ECAI 2024
U. Endriss et al. (Eds.)

doi:10.3233/FAIA240974

Planning with Logical Graph-Based Language Model for Instruction Generation

Fan Zhang [a,1], **Kebing Jin** [b,2] **and Hankz Hankui Zhuo** [a,*]

[a]School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou, China
[b]State Key Laboratory of Public Big Data, Guizhou University, Guiyang, China

Abstract. Despite the superior performance of large language models to generate natural language texts, it is hard to generate texts with correct logic according to a given task, due to the difficulties for neural models to capture strict logic from free-form texts. In this paper, we propose a novel graph-based language model, `Logical-GLM`, to extract strict logic from free-form texts and then infuse into language models. Specifically, we first capture information from natural language instructions and construct logical probability graphs that generally describe domains. Next, we generate logical skeletons to guide language model training, infusing domain knowledge into language models. At last, we alternately optimize the searching policy of graphs and language models until convergence. The experimental results show that `Logical-GLM` is both effective and efficient compared with traditional language models, despite using smaller-scale training data and fewer parameters. Our approach can generate instructional texts with more correct logic owing to the internalized domain knowledge. Moreover, the search of logical graphs reflects the inner mechanism of the language models, which improves the interpretability of black-box models.

1 Introduction

Instruction generation aims to generate a course of actions to complete the given task. It is an open issue which requires both sophisticated and knowledge-intensive reasoning abilities of models [1, 2]. Inspired by the reasoning capabilities of large language models (LLMs), works have been committed to investigating how LLMs guide the instruction generation in specific domains [3, 4]. However, the LLMs can internalize only the general world knowledge rather than domain-specific logic, which is required for instruction generation. As mentioned by Valmeekam et al. [5], even in a seemingly simple environment like BlocksWorld, LLMs are quite ineffective in generating reasonable instructions.

An interesting instance is utilizing LLMs to generate instructional texts to guide robots in cooking [6]. In detail, an example generated recipe for task "frog legs with lemon and thyme" is: *Gently melt the lemon; Add the oil; A little salt and a pinch of sugar; Bring to the boil and then cover and cook over a medium heat for 25 minutes; Stirring occasionally; Until the frogs are soft; Remove from the heat; Leave to cool slightly and stir in the remaining lemon juice; Place the loaf of bread on a work surface sprinkled with flour; Sprinkle with a little flour and quickly knead together to make a smooth;*

The above example shows that the generated recipe contains many actions whose **preconditions are not satisfied**, which hinders robots from completing the action sequence in the real world. For instance, a robot can execute *"melt the lemon"* only if *"puts lemon into the pan"* has been carried out. Next, the above example **generates incomplete instructions**, e.g., lacking corresponding operation objects. For example, the second sentence, *"Add the oil"* should be completed as *"Add the oil into the pan"*. Furthermore, the generated actions gradually **deviate from the given task**. In the above example, all of the actions from the ninth sentence to the end seem to be relevant to "bake a cake" rather than the given task. In total, the above phenomena prove that the texts generated by neural networks lack strict logic.

To augment the reasoning abilities of LLMs for generating reasonable instructions, works have been proposed to leverage planning thought to ensure strict logic. Wei et al. [7] generate a chain of thought (CoT) to improve the ability of LLMs to perform complex reasoning. Similarly, Yao et al. [8] leverage the tree of thought (ToT) to perform deliberate decision-making by considering multiple different reasoning paths and self-evaluating choices to decide the next course of action. Both CoT and ToT transfer the planning thought into LLM generation to ensure strict logic. The strict logic requires the support of knowledge. Though LLMs internalize general world knowledge and behave well on a range of common-sense tasks, they behave poorly on domain-specific tasks, which require domain-specific knowledge. Domain-specific knowledge applies to a specific domain, where common-sense knowledge is the general rules between domains. In some domain-specific tasks, the common-sense knowledge may not include all the required domain logic. Reality applications, e.g. industrial environment, require specific domain knowledge, which may not be included in the training common-sense dataset. Inspired by this, we are curious if it is possible to import domain knowledge for supporting AI Planning in a domain-specific generation. By doing this, we aim to answer the question: **how domain knowledge can be better mixed with LLM knowledge for supporting AI Planning**.

Indeed, attempts have been made to infuse domain knowledge into LLMs [9, 10, 11]. For example, Wang et al. [10] propose a novel approach named Hierarchical Concept-Driven Language Model (HCDML) to infuse knowledge into LMs. HCDML learns a Hierarchical Language Model for implementing planning thought and a Recurrent Conceptualization-enhanced Gamma Belief Network for internalizing domain logic. However, the internalized logic is not strict since the training data can contain redundant or even

* Corresponding Author. Email: zhuohank@gmail.com
[1] Email: zhangf356@mail2.sysu.edu.cn
[2] Email: kbjin@gzu.edu.cn

wrong texts and hinder HCDML from supporting planning thought when generating texts. This indicates extracting strict logic from texts is difficult. To do this, we represent domain knowledge in the form of symbolic logic, namely PDDL (Planning Domain Description Language) [12]. Then, we construct a logical graph to internalize the strict logic of extracted structural expressions.

The core steps of our `Logical-GLM` are as follows. Firstly, we convert natural language instructions into action sequences in the form of PDDL, so as to eliminate redundant and incorrect information. Actions in classical planning consist of preconditions and effects, where the interlaced predicates form complex logical relations in the domain. Therefore, we consider that the sequential order of actions implies the domain logic, which potentially reflects logical constraints and updating between actions. Thus, we then filter and leverage converted instructions to extract action pairs with higher frequency than the threshold we set and construct logical probability graphs using structuralized action pairs. Meanwhile, we train language models using converted instructional texts. Finally, the logical graph and language model guide each other to make final decisions. Specifically, in the search process, the probability graph generates logical skeletons towards the target task node under the guidance of LMs. The target-oriented search can ensure the actions are always related to the task. The precedence order of action nodes ensures the preconditions of actions can be satisfied.

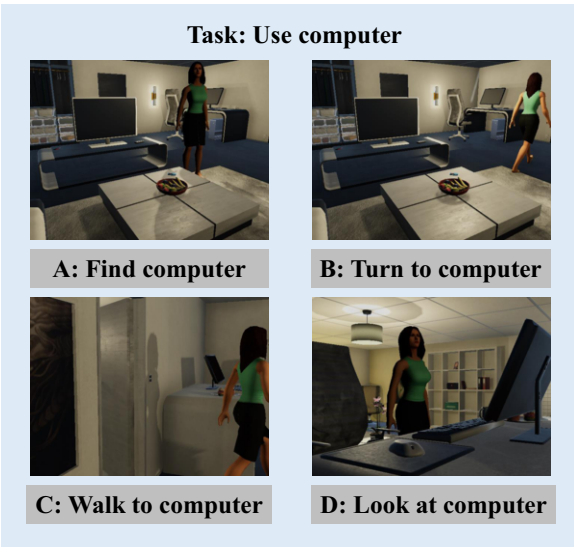


Figure 1. Samples generated by `Logical-GLM` in VirtualHome domain

2 Problem Definition

Let T_x denote the task for completion, where *executable* and *correct* instructional texts are expected to be generated (i.e., Figure 1). The label texts and generated texts are denoted by $\hat{T}_y$ and T_y, respectively. In this way, our training data can be defined as a series of tuples Φ, where each tuple consists of $\langle T_x, \hat{T}_y \rangle$, where T_x and $\hat{T}_y$ denote the task and corresponding label instructional texts. Given a task that does not exist in training data as input, the goal of the proposed model is to generate simpler and more logical instructional texts with a smaller amount of parameters compared to large language models. Specifically, the generated instructional texts should be executable and correct to ensure the completion of the given task. We will discuss the detailed evaluation metrics in the experimental section.

3 Logical Graph-based Language Model

When generating instructional texts by LLMs, there are several major challenges: (1) The forms of texts are variant and different texts may express the same domain logic. (2) Texts contain some redundant information, which influences logic extraction. (3) The amount of actions and objects is large, which adds difficulties to logic extraction. (4) Gradient explosion can happen for fine-tuning LLMs. To tackle these, we propose our `Logical-GLM` which includes:

1. A logical graph that learns core logic from action traces of each task in a specific domain;
2. A language generation model that both learns logic from the constructed logical graph and generalizes well to unseen tasks;
3. An EM-style framework where logical graphs and language models guide each other to internalize domain logic.

The overall algorithm is presented in Figure 2. Our model is composed of four components: Logical Graph Construction Preparation (blue part), Logical Probability Graph Search (gray part); Language Model Training, and Logical Graph Optimization (green part).

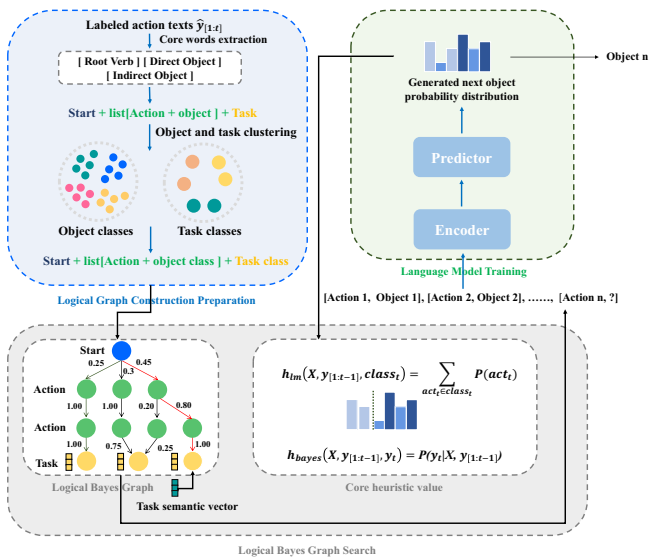


Figure 2. An overview of the proposed `Logical-GLM`

3.1 Logical graph construction preparations

To construct logical graph, we design a preprocessing module, which refines sophisticated logical relations in natural language instructions (upper left part of Figure 2). In detail, the logical relation between action A and action B means that action B can only be executed when action A has been committed.

If we directly adopt natural language instructions to mine rules, redundant and incorrect information may affect relation extraction. Therefore, for extracting refined logical relations, we first use semantic parser spaCy[3] to extract the core verbs and objects in the domain. Then, we leverage pre-trained RoBERTa model to calculate the

[3] https://spacy.io/

semantic vectors of extracted objects and tasks [13]. After that, we use K-means to cluster extracted objects and tasks according to the the semantic vectors. An object detector then substitutes objects with clustered object class. Since policy selection considers multiple previously executed actions in reinforcement learning, we then analyze relations between previous $t(t \in Z^+)$ actions with current action for extracting logical relations. We calculate the average number of preconditional actions for all the actions in the training data and take the rounded value as t. The frequency of action pair within t time steps is calculated and compared with the threshold $\epsilon(\epsilon \in R, 0 \leq \epsilon \leq 1)$. The threshold ϵ depends on the specific action (Equation 1). For each action, we calculate the frequency mean of previous actions plus the unbiased estimation of frequency variance within t time steps. The action pairs with higher frequency than ϵ are retained. In this way, we finally acquire refined logical rules for constructing logical graphs.

$$\epsilon = \bar{p} + \frac{\sum_{i=1}^{t} (p_i - \bar{p})^2}{k}, k \geq 1, p \in \mathbb{R} \tag{1}$$

where k is the number of all previous adjacent actions of the current action. p_i denotes the frequency of each previous action and $\bar{p}$ is the average frequency of all previous actions.

3.2 Logical probability graph construction

After acquiring refined logical rules, we then construct a logical probability graph to indicate the logical relations implied in the texts. However, the natural language logical rules may contain much redundant information, so we first structuralize refined logical rules using syntax from AI planning [14, 15, 16]. Consistent with Planning Domain Definition Language (PDDL), we define the lifted action node as $action(?x{-}type)$ to reserve core verbs and objects, where $action$ denotes core verb and $?x - type$ denotes the object type without instantiation [12]. The lifted action node can be visited and then grounded into $action(instance)$. For example, $grab(?x - fruit)$ can be grounded as specific $grab(apple)$. Following the same source node, we add lifted action nodes and task type node of each action sequence in order. If the path from source node to the current node exists, we just add the next node following the current node. In this way, we internalize the domain logic into the graph skeleton. Motivated by the wide use of bayes networks, we endow each edge with transition probability to learn causal relationships [17]. In detail, we represent the transition probability using the corresponding frequency of action pair in training data. In this sense, we construct a logical probability graph which contains refined domain logic.

3.3 Logical graph search and heuristics computation

The language model has better generalization abilities than traditional planning [18]. However, LLMs only work on simple reasoning tasks and fail to solve sophisticated ones [5].

Therefore, we design a logical probability graph search module, which generates instructional skeletons under the guidance of both logical probability graphs and language models. To guide the search across the constructed graph, we design a heuristic value calculation module. The total heuristic value is calculated based on four factors:

$$\mathcal{H} = -h_{dis}(.) - \omega_1 h_{len}(.) + \omega_2 h_{bayes}(.) + \omega_3 h_{lm}(.) \tag{2}$$

where $\mathcal{H}$ stands for the total heuristic value. h_{dis}, h_{len}, h_{bayes} and h_{lm} stand for distance, expected program length, bayes, and language model heuristics. We define the program length as the number of sentences. ω_1, ω_2 and ω_3 are positive hyper-parameters. The detailed heuristics calculation is formulated as follows and the candidate node with the highest heuristics can be selected :

$$h_{dis}(y_n, x) = \begin{cases} 0, & if \text{ path\_exist}(y_n, x) \\ \infty, & \text{otherwise} \end{cases} \tag{3}$$

where h_{dis} implies whether the current node can reach the target task type node. y_n denotes action type node at current step and x the target task type node, respectively. path_exist(y_n, x) denotes whether exists one path from the current node to the target task type node. If there exists no path between y_n and x in the logical graph, this means the current action node is probably irrelevant to the task. When searching the graph, we give priority to nodes which can reach the target task node.

$$h_{len}\left(y_{[1:n]}, x, \hat{l}\right) = \frac{min(|avg\_len(y_n, x) + n - 1 - \hat{l}|, \hat{l})}{\hat{l}} \tag{4}$$

where $y_{[1:n]}$ denotes the previously $n-1$ selected action type nodes concatenated a current candidate action type node. y_n denotes the current candidate node. $\hat{l}$ stands for the expected number of instructional sentences. $avg\_len(.)$ calculates the average length of all paths from the current candidate node to the target node. We use current steps plus $avg\_len(.)$ as the estimated length of our generated action sequence. Then, we calculate the difference between our estimated length and the expected length. The shorter the difference between our estimated instructional length and the ideal length, the larger the value of $-\omega_1 h_{len}(.)$. In this way, the length of our instructions can be adjusted according to the expected length.

$$h_{\text{bayes}}\left(y_i \mid y_{[1:n-1]}, x\right) = \frac{P\left(y_i \rightarrow x\right)}{\sum_{y_j \in \text{ sons }\left(y_{n-1}\right)} P\left(y_j \rightarrow x\right)} \tag{5}$$

where $\rightarrow$ stands for the probability sum of all possible paths from the visited path to the target node. $sons$ denote the action nodes which can be visited from the last action node. Based on the already visited action nodes and target type nodes, we calculate the score of one action node among all candidates.

$$h_{lm}(\overline{y}_n | x, \overline{y}_{[1:n-1]}) = \max_{\overline{y}_n \in grd(y_n)} P(\overline{y}_n | x^{tar}, \overline{y}_{[1:n-1]}) \tag{6}$$

where $\overline{y}_n$ stands for all possible grounded actions at the current step. For instance, $grab(?x{-}fruit)$ can be grounded into $grab(mango)$, $grab(bananas)$ and other possible specific actions. $\overline{y}_{[1:n-1]}$ stands for grounded action node sequences at previous $n-1$ time steps. h_{lm} calculates the probability of generating one possible grounded action given the texts of task and all previously grounded actions. The detailed probability calculation for each grounded action by the LM can be seen in Equation (7).

$$P = \gamma\left(T_x, T_{\overline{y}_{[1:n-1]}}; T_{y_n}\right) \tag{7}$$

where γ denotes the language model for generating the next action. In detail, the texts of the task T_x and previous actions $T_{\overline{y}_{[1:n-1]}}$ serve as the input, and we acquire the probability of LM generating a grounded action T_{y_n}.

We then use heuristics to guide the logical graph search. As illustrated in Figure 3, the graph search algorithm moves towards the task goal and the heuristics guide the node selection. In detail, we first select the target task type node (yellow nodes) which has the highest semantic similarity with the current task. Then, we start from the source node (blue node) and set it as the current node. At each step, we calculate the heuristics of all successor nodes from the current node. For instance, the heuristic value of Action 1 is $v_0 + p_0 + q_0 + l_0$, which

corresponds to four heuristics defined in Equation (2). The nodes with the highest heuristics (green nodes) are selected as the current nodes and grounded into $action(instance)$. The node selection continues until reach the target task type node. Finally, we concatenate all grounded actions and output the generated instructional texts.

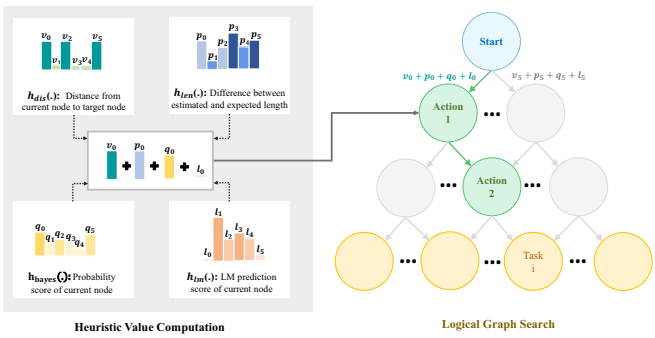


Figure 3. An example of domain-specific logical graph search

3.4 LM training and logical graph optimization

The generated instructions from logical graphs internalize domain logic. We thus use both generated and label instructions to guide language model training. The trained language model also guides the logical graph search. In this way, logical graphs and language models guide each other to learn domain logic and make policies.

Specifically, each input to the language model is comprised of task description x and label instructional text $\hat{y}$. Since both label instructions and generated instructional texts contain domain logic, we define the cross-entropy loss function $\mathcal{L}(\theta)$ as follows:

$$\mathcal{L}(\theta) = -\mathcal{H}\left(f(y_{[1:t]}), f(\overline{y}_{[1:t]})\right) - \mathcal{H}\left(f(\hat{y}_{[1:t]}), f(\overline{y}_{[1:t]})\right) \quad (8)$$

where $\mathcal{L}(.)$ is the loss function. $\mathcal{H}(.)$ stands for the cross entropy between two vectors. $f(.)$ is the embedding function. $\hat{y}_{[1:t]}$ is the label instructions and $y_{[1:t]}$ is the generated skeletons from the logical graph. We use the above loss function to update parameters by stochastic gradient descent (SGD) [19]. In this sense, the parameters of language model are updated to make the language model internalize logic contained in both label instructions and generated instructional texts from the logical graph. The trained language model then guides the logical graph search. In this way, language model and logical graph guide each other and better internalize domain logic.

3.5 Overview of our `Logical-GLM`

The overall algorithm is presented in Algorithm 1. We first initialize parameters θ of the language model randomly (Line 1). After that, we use the processed annotated task tuples $\Phi =< x, \hat{y} >$ to construct a logical graph (Line 2). We then search the logical graph to generate instructional texts under the guidance of the language models (Line 5). After that, we update parameters θ of the language model by optimizing $\mathcal{L}(\theta)$ (Line 6). We repeat the above procedure until the stop requirement is satisfied and output the language model parameters θ.

Algorithm 1 EM-style Optimization Algorithm

Input: Annotated task tuples $\Phi =< x, \hat{y} >$

Output: Logical Probability Graph and well-trained language model

1: Initialize parameters θ of the language model randomly;
2: Construct the logical graph using annotated task tuples $\Phi =< x, \hat{y} >$;
3: **for** iteration ← 1...R **do**
4: Randomly Pick annotated tasks $< x, \hat{y} >$;
5: Generate instructional texts by logical graph search with language model guidance (Equation (3));
6: Update θ using both annotated task tuples and generated instructional texts (Equation (8));
7: **end for**
8: **return** θ

4 Experiments

4.1 Datasets

To evaluate the effectiveness of the proposed model, we investigate the textual reasonability, executability and correctness of generated texts. To evaluate the executability, we need an environment where textual interactions are supported. Most existing embodied environments except VirtualHome are unsuitable. We thus choose VirtualHome as our environment [20].

In total, there are 5048 pieces of available instructional texts for 202 different tasks. A vivid example can be seen in Figure 4, where redundant information and noises may exist. We divide the tasks into training tasks and testing tasks with a ratio of 3:1. In detail, all instructional texts of 152 tasks are taken as training data, and texts of the remaining 50 tasks as testing data. Then we randomly take 20% training data for validation to avoid over-fitting.

<Task>
Set up table
<Generated Actions>
Step 1: Walk to dining room
Step 2: Walk to cupboard
Step 3: Find cupboard
Step 4: Open cupboard
Step 5: Find plate
Step 6: Grab plate
Step 7: Find plate
Step 8: Grab plate
Step 9: Find table
Step 10: Walk to table
Step 11: Put plate on table
Step 12: Put plate on table

<Task>
Write an email
<Generated Actions>
Step 1: Walk to home office
Step 2: Find computer
Step 3: Turn to computer
Step 4: Look at computer
Step 5: Walk to computer
Step 6: Find chair
Step 7: Sit on chair
Step 8: Find keyboard
Step 9: Grab keyboard
Step 10: Find mouse
Step 11: Grab mouse
Step 12: Type on keyboard

Figure 4. An example of training data

4.2 Baselines and Criteria

Baselines We consider the following large language models to characterize the performance of our `Logical-GLM` even with quite a low parameter amount.

1. **Vanilla GPT2 family:** GPT2 is a transformers model pre-trained on a very large corpus of English data. Given the task, Vanilla GPT2 can generate corresponding instructional texts. In detail, we take the Chain-of-Thought(CoT) method to prompt GPT2. The task description and history actions are taken as input and GPT2 generates the next action. We use the pretrained GPT2-small, GPT2-medium, GPT2-large and GPT2-xl with 124M, 355M, 774M, and 1.5B parameter amounts, respectively.
2. **Fine-tuned GPT2 family:** We finetune the pre-trained GPT2 family using the same label texts as `Logical-GLM` for fair compar-

ison. In detail, we follow the CoT to generate training data, where the task description and history are labeled with the next action.

3. **ChatGPT(GPT-3.5):** ChatGPT is an artificial intelligence chatbot developed by OpenAI which owns a 175 billion parameter amount. As the latest milestone in interactive large language models, ChatGPT has exhibited human-level performance on various professional and academic benchmarks.
4. **GPT4:** GPT4 is a multimodal large language model which owns a 1.8 trillion parameter amount. GPT4 is more creative and collaborative than ever before. Compared with previous models which include GPT-3.5, GPT-4 has better human-like conversations and provides more accurate results.
5. **SayCanPay:** SayCanPay is a model that combines the power of LLMs and heuristic planning by leveraging the world knowledge of LLMs and principles of heuristic search [21]. In detail, SayCanPay leverages the world knowledge of Say model and trains Can and Pay models using expert data. After that, it uses greedy and beam search to generate actions. Thus, we re-train the can and pay models and then test using greedy and beam search, separately.

Evaluation Criteria Specifically, we follow Huang et al. [22] to evaluate executability and correctness. In addition, we use BLEU and ROUGE from the NLP field to evaluate the reasonability.

Executability measures whether all actions in an action plan satisfy the common-sense constraints of the environment. The VirtualHome maintains the state graph of the current environment and checks the preconditions before executing each action. Specifically, each action (e.g. grab milk from the fridge) can be executed only if the corresponding pre-conditions (e.g. the fridge is open) are satisfied. One script is executable if and only if all preconditions of actions in the script can be satisfied. Then, we calculate the average proportion of executable scripts among all generated scripts for executability.

Correctness evaluates whether one script can ensure task completion. The complex task often contains one or more goal states that require to be satisfied, which is difficult to judge by automation metrics. Thus, we ask human evaluators to judge whether one script completes the given task by their experiences. Then, we calculate the proportion of correct scripts for correctness.

Unlike embodied environments like the OFFICEWORLD domain and Montezuma's Revenge domain [23, 24], the ambiguous nature of natural language task description makes it hard to find a perfect standard for the correctness of generated instructions. Referred to Huang et al. [22], we thus commit human evaluation to measure whether the generated instructions ensure the task completion from human common sense. We randomly select 20 tasks in the testing datasets. Then we invite 50 human annotators to measure whether the generated instructions are correct. The correctness rate of texts is calculated by averaging the correctness ratio from different human annotators.

BLEU measures the similarity between generated texts and reference texts based on n-gram (Equation (9)) [25]. In detail, BLEU detects the common segments with different lengths between generated texts and reference texts. The 1-gram BLEU detects the same word and evaluates how many redundant words the generated instructions have. The n-gram (≥ 1) detects textual segments and evaluates how the generated actions submit to actions in label texts.

$$BLEU_n = \frac{\sum_{c \in texts} \sum_{n_{gram} \in c} Count_{clip}(n_{gram})}{\sum_{c' \in texts} \sum_{n_{gram'} \in c'} Count\left(n_{gram'}\right)} \quad (9)$$

where $texts$ represents the sentences in generated instructional texts. $Count_{clip}(n_{gram})$ and $Count\left(n_{gram'}\right)$ denote the number of each n_{gram} and $n_{gram'}$ from instructional texts appear in both and candidates, respectively.

ROUGE measures the similarity between generated texts and reference texts (Equation (10)) [26]. Different from BLEU, it can consider the overlapping of n-gram, the longest common subsequence, and weights of different words.

$$ROUGE_n = \frac{\sum_{c \in texts} \sum_{n_{gram} \in c} Count_{clip}(n_{gram})}{\sum_{c' \in labels} \sum_{n_{gram'} \in c'} Count\left(n_{gram'}\right)} \quad (10)$$

where $labels$ represents the sentences in label texts. $Count_{clip}(n_{gram})$ and $Count\left(n_{gram'}\right)$ denote the number of each n_{gram} and $n_{gram'}$ from instructional texts appear in both and only references, respectively.

In this paper, we do not consider the optimality of solutions, where the optimal solution of a task is defined as the shortest action sequence that can ensure the task completion. This is because it is challenging to generate the optimal solution even in AI Planning. Thus, the expert data we use are often inoptimal and may contain redundant actions. Limited by such training data, it is hard to ensure the optimality of solutions.

4.3 Experimental Results

Our experiments evaluated the text quality and practicability of generated instructional texts. We will examine `Logical-GLM` in the following aspect:

1 We first evaluate `Logical-GLM` on classic NLP metrics like BLEU and ROUGE, which measure the precision and recall of generated instructional texts with label texts and reflect the text quality and reasonability to some degree.
2 We then evaluate `Logical-GLM` on executability and correctness of instructional texts. We judge the executability and correctness by the completion of the task and human, respectively.
3 Additionally, we evaluate the effects of language model guidance on `Logical-GLM` by modifying training epochs and evaluating the generated instructions.
4 We adjust the weights of heuristics in graph search and aim to see the impacts of heuristics and find the optimal hyperparameter pair.
5 Shorter instructional texts often have higher executability, which may disturb the evaluation of text quality. To validate the robustness of our model, we finally evaluate the executability of generated instructional texts with varying program lengths.
6 Finally, we commit a case study by analyzing instructions of one specific task generated from `Logical-GLM` and all baselines.

Text quality of generated instructional texts

The BLEU and ROUGE score of generated instructional texts with label texts can reflect the text quality to some degree. In detail, we leverage ground truth instructional texts provided by [22]. We then compare BELU and ROUGE score among `Logical-GLM` and all fine-tuned baselines. Table (1) shows BLEU and ROUGE results from different language models. From Table (1), we can observe that our `Logical-GLM` outperforms all baselines and achieves state-of-the-art performance on both BLEU and ROUGE score with the least parameter amount, which demonstrates the reasonability and submissiveness to label texts of our generated instructional texts.

Executability and correctness of generated instructions

The final goal of our `Logical-GLM` is to complete tasks in sophisticated open environments. We thus compare the executability and correctness of generated instructional texts. In this experiment,

Table 1. The comparison of BLEU and ROUGE score.

Model	Parameter	BLEU	ROUGE
Logical-GLM	**35.3M**	**0.40**	**0.26**
Tuned GPT2-small	124M	0.05	0.17
Tuned GPT2-medium	355M	0.05	0.15
Tuned GPT2-large	774M	0.05	0.16
Tuned GPT2-xl	1.5B	0.04	0.17
SayCanPay (Beam)	218M	0.04	0.15
SayCanPay (Greedy)	218M	0.03	0.16

executability indicates each action from instructional texts is effective and feasible. We also judge correctness by humans and it implies instructional texts can ensure the completion of tasks. Specifically, we use the same questionnaires as Huang et al. [22], but we take the average correctness of evaluated tasks over all human evaluators for each model. The results together with the average program lengths are shown in Figure 5. Considering the great expenses of human evaluation, we randomly pick 20 tasks from evaluated tasks for human evaluation. In Figure 5, we see that our model performs better than all comparison models including ChatGPT. It is not surprising because our Logical-GLM generates instructional texts based on domain-specific logical graph. Also, we note that GPT2-small has the highest executability among all models except our Logical-GLM. The reason we believe is that the program length of instructional texts from GPT2-small is quite small than all other models and shorter instructional texts tend to have higher executability. In comparison, the program length of instructional texts from our Logical-GLM is between those of GPT2-medium and GPT2-large but still performs better than almost all comparison models. Moreover, it can be observed that the fine-tuned GPT2 generates inexecutable instructional texts even if the original GPT2 small produced some executable output. We attribute this to that the scarce training data and finetuning LLMs may damage the safety strategies of LLMs. Note that GPT4 gets higher correctness but lower executability compared to our Logical-GLM. The higher correctness is not surprising because GPT4 has 1 trillion parameters and the enormous training corpus accidentally contains the required world knowledge known by humans. However, GPT4 may lack the required strict domain logic, resulting in lower executability. Moreover, our Logical-GLM surpasses both the beam search and greedy search of SayCanPay. The reason we believe is that SayCanPay essentially internalizes domain logic using language models, which calls for sufficient expert data. Intuitively, our Logical-GLM works better for generating instructional texts when domain-specific training data is scarce.

Influence measure of language model guidance

To evaluate the effects of language model guidance on Logical-GLM, we vary the training epochs of language models for different quality of language model guidance. We compare generated texts from Logical-GLM on evaluated tasks with varying quality of language model guidance. As shown in Figure 6(a), both the BLEU score and ROUGE score rise first and then decline with the improvement of training epochs. The same trend of executability with varying quality of language models is also depicted in Figure 6(b). The early rising trend shows that our language model guidance influences Logical-GLM and higher training epochs indicate better guidance. Although higher epochs indicate better language model guidance, we should note that overlarge training epochs result in over-fitting, which illustrates the latter declining trend. Note that the over-fitting is unavoidable since our lightweight Logical-GLM applies to domains with few expert samples. Thus, we randomly take 20% training data for validation and we

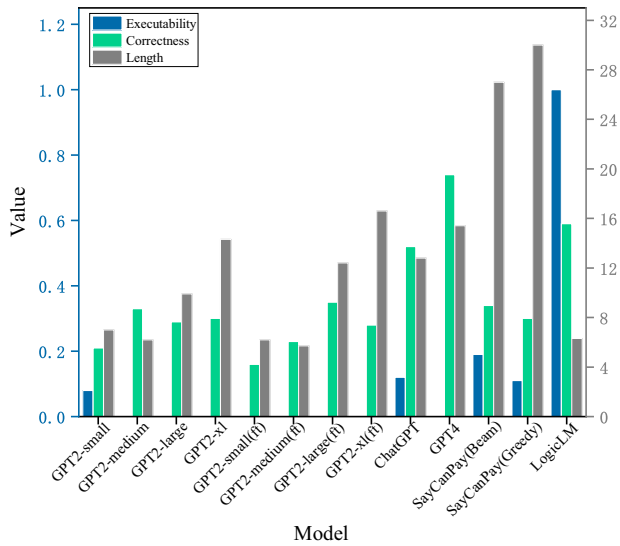
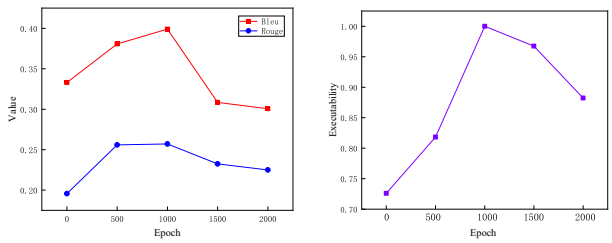


Figure 5. Quality measure of instructional texts.

leverage the effects of validation samples to break off training in advance. In Figure 6(b), we also see that the executability is still high at epoch 0. This is not surprising because our Logical-GLM is also guided by bayes heuristic value, which contains domain logic.



(a) BLEU and ROUGE (b) Executability

Figure 6. The text quality trends with varying epochs

Executability with respect to various hyper-parameters

To see the impact of different hyper-parameters, we evaluate the executability of generated instructional texts on evaluated tasks with respect to different weights of heuristic values(i.e., w_1, w_2, and w_3 in Equation (2)). The detailed heatmap about how hyper-parameters influence executability is depicted in Figure 7. We can fix the value of one axis and study how executability changes with another axis value. In most cases, we can see that the executability of generated instructional texts drops with w_1 increasing. A big value of w_1 means to focus on generated instructional text length and thus ignore text logic. When $w_3 : w_2 = 100 : 1$, the executability of generated texts is the highest for the same $w_1 : w_2$. The reason we believe is that the heuristic value $h_{lm}(.)$ is much smaller than $h_{bayes}(.)$. The $h_{lm}(.)$ can have considerable effects as $h_{bayes}(.)$ only when the weight of $h_{lm}(.)$ is much higher than that of $h_{bayes}(.)$.

Controllable program length study

We next evaluate controllable program length ability of our model and the performance with regard to the answer length. Since the graph search uses the expected program length as heuristic guidance, the program length of generated instructions is human controllable. In Figure 8, we see that the program length (blue lines) truly changes with different program length. However, the generated

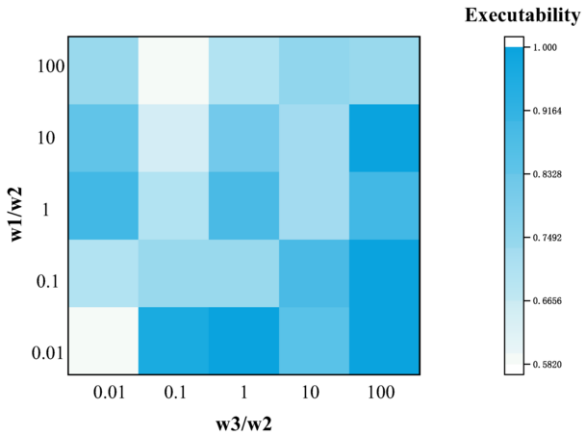


Figure 7. The heatmap of executability with different hyper-parameters

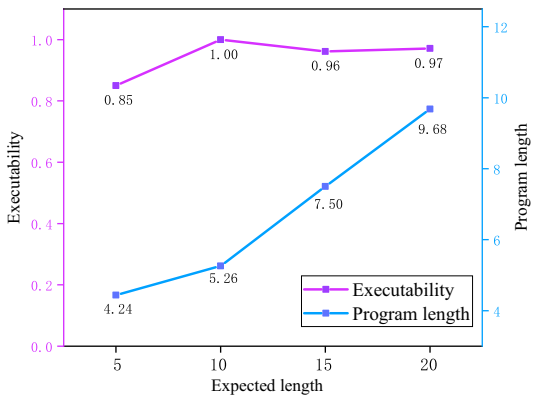


Figure 8. The model robustness for varying expected program length

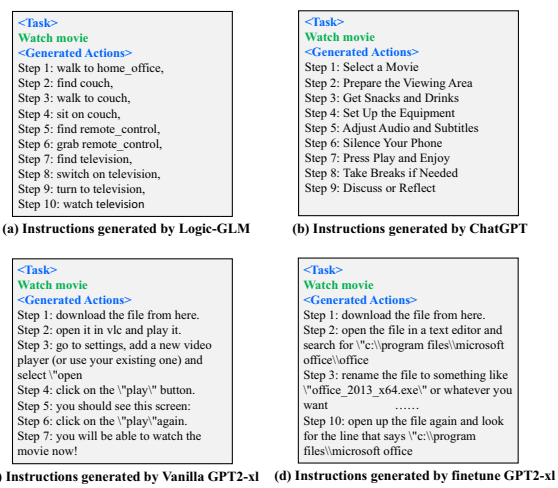


Figure 9. A case study on task "Watch movie"

program length change is not obvious. The small change range is because the training instances for constructing the logical graph have short program lengths and there exist no long paths in the logical graph. To validate the robustness of our Logical-GLM, we also compare executability of generated program length as the expected program length changes. From purple lines in Figure 8, we can observe that the executability soars as the program length increases. In the future, we hope to construct logical graph with longer paths and further explore the program length robustness of our model.

Case study on task "Watch movie"

To vividly demonstrate the difference between generated instructional texts from different language models, we analyze the outputs of Logical-GLM and other language models. In detail, we choose ChatGPT, both Vanilla and fine-tuned GPT2-xl which has the higher parameter amount among GPT2 family. We sample the task "Watch movie" and demonstrate instructions from different models in Figure 9. It is intriguing to note that our model can handle the task well.

From Figure 9, we can observe that both our Logical-GLM and ChatGPT can generate reasonable instructional texts relevant to the given task, which indicates that our Logical-GLM can achieve comparative performance as ChatGPT with the guidance of domain-specific logical graph. It is obvious that the instructional texts generated from our Logical-GLM are simple and have higher consistency between sentences. The actions generated by ChatGPT ignore the environmental constraints owing to a lack of domain knowledge, e.g., the generated action may not exist in the VirtualHome, which is inexecutable. Moreover, we can observe that both Vanilla GPT2-xl and Finetune GPT2-xl generate more sophisticated texts which lack consistency and include many non-critical objects in the specific task.

5 Related Work

Along with rapid advances in language models, there has been a greater interest in uncovering the reasoning abilities of such models [27, 28]. The research on reasoning includes two parts. A significant part is to extract logical rules from texts using language models. For example, Cresswell et al. [29, 30] use extracted rules to construct domain models for planning. Also, Huang et al. [22] directly use the pre-trained LLMs for generating logical instructional texts. Li claims that LLMs possess simple reasoning abilities [31]. With respect to AI planning, there have been lots of works on integrating AI planning with natural language processing (c.f. [32, 33, 34]). Large language models, however, fail to reason in some special or sophisticated scenes [5]. In this work, we aim to combine language models with AI planning for augmenting the logical reasoning abilities.

A variety of works have been studied in capturing domain logic and augmenting the reasoning abilities of models [35, 36, 37]. Song et al. [38, 39] represent domain logic with knowledge graph and then guide the text generation. Also, Zhang et al. [40] use a graph to represent relationships between events and personas for persona-guided text generation. Similarly, our approach leverages both AI planning and graph structure for quick domain knowledge capturing. However, the graph structure and language models in our approach guides each other to make logical policies in specific environments.

6 Conclusions

In this paper, we propose a novel approach Logical-GLM to generate logical skeletons which can be infused into language models with the smallest training expenses. In detail, the language model and logical graph guide each other to internalize the domain logic. In addition, the search process generates skeletons according to heuristics and indirectly explains the policy of language models to generate texts. We show that our Logical-GLM can be used in novel scenes of the same domain. The generalizability of the lightweight model towards scenes of different domains deserves further exploration. In the future, it would be interesting to investigate the integration of planning model learning [41, 42] and plan recognition techniques [43, 44] for interpretably generating instructions of high-quality.

Acknowledgements

This research was funded by the National Natural Science Foundation of China (Grant No. 62076263) and the Foundation of Guizhou University (Grant No. X2024048 and No. X2024076).

References

[1] Jonathan Francis, Nariaki Kitamura, Felix Labelle, Xiaopeng Lu, Ingrid Navarro, and Jean Oh. Core challenges in embodied vision-language planning. *Journal of Artificial Intelligence Research*, 74:459–515, 2022.

[2] Xiaohan Wang, Wenguan Wang, Jiayi Shao, and Yi Yang. Lana: A language-capable navigator for instruction following and generation. In *CVPR*, pages 19048–19058, 2023.

[3] Geunwoo Kim, Pierre Baldi, and Stephen McAleer. Language models can solve computer tasks. *Advances in Neural Information Processing Systems*, 36, 2024.

[4] Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. In *ICRA*, pages 9493–9500. IEEE, 2023.

[5] Karthik Valmeekam, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. Large language models still can't plan (a benchmark for llms on planning and reasoning about change). *arXiv preprint arXiv:2206.10498*, 2022.

[6] Konstantinos Katserelis and Konstantinos Skianis. Towards fine-dining recipe generation with generative pre-trained transformers. *arXiv preprint arXiv:2209.12774*, 2022.

[7] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

[8] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in Neural Information Processing Systems*, 36, 2024.

[9] Jinzhe Liu, Xiangsheng Huang, Zhuo Chen, and Yin Fang. Drak: Unlocking molecular insights with domain-specific retrieval-augmented knowledge in llms. *Authorea Preprints*, 2024.

[10] Yashen Wang, Huanhuan Zhang, Zhirun Liu, and Qiang Zhou. Hierarchical concept-driven language model. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 15(6):1–22, 2021.

[11] Giovanni Ciatto, Andrea Agiollo, Matteo Magnini, and Andrea Omicini. Large language models as oracles for instantiating ontologies with domain-specific knowledge. *arXiv preprint arXiv:2404.04108*, 2024.

[12] Constructions Aeronautiques, Adele Howe, Craig Knoblock, ISI Drew McDermott, Ashwin Ram, Manuela Veloso, Daniel Weld, David Wilkins SRI, Anthony Barrett, Dave Christianson, et al. Pddl| the planning domain definition language. *Technical Report, Tech. Rep.*, 1998.

[13] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *EMNLP*. ACL, 11 2019.

[14] Hankz Hankui Zhuo, Tuan Anh Nguyen, and Subbarao Kambhampati. Model-lite case-based planning. In *AAAI*, pages 1077–1083. AAAI Press, 2013.

[15] Hankz Hankui Zhuo and Subbarao Kambhampati. Model-lite planning: Case-based vs. model-based approaches. *Artif. Intell.*, 246:1–21, 2017.

[16] Kebing Jin, Hankz Hankui Zhuo, Zhanhao Xiao, Hai Wan, and Subbarao Kambhampati. Gradient-based mixed planning with symbolic and numeric action parameters. *Artif. Intell.*, 313:103789, 2022.

[17] David Heckerman. A tutorial on learning with bayesian networks. *Innovations in Bayesian networks: Theory and applications*, pages 33–82, 2008.

[18] Jiageng Mao, Yuxi Qian, Hang Zhao, and Yue Wang. Gpt-driver: Learning to drive with gpt. *arXiv preprint arXiv:2310.01415*, 2023.

[19] Stephen P Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[20] Xavier Puig, Kevin Ra, Marko Boben, Jiaman Li, Tingwu Wang, Sanja Fidler, and Antonio Torralba. Virtualhome: Simulating household activities via programs. In *CVPR*, pages 8494–8502, 2018.

[21] Rishi Hazra, Pedro Zuidberg Dos Martires, and Luc De Raedt. Saycanpay: Heuristic planning with large language models using learnable domain knowledge. *arXiv preprint arXiv:2308.12682*, 2023.

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

[23] Rodrigo Toro Icarte, Toryn Klassen, Richard Valenzano, and Sheila McIlraith. Using reward machines for high-level task specification and decomposition in reinforcement learning. In *ICML*, pages 2107–2116. PMLR, 2018.

[24] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[25] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *ACL*, pages 311–318, 2002.

[26] Chin-Yew Lin. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81, 2004.

[27] Jie Huang and Kevin Chen-Chuan Chang. Towards reasoning in large language models: A survey. *arXiv preprint arXiv:2212.10403*, 2022.

[28] Liangming Pan, Alon Albalak, Xinyi Wang, and William Yang Wang. Logic-lm: Empowering large language models with symbolic solvers for faithful logical reasoning. *arXiv preprint arXiv:2305.12295*, 2023.

[29] Stephen Cresswell and Peter Gregory. Generalised domain model acquisition from action traces. In *Twenty-First International Conference on Automated Planning and Scheduling*, 2011.

[30] SRK Branavan, Nate Kushman, Tao Lei, and Regina Barzilay. Learning high-level planning from text. The Association for Computational Linguistics, 2012.

[31] Belinda Z Li, Maxwell Nye, and Jacob Andreas. Implicit representations of meaning in neural language models. *arXiv preprint arXiv:2106.00737*, 2021.

[32] Kebing Jin and Hankz Hankui Zhuo. Integrating AI planning with natural language processing: A combination of explicit and tacit knowledge. *CoRR*, abs/2202.07138, 2022.

[33] Kebing Jin, Huaixun Chen, and Hankz Hankui Zhuo. Text-based action-model acquisition for planning. *CoRR*, abs/2202.08373, 2022.

[34] Wenfeng Feng, Hankz Hankui Zhuo, and Subbarao Kambhampati. Extracting action sequences from texts based on deep reinforcement learning. In Jérôme Lang, editor, *IJCAI*, pages 4064–4070. ijcai.org, 2018.

[35] Harsha Kokel, Arjun Manoharan, Sriraam Natarajan, Balaraman Ravindran, and Prasad Tadepalli. Dynamic probabilistic logic models for effective abstractions in rl. *arXiv preprint arXiv:2110.08318*, 2021.

[36] Samy Badreddine, Artur d'Avila Garcez, Luciano Serafini, and Michael Spranger. Logic tensor networks. *Artificial Intelligence*, 303:103649, 2022.

[37] Tirtharaj Dash, Sharad Chitlangia, Aditya Ahuja, and Ashwin Srinivasan. A review of some techniques for inclusion of domain-knowledge into deep neural networks. *Scientific Reports*, 12(1):1040, 2022.

[38] Rik Koncel-Kedziorski, Dhanush Bekal, Yi Luan, Mirella Lapata, and Hannaneh Hajishirzi. Text generation from knowledge graphs with graph transformers. *arXiv preprint arXiv:1904.02342*, 2019.

[39] Linfeng Song, Ante Wang, Jinsong Su, Yue Zhang, Kun Xu, Yubin Ge, and Dong Yu. Structural information preserving for graph-to-text generation. *arXiv preprint arXiv:2102.06749*, 2021.

[40] Zhexin Zhang, Jiaxin Wen, Jian Guan, and Minlie Huang. Persona-guided planning for controlling the protagonist's persona in story generation. *arXiv preprint arXiv:2204.10703*, 2022.

[41] Hankz Hankui Zhuo, Qiang Yang, Rong Pan, and Lei Li. Cross-domain action-model acquisition for planning via web search. In Fahiem Bacchus, Carmel Domshlak, Stefan Edelkamp, and Malte Helmert, editors, *ICAPS*. AAAI, 2011.

[42] Hankz Hankui Zhuo. Crowdsourced action-model acquisition for planning. In Blai Bonet and Sven Koenig, editors, *AAAI*, pages 3439–3446. AAAI Press, 2015.

[43] Hankz Hankui Zhuo. Recognizing multi-agent plans when action models and team plans are both incomplete. *ACM Trans. Intell. Syst. Technol.*, 10(3):30:1–30:24, 2019.

[44] Hankz Hankui Zhuo, Yantian Zha, Subbarao Kambhampati, and Xin Tian. Discovering underlying plans based on shallow models. *ACM Trans. Intell. Syst. Technol.*, 11(2):18:1–18:30, 2020.