2019). However, these approaches are environment-dependent and cannot be easily adapted to other environments. Besides, they either rely on natural language-program pairs as supervision or require complex heuristic rules, which is costly. Recent approaches generally treat the LEM task as a state prediction problem by predicting the goal state directly (Dalvi et al., 2018; Du et al., 2019; Das et al., 2019; Tang et al., 2020; Rajaby Faghihi and Kordjamshidi, 2021; Zhang et al., 2021). These models can eliminate the data collection issue, but require complex models designed to meet the needs of different kinds of environments. Compared with the above work, LEMON has the following advantages: 1) The proposed task-agnostic approach does not require additional annotations and is easy to generalize across different environments. 2) The proposed execution-guided pre-training strategy can further improve the model performance with synthetic data only.

Program Execution The second line of our related work is the execution-guided work, of which the most related work are ProTo (Zhao et al., 2021) and TAPEX (Liu et al., 2021). ProTo learns to execute given programs on the observed task specifications, which focuses on following a given program to perform the corresponding task. Different from ProTo, LEMON focuses on pre-training with program execution to enhance the downstream performance. Following a similar idea, TAPEX (Liu et al., 2021) improves the table pre-training by learning SQL execution over tables. The main difference between TAPEX and LEMON is that TAPEX choose SQL execution as the pre-training task, which is suitable for a single environment only. While, LEMON is more flexible, and enables us to systematically design the pre-training task and synthesize pre-training corpus based on environment properties, and proven effective on multiple environments.

6 Conclusion & Future Work

In this work, we propose LEMON, a general framework for language-based environment manipulation tasks that not only models different environments using the same protocol, but also injects prior knowledge of environments into our model. Experimental results on five tasks demonstrate the effectiveness of LEMON: the execution-guided pre-training strategy brings significant improvements on all of them and LEMON achieves the state-of-the-art performance on four of them. For future work, we hope to extend our approach to more complex environments and tasks such as image editing (Fu et al., 2020) and text editing (Faltings et al., 2021).

Limitations

The main limitation in this paper is that LEMON focus on symbolic environments instead of raw environments with only visual features. Compared to the latter, the former can be represented by semantic symbols, and thus enjoys better controllability and interpretability. We leave the exploration of raw environments for future work.

Ethics Statement

In this paper, we propose LEMON, a general framework for language-based environment manipulation tasks, consisting of a task-agnostic approach and an execution-guided pre-training strategy. We conduct experiments on five benchmarks, namely, ALCHEMY, SCENE, TANGRAMS, PROPARA, RECIPES. All benchmarks are free and open for research use. The pre-training corpus is generated based on open-source program grammars, which are no ethics issues.

Acknowledgement

We would like to thank the anonymous reviewers for their helpful comments.

References

Aida Amini, Antoine Bosselut, Bhavana Dalvi Mishra, Yejin Choi, and Hannaneh Hajishirzi. 2020. Procedural reading comprehension with attribute-aware context flow. In *Conference on Automated Knowledge Base Construction, AKBC 2020, Virtual, June 22-24, 2020*.

Jacob Andreas and Dan Klein. 2015. Alignment-based compositional semantics for instruction following. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1165–1174, Lisbon, Portugal. Association for Computational Linguistics.

Antoine Bosselut, Omer Levy, Ari Holtzman, Corin Ennis, Dieter Fox, and Yejin Choi. 2018. Simulating action dynamics with neural process networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net.

Bhavana Dalvi, Lifu Huang, Niket Tandon, Wen-tau Yih, and Peter Clark. 2018. Tracking state changes in procedural text: a challenge dataset and models for process paragraph comprehension. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1595–1604, New Orleans, Louisiana. Association for Computational Linguistics.

Bhavana Dalvi, Niket Tandon, Antoine Bosselut, Wen-tau Yih, and Peter Clark. 2019. Everything happens for a reason: Discovering the purpose of actions in procedural text. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4496–4505, Hong Kong, China. Association for Computational Linguistics.

Rajarshi Das, Tsendsuren Munkhdalai, Xingdi Yuan, Adam Trischler, and Andrew McCallum. 2019. Building dynamic knowledge graphs from text using machine reading comprehension. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Xinya Du, Bhavana Dalvi, Niket Tandon, Antoine Bosselut, Wen-tau Yih, Peter Clark, and Claire Cardie. 2019. Be consistent! improving procedural text comprehension using label consistency. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2347–2356, Minneapolis, Minnesota. Association for Computational Linguistics.

Felix Faltings, Michel Galley, Gerold Hintz, Chris Brockett, Chris Quirk, Jianfeng Gao, and Bill Dolan.

2021. Text editing by command. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5259–5274, Online. Association for Computational Linguistics.

Daniel Fried, Jacob Andreas, and Dan Klein. 2018. Unified pragmatic models for generating and following instructions. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1951–1963, New Orleans, Louisiana. Association for Computational Linguistics.

Tsu-Jui Fu, Xin Wang, Scott Grafton, Miguel Eckstein, and William Yang Wang. 2020. SSCR: Iterative language-based image editing via self-supervised counterfactual reasoning. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4413–4422, Online. Association for Computational Linguistics.

Aditya Gupta and Greg Durrett. 2019a. Effective use of transformer networks for entity tracking. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 759–769, Hong Kong, China. Association for Computational Linguistics.

Aditya Gupta and Greg Durrett. 2019b. Tracking discrete and continuous entity state for process understanding. In *Proceedings of the Third Workshop on Structured Prediction for NLP*, pages 7–12, Minneapolis, Minnesota. Association for Computational Linguistics.

Kelvin Guu, Panupong Pasupat, Evan Liu, and Percy Liang. 2017. From language to programs: Bridging reinforcement learning and maximum marginal likelihood. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1051–1062, Vancouver, Canada. Association for Computational Linguistics.

Mikael Henaff, Jason Weston, Arthur Szlam, Antoine Bordes, and Yann LeCun. 2017. Tracking the world state with recurrent entity networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

Hao Huang, Xiubo Geng, Jian Pei, Guodong Long, and Daxin Jiang. 2021. Reasoning over entity-action-location graph for procedural text understanding. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 5100–5109.

Hsin-Yuan Huang, Eunsol Choi, and Wen-tau Yih. 2019. Flowqa: Grasping flow in history for conversational machine comprehension. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online. Association for Computational Linguistics.

Qian Liu, Bei Chen, Jiaqi Guo, Jian-Guang Lou, Bin Zhou, and Dongmei Zhang. 2020. How far are we from effective context modeling? an exploratory study on semantic parsing in context. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, pages 3580–3586. ijcai.org.

Qian Liu, Bei Chen, Jiaqi Guo, Morteza Ziyadi, Zeqi Lin, Weizhu Chen, and Jian-Guang Lou. 2021. TAPEX: table pre-training via learning a neural SQL executor. *CoRR*, abs/2107.07653.

Reginald Long, Panupong Pasupat, and Percy Liang. 2016. Simpler context-dependent logical forms via model projections. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1456–1465, Berlin, Germany. Association for Computational Linguistics.

Anjali Narayan-Chen, Prashant Jayannavar, and Julia Hockenmaier. 2019. Collaborative dialogue in Minecraft. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5405–5415, Florence, Italy. Association for Computational Linguistics.

Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. 2019. fairseq: A fast, extensible toolkit for sequence modeling. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*, pages 48–53, Minneapolis, Minnesota. Association for Computational Linguistics.

Hossein Rajaby Faghihi and Parisa Kordjamshidi. 2021. Time-stamped language model: Teaching language models to understand the flow of events. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4560–4570, Online. Association for Computational Linguistics.

Danilo Ribeiro, Thomas Hinrichs, Maxwell Crouse, Kenneth Forbus, Maria Chang, and Michael Witbrock. 2019. Predicting state changes in procedural text using analogical question answering. In *Proc. of ACS*.

Min Joon Seo, Sewon Min, Ali Farhadi, and Hannaneh Hajishirzi. 2017. Query-reduction networks for question answering. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

Alane Suhr and Yoav Artzi. 2018. Situated mapping of sequential instructions to actions with single-step reward observation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2072–2082, Melbourne, Australia. Association for Computational Linguistics.

Alane Suhr, Claudia Yan, Jack Schluger, Stanley Yu, Hadi Khader, Marwa Mouallem, Iris Zhang, and Yoav Artzi. 2019. Executing instructions in situated collaborative interactions. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2119–2130, Hong Kong, China. Association for Computational Linguistics.

Niket Tandon, Bhavana Dalvi, Joel Grus, Wen-tau Yih, Antoine Bosselut, and Peter Clark. 2018. Reasoning about actions and state changes by injecting commonsense knowledge. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 57–66, Brussels, Belgium. Association for Computational Linguistics.

Niket Tandon, Keisuke Sakaguchi, Bhavana Dalvi, Dheeraj Rajagopal, Peter Clark, Michal Guerquin, Kyle Richardson, and Eduard Hovy. 2020. A dataset for tracking entities in open domain procedural text. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6408–6417, Online. Association for Computational Linguistics.

Jizhi Tang, Yansong Feng, and Dongyan Zhao. 2020. Understanding procedural text using interactive entity networks. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7281–7290, Online. Association for Computational Linguistics.

Terry Winograd. 1971. Procedures as a representation for data in a computer program for understanding natural language. Technical report.

Yi-Ting Yeh and Yun-Nung Chen. 2019. Flowdelta: Modeling flow information gain in reasoning for conversational machine comprehension. In *Proceedings of the 2nd Workshop on Machine Reading for Question Answering, MRQA@EMNLP 2019, Hong Kong, China, November 4, 2019*, pages 86–90. Association for Computational Linguistics.

Zhihan Zhang, Xiubo Geng, Tao Qin, Yunfang Wu, and Daxin Jiang. 2021. Knowledge-aware procedural text understanding with multi-stage training. In *WWW '21: The Web Conference 2021, Virtual Event / Ljubljana, Slovenia, April 19-23, 2021*, pages 3512–3523. ACM / IW3C2.

Zelin Zhao, Karan Samel, Binghong Chen, and Le Song. 2021. Proto: Program-guided transformer for program-guided tasks. *CoRR*, abs/2110.00804.

Wanjun Zhong, Duyu Tang, Nan Duan, Ming Zhou, Jiahai Wang, and Jian Yin. 2020. A heterogeneous graph with factual, temporal and logical knowledge for question answering over dynamic contexts. *CoRR*, abs/2004.12057.

A Program Grammar in Each Domain

Table 6, Table 7, Table 8 and Table 9 show the grammar rules of used programs in each domain.

B Statistics of Each Dataset

Table 10 show the data statistics for ALCHEMY, SCENE, TANGRAMS, PROPARA, RECIPES.

C Downstream Performance *w.r.t* Overlap Ratio

Table 11 shows the downstream performance on the validation sets with respect to the overlap ratio in the pre-training corpus.

D Pre-training Improvement Analysis

The main types of the improvements by the execution-guided pre-training on the five tasks are shown in Table 12 and Table 13.

E Example Program, Initial State and Goal State of Each Domain

Table 14 shows the examples of each domain, including the initial environment state, the program, and the corresponding goal environment state.

F Case Study

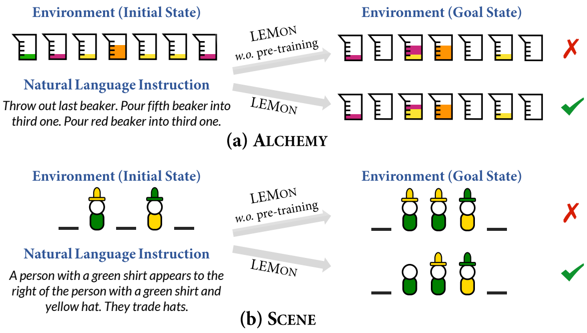
Figure 5 shows two cases in ALCHEMY and SCENE, providing a more intuitive view of the role played by the execution-guided pre-training in LEMON. We display the initial environment states, the natural language instructions, and the goal environment states predicted with / without applying the execution-guided pre-training strategy, respectively. In the first case (a), when pouring yellow liquid from the 5-th beaker into the 3-th beaker, the latter receives red liquid, which is clearly an inconsistent change. However, with pre-training, LEMON can predict the correct goal environment state via deeply understanding the actions conveyed

Grammar Rule			Description
⟨state⟩	→	⟨action⟩ ⟨state⟩ | ⟨action⟩	A list of actions.
⟨action⟩	→	⟨mix⟩ | ⟨pour⟩ | ⟨drain⟩	One of the three actions.
⟨mix⟩	→	MIX (⟨beaker⟩)	Mix the liquid in the ⟨beaker⟩ beaker.
⟨pour⟩	→	POUR (⟨beaker⟩, ⟨beaker⟩)	Pour the liquid from the first beaker to the second beaker.
⟨drain⟩	→	DRAIN (⟨beaker⟩, ⟨integer⟩) |	Pour out the ⟨integer⟩ unit from the ⟨beaker⟩ beaker.
		DRAIN (⟨beaker⟩, ⟨fraction⟩)	Pour ⟨fraction⟩ of the liquid out of the ⟨beaker⟩ beaker.
⟨beaker⟩	→	BEAKER (⟨index⟩) |	The ⟨index⟩-th beaker.
		BEAKER (⟨index⟩, ⟨color⟩)	The ⟨index⟩-th beaker of ⟨color⟩ color.
⟨index⟩	→	1 | 2 | ··· | 7 | −1 | −2 | ··· | −7	The index of the certain components in the environment.
⟨color⟩	→	r | g | o | p | y | b	The symbols corresponding to the color red, green, orange, purple, yellow and brown.
⟨integer⟩	→	1 | 2 | 3 | 4	The unit of the liquid.
⟨fraction⟩	→	$\frac{1}{2}$ | $\frac{1}{3}$ | $\frac{1}{4}$ | $\frac{2}{3}$ | $\frac{2}{4}$ | $\frac{3}{4}$	The percentage of the liquid.

Table 6: Grammar rules and corresponding descriptions of used program in ALCHEMY.

Grammar Rule			Description
⟨state⟩	→	⟨action⟩ ⟨state⟩ | ⟨action⟩	A list of actions.
⟨action⟩	→	⟨person⟩ | ⟨rmperson⟩ | ⟨hat⟩ | ⟨rmhat⟩	One of the four actions.
⟨person⟩	→	PERSON (⟨index⟩, ⟨color⟩)	Add a person with ⟨color⟩ shirt on the ⟨index⟩-th position.
⟨rmperson⟩	→	RMPERSON (⟨index⟩)	Remove the person on the ⟨index⟩-th position.
⟨hat⟩	→	HAT (⟨index⟩, ⟨color⟩)	Add a hat of ⟨color⟩ color for the person on the ⟨index⟩-th position.
⟨rmhat⟩	→	RMHAT (⟨index⟩)	Remove the person's hat on the ⟨index⟩-th position.
⟨index⟩	→	1 | 2 | 3 | ··· | 10	The index of the certain components in the environment.
⟨color⟩	→	r | g | o | p | y | b	The symbols corresponding to the color red, green, orange, purple, yellow and brown.

Table 7: Grammar rules and corresponding descriptions of used program in SCENE.


Environment (Initial State)
Environment (Goal State)
LEMON w.o. pre-training
LEMON
✗
✓
Natural Language Instruction
Throw out last beaker. Pour fifth beaker into third one. Pour red beaker into third one.
Environment (Initial State)
Environment (Goal State)
LEMON w.o. pre-training
LEMON
✗
✓
Natural Language Instruction
A person with a green shirt appears to the right of the person with a green shirt and yellow hat. They trade hats.

(a) ALCHEMY

(b) SCENE

Figure 5: Two cases of LEMON and LEMON *w.o.* pre-training in ALCHEMY and SCENE. The predictions of LEMON are more consistent with the semantics of the natural language instruction.

by natural language. Similarly, in the second case (b), when swapping the hats in the last step, the model does not understand the TRADE-HAT action correctly, while it can be well understood to generate the goal state after pre-training. The above two cases indicate that the execution-guided pre-training strategy is able to inject prior knowledge of environments into LEMON and benefit the downstream tasks.

Grammar Rule			Description
⟨state⟩	→	⟨action⟩ ⟨state⟩ | ⟨action⟩	A list of actions.
⟨action⟩	→	⟨insert⟩ | ⟨remove⟩	One of the two actions.
⟨insert⟩	→	INSERT (⟨index⟩, ⟨object⟩)	Insert the ⟨object⟩ object at the ⟨index⟩ position.
⟨remove⟩	→	REMOVE (⟨index⟩)	Remove the object at the ⟨index⟩ position.
⟨index⟩	→	1 | 2 | 3 | 4 | 5	The index of the certain components in the environment.
⟨object⟩	→	A | B | C | D | E	The object name.

Table 8: Grammar rules and corresponding descriptions of used program in TANGRAMS.

Grammar Rule			Description
⟨state⟩	→	⟨action⟩ ⟨state⟩ | ⟨action⟩	A list of actions.
⟨action⟩	→	⟨create⟩ | ⟨move⟩ | ⟨destroy⟩	One of the three actions.
⟨create⟩	→	CREATE (⟨participant⟩, ⟨location⟩)	Create ⟨participant⟩ at the ⟨location⟩.
		CREATE (⟨participant⟩, ?)	Do not fill the location if ⟨location⟩ is not given.
⟨move⟩	→	MOVE (⟨participant⟩, ⟨location1⟩, ⟨location2⟩)	Move ⟨participant⟩ from ⟨location1⟩ to ⟨location2⟩.
⟨destroy⟩	→	DESTROY (⟨participant⟩)	Remove ⟨participant⟩ from the current location.
⟨participants⟩	→	water | light | carbon | ...	Entities in the environments.
⟨locations⟩	→	soil | sun | cloud | ...	Entities' locations in the environments

Table 9: Grammar rules and corresponding descriptions of used program in PROPARA and RECIPES.

Dataset	Statistics	Train	Dev	Test	Total
ALCHEMY	#Interaction	3657	245	899	4801
	#Instruction	18285	1225	4495	24005
	Avg.inst/inte	5	5	5	5
	Avg.word/inst	-	-	-	8.0
SCENE	#Interaction	3352	198	1035	4585
	#Instruction	16760	990	5175	22925
	Avg.inst/inte	5	5	5	5
	Avg.word/inst	-	-	-	10.5
TANGRAMS	#Interaction	4189	199	800	5188
	#Instruction	20945	995	4000	25940
	Avg.inst/inte	5	5	5	5
	Avg.word/inst	-	-	-	5.4
PROPARA	#Paragraph	391	43	54	488
	#Sentence	2639	290	373	3302
	Avg.sent/para	6.7	6.7	6.9	6.8
	Avg.word/para	61.1	57.8	67.0	61.4
RECIPES	#Paragraph	693	86	87	866
	#Sentence	6101	766	781	7648
	Avg.sent/para	8.8	8.9	9.0	8.8
	Avg.word/para	93.1	89.1	93.9	92.8

Table 10: Data statistics for ALCHEMY, SCENE, TANGRAMS, PROPARA, RECIPES.

Datasets	**Overlap Ratio (%)**					
	0%	20%	40%	60%	80%	100%
ALCHEMY	81.0	83.2	81.2	80.8	82.0	83.3
SCENE	65.5	63.1	59.6	62.6	63.1	64.6
TANGRAMS	62.8	63.3	62.8	64.3	62.8	63.3
PROPARA	72.5	70.7	72.7	70.0	70.8	72.8
RECIPES	61.7	59.5	59.3	62.4	60.8	59.8

Table 11: Downstream Performances on the validation sets of five datasets with respect to the overlap ratio. For ALCHEMY, SCENE and TANGRAMS, we report the 5utts denotation accuracy, while we report the F1 score on PROPARA and RECIPES.

Type (Percent)	Example	Environment (Goal State) Comparison
ALCHEMY		
Operation Correctness (68.3%)	**Environment (Initial State)** : 1:p | 2:r | 3:o **Instruction**: Empty out the first beaker, add the orange chemical to the red.	The difference lies in the third beaker, *o* (*w.o.* pre-training) versus _ (*w.* pre-training). Without pre-training, the model does not correctly understand the semantics of "add", i.e., removing the liquid from the third beaker.
Instruction Completeness (18.3%)	**Environment (Initial State)**: 1:o | 2:rr **Instruction**: Throw out one unit of the second beaker, pour the second beaker into first one.	The difference lies in the first beaker, *orr* (*w.o.* pre-training) versus *or* (*w.* pre-training). As observed, without pre-training, the model seems to ignore the instruction "throw out one unit of the second beaker".
Grounding Correctness (8.5%)	**Environment (Initial State)**: 1:y | 2:yyy | 3:yy **Instruction**: Pour out one part of the second yellow beaker.	The whole states are 1:y | 2:yyy | 3:y (*w.o.* pre-training) versus 1:y | 2:yy | 3:yy (*w.* pre-training). Without pre-training, the model does not find the correct beaker according to the instructions.
SCENE		
Operation Correctness (22.7%)	**Environment (Initial State)**: 1:og | 2:oo | 3:__ | 4:__ **Instruction**: The man in an orange shirt and green hat moves to the right end.	The difference lies in the second position, __ (*w.o.* pre-training) versus *oo* (*w.* pre-training). Without pre-training, the *oo* disappears for no reason, which indicates that the "move" operation cannot be performed correctly.
Instruction Completeness (15.2%)	**Environment (Initial State)**: 1:__ | 2:bo | 3:__ | 4:__ **Instruction**: A person in a yellow shirt enters from the right, the person in yellow takes the hat from the person in blue, the person in blue retrieves the hat from the person in yellow.	The difference lies in the second position and the third position, *b_*, *yo* (*w.o.* pre-training) versus *bo*, *y_* (*w.* pre-training). The model requires to swap hats twice, but without pre-training, only once performed, which indicates that one of the TRADE-HATS operations is ignored.
Grounding Correctness (62.1%)	**Environment (Initial State)**: 1:rg | 2:__ | 3:gr | 4:__ **Instruction**: A man in an orange shirt appears to the right of the man in a red shirt and green hat.	The whole states are 1:rg | 2:__ | 3:gr | 4:o_ (*w.o.* pre-training) versus 1:rg | 2:o_ | 3:gr | 4:__ (*w.* pre-training). Without pre-training, the model does not find the correct position according to the instructions.
TANGRAMS		
Instruction Completeness (52.7%)	**Environment (Initial State)**: 1:A | 2:C | 3:D **Instruction**: Delete the 3rd figure, swap the two figures, delete the 1st figure.	The state is 1:C (*w.o.* pre-training) versus 1:A (*w.* pre-training). Without pre-training, the model does not correctly understand the semantics of "swap", i.e. change the positions of two figures.
Grounding Correctness (47.3%)	**Environment (Initial State)**: 1:A | 2:E | 3:B | 4:C | 5:D **Instruction**: Delete the 3rd figure, delete the 4th figure.	The whole states is 1:A | 2:E | 3:D (*w.o.* pre-training) versus 1:A | 2:E | 3:C (*w.* pre-training). After performing the first instruction, the positions of each item have changed. When performing the second instruction, without pre-training, the model does not find the correct positions.

Table 12: The main types of the improvements by the execution-guided pre-training in the validation set of ALCHEMY, SCENE, and TANGRAMS.

<table>
<tr><th>Type (Percent)</th><th>Example</th><th>Environment (Goal State) Comparison</th></tr>
<tr><td></td><td></td><td>ProPara</td></tr>
<tr><td>Operation Correctness (12.0%)</td><td><b>Environment (Initial State)</b>: ent: algae | plankton | sediment; loc: ? | ? | seafloor
<b>Instruction</b>: Algae and plankton die. The dead algae and plankton end up part of sediment on a seafloor.</td><td>The whole predicted locations contains 2 items (<i>w.o.</i> pre-training) versus 3 items (<i>w.</i> pre-training). Without pre-training, the amount of the predicted locations is inconsistent with the entities, namely, the instructions cannot be performed correctly.</td></tr>
<tr><td>Instruction Completeness (28.0%)</td><td><b>Environment (Initial State)</b>: ent: bacteria | enzymes; loc: ground | bacterium
<b>Instruction</b>: Bacteria from the ground migrate to the plant material. Bacteria release enzymes onto the plant material.</td><td>The predicted location of enzymes is <i>bacterium</i> (<i>w.o.</i> pre-training) versus <i>plant material</i> (<i>w.</i> pre-training). The most potential reason is that the second instruction is ignored by the model without pre-training.</td></tr>
<tr><td>Grounding Correctness (60.0%)</td><td><b>Environment (Initial State)</b>: ent: silk | web; loc: - | -
<b>Instruction</b>: The spider picks a suitable place. The spider produces sticky silk from its abdomen.</td><td>The predicted location of silk is <i>spider</i> (<i>w.o.</i> pre-training) versus <i>abdomen</i> (<i>w.</i> pre-training). Without pre-training, the model does not find the most suitable location of the entity based on the instructions.</td></tr>
<tr><td></td><td></td><td>Recipes</td></tr>
<tr><td>Operation Correctness (9.1%)</td><td><b>Environment (Initial State)</b>: ent: scallion | cloves garlic | canola oil states: - | - | -
<b>Instruction</b>: First to go in the wok was the oil, scallions , and garlic .heat these ingredients until the garlic starts to turn brown.</td><td>The whole predicted locations contain 4 items (<i>w.o.</i> pre-training) versus 3 items (<i>w.</i> pre-training). Without pre-training, the amount of the predicted locations is inconsistent with the entities, namely, the instructions does not be performed correctly.</td></tr>
<tr><td>Instruction Completeness (21.8%)</td><td><b>Environment (Initial State)</b>: ent: green pepper | tomato | green onion states: - | - | -
<b>Instruction</b>: Cut tin foil into 12x16 inch rectangle. Place green pepper, tomato and green onion on lower half of foil sheet.</td><td>All of the predicted locations of the three entities are ? (<i>w.o.</i> pre-training) versus <i>foil</i> (<i>w.</i> pre-training). The most potential reason is that the second instruction is ignored without pre-training.</td></tr>
<tr><td>Grounding Correctness (69.1%)</td><td><b>Environment (Initial State)</b>: ent: salt | olive oil; loc: - | -
<b>Instruction</b>: Lightly grease large bowl and two loaf pans with olive oil.</td><td>The predicted location of olive oil is <i>bowl</i> (<i>w.o.</i> pre-training) versus <i>pan</i> (<i>w.</i> pre-training), which indicates that the model does not find the most suitable location of the entity based on the instructions without pre-training.</td></tr>
</table>

Table 13: The main types of the improvements by the execution-guided pre-training in the validation set of ProPara and Recipes.

<table>
<tr><th>Domain</th><th>Program</th><th>Initial State & Goal State</th><th>Description</th></tr>
<tr><td>Alchemy</td><td>Pour (Beaker (1), Beaker (2, g))</td><td>1:rr | 2:gg | 3:g | 4:ooo
1:_ | 2:gg | 3:grr | 4:ooo</td><td>Pour the liquid from the first beaker into the second green beaker.</td></tr>
<tr><td>Scene</td><td>Person (2, r); Hat (2, y)</td><td>1:__ | 2:__ | 3:__ | 4:__ | 5:ob
1:__ | 2:ry | 3:__ | 4:__ | 5:ob</td><td>A person with a red shirt and a yellow hat appears on the second position.</td></tr>
<tr><td>Tangrams</td><td>Remove (2); Insert (4, B)</td><td>1:A | 2:B | 3:C | 4:D | 5:E
1:A | 2:C | 3:D | 4:B | 5:E</td><td>Remove the second figure, and add it back into the fourth position.</td></tr>
<tr><td>ProPara</td><td>Move (bacteria, cell, bladder)</td><td>ent : bacteria | sickness
loc : cell | -
ent : bacteria | sickness
loc : bladder | -</td><td>Move bacteria from cell to bladder.</td></tr>
<tr><td>Recipes</td><td>Create (beef, oven)</td><td>ent : beef | pepper
loc : - | -
ent : beef | pepper
loc : oven | -</td><td>Beef appears in the oven.</td></tr>
</table>

Table 14: Example programs, initial states, and goal states for each domain.