Reasoning Like Program Executors

Xinyu Pi, Qian Liu, Bei Chen, Morteza Ziyadi, Zeqi Lin
Qiang Fu, Yan Gao, Jian-Guang Lou, Weizhu Chen

University of Illinois Urbana-Champaign, Urbana, USA; Sea AI Lab, Singapore
Microsoft Research Asia, Beijing, China; Microsoft Azure AI, Redmond, WA, USA

xinyupi2@illinois.edu; liuqian@sea.com;
{beichen, morteza.ziyadi, zeqi.lin, qifu, yan.gao, jlou, wzchen}@microsoft.com

Abstract

Reasoning over natural language is a long-standing goal for the research community. However, studies have shown that existing language models are inadequate in reasoning. To address the issue, we present PoET, a novel reasoning pre-training paradigm. Through pre-training language models with programs and their execution results, PoET empowers language models to harvest the reasoning knowledge possessed by program executors via a data-driven approach. PoET is conceptually simple and can be instantiated by different kinds of program executors. In this paper, we showcase two simple instances PoET-Math and PoET-Logic, in addition to a complex instance, PoET-SQL. Experimental results on six benchmarks demonstrate that PoET can significantly boost model performance in natural language reasoning, such as numerical reasoning, logical reasoning, and multi-hop reasoning. PoET opens a new gate on reasoning enhancement pre-training, and we hope our analysis would shed light on the future research of reasoning like program executors.

1 Introduction

Recent breakthroughs in pre-training illustrate the power of pre-trained Language Models (LM) on a wide range of Natural Language (NL) tasks. Pre-training on self-supervised tasks, such as masked language modeling (Devlin et al., 2019; He et al., 2021) using large amounts of NL sentences, boosts the language understanding of models by a large margin (Wang et al., 2018a). However, existing pre-training paradigms have primarily focused on language modeling and paid little attention to advanced reasoning capabilities (Table 1). As a result, though reaching near-human performance on several tasks, pre-trained LMs are still far behind expectations in reasoning-required scenarios (Rae et al., 2021), such as numerical reasoning (Wallace et al., 2019; Ravichander et al., 2019) and logical reasoning (Yu et al., 2020; Liu et al., 2020).

To alleviate the deficiency, reconciling NL understanding in LMs and reasoning in symbolic representations, i.e., neuro-symbolic reasoning, has been a major area of interest (Besold et al., 2017; Zhang et al., 2021). With a hybrid architecture, i.e., symbolic reasoners attached to LMs, neural-symbolic reasoning shines in a variety of reasoning tasks (Chen et al., 2020c; Tu et al., 2020; Wolfson et al., 2020). However, the reasoning mechanism remains in the symbolic reasoner and is not internalized into LMs, making it difficult to reuse the reasoning mechanism on unseen tasks. Meanwhile, neural models are notorious for their reliance on correlations among concrete tokens of a representation system and are usually assumed to be hard to grasp abstract rules of a symbolic reasoner (Helwe et al., 2021; Sinha et al., 2021). This drives us to
<table>
<thead>
<tr>
<th>Type</th>
<th>Example</th>
<th>Dataset</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numerical</td>
<td><strong>Question:</strong> What is the difference in casualty numbers between Bavarian and Austrian? <strong>Passage:</strong> [DOC] The popular uprising included large areas of ...</td>
<td>DROP (Dua et al., 2019)</td>
<td>Reading Comprehension (RC)</td>
</tr>
<tr>
<td>Logical</td>
<td><strong>Conclusion:</strong> One employee supervises another who gets more salary than himself. <strong>Fact:</strong> [DOC] David, Jack and Mark are colleagues in a company. David supervises Jack, and Jack supervises Mark. David gets more ...</td>
<td>LogiQA (Liu et al., 2020)</td>
<td>Reading Comprehension (RC)</td>
</tr>
<tr>
<td>Multi-hop</td>
<td><strong>Question:</strong> At which university does the biographer of John Clare teach English Literature? <strong>Passage:</strong> [DOC] John Clare: John Clare was an English poet ...</td>
<td>HotpotQA (Yang et al., 2018)</td>
<td>Reading Comprehension (RC)</td>
</tr>
<tr>
<td>Hybrid</td>
<td><strong>Question:</strong> What was the percentage change in gaming between 2018 and 2019? <strong>Context:</strong> [TAB] Server products and cloud services (32, 622, 26, 129 ... [DOC] Our commercial cloud revenue, which includes Office ...</td>
<td>TAT-QA (Zhu et al., 2021)</td>
<td>Question Answering (QA)</td>
</tr>
<tr>
<td>Quantitative</td>
<td><strong>Hypothesis:</strong> Teva earns $7 billion a year. <strong>Premise:</strong> After the deal closes, Teva will generate sales of about $7 billion a year, the company said.</td>
<td>EQUATE (Ravichander et al., 2019)</td>
<td>Natural Language Inference (NLI)</td>
</tr>
</tbody>
</table>

Table 1: The demonstration of five representative reasoning types. Listed are the types, the example questions, the representative dataset, and their corresponding tasks. [DOC] and [TAB] indicates a start of a passage and a semi-structured table respectively. Here we regard **Question**, **Conclusion** and **Hypothesis** as sentence, and **Passage**, **Fact**, **Context** and **Premise** as natural context in Figure 1.

explore whether symbolic reasoning can be internalized by language models and, especially,

Can neural language models advance reasoning abilities by imitating symbolic reasoners?

Motivated by this, we conceive a new pre-training paradigm, POET (Program Executor), to investigate the learnability of language models from symbolic reasoning and transferability across distinct representation systems. As illustrated in Figure 1, with a program (e.g., SQL query) and its program context (e.g., database) as input, the model receives automatic supervision from an established program executor (e.g., MySQL) and learns to produce correct execution result. By imitating program execution procedures, we believe LMs could potentially learn the reasoning knowledge that humans adopted to create the associated program executor and tackle NL sentences with the learned reasoning capability. This reveals the key hypothesis of POET: *program executors are crystallized knowledge of formal reasoning, and such knowledge can be grasped by language models and transferred to NL reasoning via pre-training.* In other words, pre-training over natural language might be a contingent condition for LMs to have better reasoning capabilities over natural language.

This contingency assumption of NL brings POET another great merit in data quality: while it is typically difficult to obtain large amounts of clean natural language sentences containing clear evidence of reasoning, synthesized programs can be made arbitrarily complicated but readily available on any scale, thanks to the artificial and compositional nature of programming languages. These merits greatly facilitate the construction of high-quality corpora, addressing most of the unresolved shortcomings in previous reasoning-enhancement pre-training. In other words, POET differs from existing pre-training paradigms relying on noisy NL data. In summary, our contribution is three-fold:

- We propose POET, a new pre-training paradigm for boosting the reasoning capabilities of language models by imitating program executors. Along with this paradigm, we present three exemplary across-program POET instantiations for various reasoning capabilities.

- We show with quantitative experiments that the reasoning ability our models obtains from POET pre-training is transferable to broader natural language scenarios. On six reasoning-focused downstream tasks, POET enables general-purpose language models to achieve competitive performance.

- We carry out comprehensive analytical studies, summarize insightful open questions, and provide insights for future work. We hope these insights would shed light on more research on reasoning like program executors.

2 Related Work

Since we focus on reasoning over natural language, our work is closely related to previous work on

---

1The code is available at [https://github.com/microsoft/ContextualSP](https://github.com/microsoft/ContextualSP)
reasoning skills in NL tasks. Regarding methods to inject reasoning skills into LMs, our method is related to two lines of work contributing to the topic: specialized models and pre-training. Last, our work is also related to program execution since we employ program executors during pre-training.

Reasoning Skills The literature focuses on reasoning skills, including numerical reasoning (Dua et al., 2019; Li et al., 2022a), multi-hop reasoning (Yang et al., 2018), reasoning in hybrid context (Chen et al., 2020b; Zhu et al., 2021) and logical reasoning (Liu et al., 2020; Yu et al., 2020). Our work concentrates on improving the above reasoning skills, leaving the other reasoning abilities such as commonsense reasoning (Zellers et al., 2018; Talmor et al., 2019; Bhagavatula et al., 2020) for future work.

Reasoning via Specialized Models Early works typically design specialized models and augment them into LMs for different types of questions (Dua et al., 2019; Andor et al., 2019; Hu et al., 2019; Ding et al., 2019). Taking Hu et al. (2019) as an example, they first predicted the answer type of a given question (e.g., “how many”), and then adopted the corresponding module (e.g., count module) to predict the answer. Although these methods work well on a specific dataset, it is challenging for them to scale to complex reasoning scenarios (Chen et al., 2020c). Differently, our work follows the line of reasoning via pre-training, which enjoys better scalability.

Reasoning via Pre-training This line of work focuses on the continued pre-training of LMs using large-scale data which involves reasoning. The pre-training data are generally NL text, which are either crawled from Web with distant supervision (Deng et al., 2021), generated by a model-based generator (Asai and Hajishirzi, 2020), or synthesized via human-designed templates (Geva et al., 2020; Yoran et al., 2022; Campagna et al., 2020; Wang et al., 2022). However, large-scale high-quality textual data involving reasoning is difficult to collect (Deng et al., 2021). Meanwhile, as the complexity of desired reasoning operations increases, synthesizing high-quality (e.g., fluent) NL sentences becomes more challenging. Different from the above pre-training methods relying on NL data, our pre-training is performed on programs. These programs can be synthesized at any scale with high quality, and thus are much easier to collect.

Reasoning in Giant Language Models Recent works demonstrate that with proper prompting (e.g., chain-of-thoughts prompting), giant language models (e.g., GPT-3) can perform well on reasoning tasks (Wei et al., 2022; Kojima et al., 2022; Li et al., 2022b). For example, Wei et al. (2022) find that giant language models have the ability to perform complex reasoning step by step with few-shot examples. Although these prompting strategies do not need further fine-tuning, the basic assumptions of them and PoET are similar, i.e., it is difficult to obtain large amounts of clean sentences involving complex reasoning. However, these prompting strategies do not work well for non-giant language models, while PoET is simultaneously applicable to language models ranging from millions (e.g., BART) to billions (e.g., T5-11B). It is also interesting to investigate how these prompting strategies and PoET can be combined.

Program Execution We present a framework to leverage program executors to train LMs, and thus our work is close to recent work on learning a neural program executor. In this line, the most related
work to ours is Liu et al. (2022), which revealed the possibility of SQL execution on helping table pre-training. Different from them mainly focusing on table-related tasks, we present a generalized approach to include Math, Logic, and SQL, as well as their applications on many different natural language downstream tasks. Other related studies include learning program executors on visual question answering (Andreas et al., 2016), reading comprehension (Gupta et al., 2019; Khot et al., 2021), knowledge base question answering (Ren et al., 2021) and 3D rendering (Tian et al., 2019).

These works mainly focus on learning a neural network to represent the program executor, while ours focuses on transferring the knowledge of program executor to downstream tasks via pre-training. Other lines of research leverage program execution in inference as a reliable sanity guarantee for generated programs by pruning non-executable candidates (Wang et al., 2018b; Chen et al., 2019b, 2021; Odena et al., 2020; Ellis et al., 2019; Chen et al., 2019b; Sun et al., 2018; Zohar and Wolf, 2018).

3 Reasoning Like Program Executors

Reasoning is the process where deduction and induction are sensibly applied to draw conclusions from premises or facts (Scriven, 1976). As a supreme feature of intelligence, humans apply reasoning across modalities. Taking numerical reasoning as an example, humans can tell how many chocolates are consumed from a math word problem description, or from a real-world event where a mother gets off work and finds the choco-can empty, aside standing their guilty-looking kids with brownish stains on their faces. Through detachment of information from their superficial modality and symbolic abstraction, humans manage to unify input formats and condense their numerical reasoning knowledge into one executable symbolic system – This is the origin of an arithmetic program executor. If a model can master these reasoning skills by imitating program executors, we believe in the possibility of transferring those reasoning skills to different modalities. In our case, we expect language models to transfer reasoning to NL-related tasks. Given this motivation, we discuss the fundamental components of PoET in the rest of this section and present its instantiations later.

Program refers to a finite sequence of symbols that can be understood and executed by machines. For example, a program can be a logical form (e.g., Prolog), a piece of code (e.g., Python), or a math expression. Compared with NL sentences, programs are more formal. Each well-established program follows a specific set of grammar rules and can thus be synthesized systematically. The generalizability of PoET framework is free from assumption and derived from the set of grammar rules on which a program follows. In PoET, as long as a program returns meaningful output to reflect its computational procedure, it is an acceptable program.

Program Context is the environment in which a program is running, which holds numerous variables accessible to the program. These variables serve as pivot points that anchor the program context with the program. In the same sense, the question and the passage in reading comprehension hold a similar relationship. This suggests a natural analogy between the program-to-program context and the sentence-to-natural context in Figure 1.

Program Executor is a black-box software that can execute a given program within the program context. An example could be the Python interpreter that executes each line of code, with its specific input data structures as program context. For PoET, program executors play the role of teachers to educate students (i.e., LMs) on reasoning knowledge they contain. PoET expects program executors to deterministically execute an input program with respect to a specific program context.

Execution Result is obtained from the program executor, given a program and program context as input. It is much analogous to the answer part in NL downstream tasks. The execution result is the primary observable data reflecting the intermediate
reasoning process and serves as the supervision provided by the program executor.

4 POET with Singleton Executors

We first instantiate POET with two singleton (i.e., a single type of reasoning capability) executors and then move on to POET with integrated executors.

4.1 Learning from Math Calculators

The POET-Math (Left in Figure 3) aims at injecting numerical reasoning skills into LMs via learning from math calculators. Specifically, POET-Math is designed to boost the basic arithmetic skills (i.e., addition and subtraction) of LMs on downstream tasks. This arithmetic skill aligns with requirements to answer questions centered on addition/subtraction between two numbers, such as “What is the difference in casualty numbers between Bavarian and Austrian?”.

Pre-training Task Given several floating-point variables as the program context and a math expression only involving addition/subtraction as the program, the pre-training task of POET-Math is to calculate the math expression. Taking the leftmost example from Figure 3, receiving the concatenation of the program and the program context as the input, POET-Math is trained to output the number 180.7. Considering the output can be an arbitrary number, the encoder-decoder model (Lewis et al., 2020) is more suitable for this pre-training task.

Pre-training Corpus Each example in the corpus contains several premise statements and a conclusion statement. Initially, the statement collection for each example is empty. To produce it, we first allocate 5 Boolean variables (e.g., p and q in Figure 3) and randomly sample at most 8 pairs from their pairwise combinations. For each sampled pair (p, q), we randomly select a statement from the set \( \{ p \rightarrow q, p \rightarrow \neg q, \neg p \rightarrow q, \neg p \rightarrow \neg q \} \) and add it to the collection. Once the statement collection is prepared, we randomly select a statement as the conclusion statement (i.e., program) and the rest as the premise statements (i.e., program context). Last, we employ Z3 (De Moura and Björner, 2008), the well-known satisfiability modulo theory solver, as our program executor to obtain the implied result. Finally, we synthesize 1 million examples as the pre-training corpus for POET-Math.

4.2 Learning from Logic Solvers

The POET-Logic (Mid in Figure 3) aims at injecting logical reasoning (e.g., necessary conditional reasoning) skills into LMs via learning from logic solvers. For example, taking the facts “Only if the government reinforces basic education can we improve our nation’s education to a new stage. In order to stand out among other nations, we need to have a strong educational enterprise.” as premises, POET-Logic is intended to help LMs identify whether the conclusion “In order to stand out among nations, we should reinforce basic education” is necessarily implied.

Pre-training Task Given a few first-order logic premise statements as the program context and one conclusion statement as the program, the pre-training task of POET-Logic is to identify if the program is necessarily implied from the program context. The execution result, i.e., the implication relationship between the program and the program context, is either True or False. Since the output is binary, an encoder-only model (Liu et al., 2019) is sufficient to perform this pre-training task.

Pre-training Corpus Each example in the corpus contains several premise statements and a conclusion statement. Initially, the statement collection for each example is empty. To produce it, we first allocate 5 Boolean variables (e.g., p and q in Figure 3) and randomly sample at most 8 pairs from their pairwise combinations. For each sampled pair (p, q), we randomly select a statement from the set \( \{ p \rightarrow q, p \rightarrow \neg q, \neg p \rightarrow q, \neg p \rightarrow \neg q \} \) and add it to the collection. Once the statement collection is prepared, we randomly select a statement as the conclusion statement (i.e., program) and the rest as the premise statements (i.e., program context). Last, we employ Z3 (De Moura and Björner, 2008), the well-known satisfiability modulo theory solver, as our program executor to obtain the implied result. Finally, we synthesize 1 million examples as the pre-training corpus for POET-Logic, and nearly 16% examples correspond to True.

4.3 Preliminary Observation

We perform experiments on DROP and LogiQA to verify if our method improves the reasoning capability required by the dataset. As observed in Figure 4, POET-Math boosts the numerical rea-

---

3More discussion can be found in Appendix § A.
Table 2: The six typical SQL programs that require reasoning. Listed are the type and the example SQL programs. [COL] and [VAL] represent the table column and the table cell value, respectively.

<table>
<thead>
<tr>
<th>Type</th>
<th>Example SQL Program</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arithmetic</td>
<td>SELECT [COL]1 - [COL]2</td>
</tr>
<tr>
<td>Superlative</td>
<td>SELECT MAX([COL]1)</td>
</tr>
<tr>
<td>Comparative</td>
<td>SELECT [COL]1, WHERE [COL]2 &gt; [VAL]3</td>
</tr>
<tr>
<td>Aggregation</td>
<td>SELECT COUNT([COL]1)</td>
</tr>
<tr>
<td>Union</td>
<td>SELECT [COL]1, WHERE [COL]2 = [VAL]3 OR [COL]3 = [VAL]3</td>
</tr>
<tr>
<td>Nested</td>
<td>SELECT [COL]1, WHERE [COL]2 IN ( SELECT [COL]3 WHERE [COL]3 = [VAL]3 )</td>
</tr>
</tbody>
</table>

Seasoning ability of BART, bringing in 9.0% EM gain on DROP. Meanwhile, PoET-Logic improves the logical reasoning skills of RoBERTa, resulting in a 2.2% EM improvement on LogiQA. These significant improvements support our main claim that reasoning knowledge of program executors can be transferred to NL scenarios via pre-training.

5 PoET with Integrated Executors

PoET-Math and PoET-Logic each focus on one specific reasoning skill, making the pre-training task highly dependent on the downstream task. Different from them, PoET-SQL is proposed to allow LMs to master different reasoning skills simultaneously. In our implementation, PoET-SQL is pre-trained with an integrated SQL executor, since we believe that SQL queries are complex enough to encompass a wide variety of computational procedures (Table 2).

Pre-training Task Given a SQL query as the program and a database as the program context, the pre-training task of PoET-SQL is to mimic the query result generation. As shown on the right side of Figure 5, given the concatenation of the program and the program context, the model is pre-trained to output the query result. Since the encoder-decoder LMs can generate arbitrary tokens, they are well suited for the task. On the other hand, encoder-only LMs have insufficient expressiveness to produce out-of-context query results. To allow them to benefit from the SQL execution, we tailor the task into a query result selection task for encoder-only LMs, which only utilizes query results that can be found in the database. Specifically, the task requires encoder-only LMs to perform an 10 sequence tagging process to find the query results in the database, as shown on the left side of Figure 5.

Figure 5: The illustration of PoET-SQL pre-training tasks: query result selection for encoder-only and query result generation for encoder-decoder LMs.

Note that the tag 1 is for tokens in the query results (e.g., Athens), while 0 is for other tokens.

Pre-training Corpus Each example in the corpus contains a SQL query, a database, and a query result. Notably, following Liu et al. (2022), each database is flattened into a sequence when it is fed into LMs. Meanwhile, to avoid databases being too large to fit into memory, we randomly drop the rows of large databases until their flattened sequences contain less than 450 tokens. For the query result generation task, we follow the same corpus construction strategy as described in Liu et al. (2022). Concretely, by instantiating SQL templates from SQuALL (Shi et al., 2020) over databases provided by WIKIQL (Zhong et al., 2017), 5 million examples are synthesized for pre-training. For the query result selection task, the pre-training corpus is constructed in a similar way as above, except that only the examples whose query results are suitable for encoder-only are retained. This filtering results in a corpus containing nearly 2 million examples.

6 Experiments and Analysis

To verify the effectiveness of PoET-SQL on boosting the reasoning capabilities of LMs, we first apply our method on several backbone models, including encoder-only models and encoder-decoder models. Then we conduct experiments on five typical reasoning benchmark datasets and compare PoET-SQL with previous methods. Last, we perform a detailed model analysis to provide more insights.
Comparing to Previous Methods Table 4 lists all experimental results of baselines and our models on different datasets. As seen, our model generally achieves highly competitive results on different reasoning skills, showing its strong performance. Compared with other reasoning-enhanced LMs, PoET-SQL\footnote{Note: this footnote is not included in the text.} surpasses them by a large
Figure 6: The EM performance [%] on the pre-training dev set (Left) and the downstream DROP dev set (Right) with different pre-training steps and scales. PoET-SQL (x%) denotes the model trained with x% pre-training examples, while 100% corresponds to the model trained with the whole pre-training corpus of PoET-SQL, which contains 5 million examples.

margin, demonstrating the effectiveness of our proposed program execution pre-training. For example, compared with PReasM initialized from TS-Large, PoET-SQL \textsc{Bart} initialized from BART-Large exceeds it by 8.3%. Furthermore, PoET that learns from a mix of program executors (i.e., PoET-Math+SQL-\textsc{Bart}) achieves a slightly better performance than the single program executor.

6.2 Pre-training Analysis

We show part of the analysis results below due to the limited space, and more analysis can be found in Appendix § A and § D.

Necessity of Program Execution PoET hypothesizes that the acquisition of reasoning ability by models happens at the stage of mimicking program execution, rather than program language modeling. To verify it, we ablate the program executor in PoET-SQL-\textsc{Bart} and carry out a SQL language modeling pre-training instead. Practically, we mask each SQL query in the pre-training corpus of PoET-SQL using the strategy adopted in BART (Lewis et al., 2020), and pre-train BART to output the complete SQL query given the masked SQL query and the database. The trivial performance variance corroborates the necessity of program execution.

Effect of the Pre-training Step and Scale Figure 6 illustrates the pre-training and downstream performance with different pre-training steps and scales. It can be seen that both pre-training and downstream performance gradually increase towards the asymptote with increasing pre-training steps, while extra pre-training data steadily accelerate the convergence rate. Although a larger scale yields better performance on the pre-training dev set, 10% (500k) data can already converge approximately to the same asymptote as the full data pre-training, showing the high data efficiency of PoET. The highly plateaued curve also serves as sound evidence that execution pre-training is a data-efficient pre-training approach that converges quickly.

7 Discussion and Open Questions

In this section, we carry out comprehensive studies on PoET, summarize interesting open questions, and provide insights for future work.

💡 Does PoET improve reasoning skills at the sacrifice of NL understanding abilities? No.

During pre-training, our models are exposed to artificial programs that are dramatically different from NL sentences, raising the concern that models may catastrophically forget their original NL understanding ability. We explore this by comparing PoET-SQL-\textsc{RoBERTa} and the vanilla RoBERTa model on tasks focusing on NL understanding, including SQuAD, MNLI and QuoRef. As can be observed in Figure 7, PoET-SQL-\textsc{RoBERTa} performs almost equally well as RoBERTa on two datasets (i.e., SQuAD and QuoRef), which suggests that PoET barely sacrifices the intrinsic NL understanding ability of LMs. And the performance drop on MNLI (1.2%) is also noteworthy, which may be alleviated by joint pre-training on language modeling and our proposed program execution. More experimental details can be found in Appendix § E.

💡 Will PoET be affected by naturalness of program context or program? No.

An intuitive hypothesis is that the effectiveness of PoET should be positively associated with the naturalness of program and program context, due to
closer learning objectives. To test this hypothesis, we design two groups of experiments: (i) Tuning the naturalness of program - we follow Liu et al. (2022) to translate SQL queries into NL sentences to make a more natural program, and replace SQL reserved keywords with low-frequency tokens to make a more unnatural program. (ii) Tuning the naturalness of program context - we follow Chen et al. (2019a) to convert each database in PoET-SQL into a set of NL sentences. Surprisingly, results in Table 5 provide counter-evidence to the intuitive hypothesis, since tuning the naturalness of program or program context do not significantly impact PoET effectiveness. For example, unnatural program only leads to a slight decrease in DROP EM from 77.7% to 76.9%. It also indicates that the model learns certain abstract reasoning capabilities rather than lexical associations.

<table>
<thead>
<tr>
<th>Settings</th>
<th>EM</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>PoET-SQL$_{BART}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tuning Program</td>
<td></td>
<td></td>
</tr>
<tr>
<td>w. Natural program</td>
<td>77.2</td>
<td>79.9</td>
</tr>
<tr>
<td>w. Unnatural program</td>
<td>76.9</td>
<td>79.7</td>
</tr>
</tbody>
</table>

Table 5: The EM and F1 of PoET-SQL$_{BART}$ on the DROP dev set with respect to different naturalness of program and program context.

Figure 8: The train and dev perplexity of vanilla BART and BART pre-trained on DROP (BART+DROP) on the pre-training corpus of PoET-SQL.

If reasoning ability can be transferred from program execution to NL reasoning tasks in PoET, then the reversed process of PoET may also work, i.e., models pre-trained with NL reasoning would have better learnability on program execution. To test this speculation, we compare the behavioral difference of vanilla BART and BART pre-trained on DROP in terms of learning SQL execution in Figure 8. There are two indicators that can be used to characterize the behavior of LMs on the SQL execution task: execution accuracy and perplexity, and the execution accuracy always goes higher when the perplexity goes lower. Here the perplexity is presented because it is smoother compared to the execution accuracy, which is either 100% or 0%. Parallel with our expectation, pre-training on DROP leads to observably lower perplexity for SQL execution learning on both the train and dev sets. The bidirectional enhancement suggests some relative independence between reasoning mechanisms and their symbolic representations.

💡 Can PoET boost reasoning abilities of giant pre-trained language models? Yes.

Recent work suggest that giant LMs excel at reasoning (Brown et al., 2020), so we are curious if PoET is effective for them. Following the same procedure as in § 6, we apply PoET-SQL to T5-11B, one of the largest publicly available LMs. As shown in Table 6, albeit not as shining as in cases of smaller LMs, PoET still succeeds in boosting numerical reasoning abilities of giant LMs.

8 Conclusion & Future Work

We introduce PoET, a new pre-training paradigm for boosting reasoning capability of language models via imitating program executors. Experimental results on six datasets demonstrate that PoET can significantly boost existing language models on several reasoning skills, including numerical, logical and multi-hop reasoning. Our best language model under PoET can reach highly competitive performance with previous specialized models. In the future, we hope our work could inspire more transference of reasoning knowledge from program executors to models. And we will also investigate the causes of the reasoning transfer with more insightful experiments, since we still do not know how the reasoning transfer occurs.
Limitations

The first limitation of our approach is that it has a relatively strong coupling between the reasoning skills learned in the pre-training task and the reasoning skills required for the downstream task. In other words, POET expects reasoning abilities of the program executor to overlap with the downstream reasoning requirements to make the execution learning transferable. Such an expectation also applied to POET-SQL, although it allows LM to master different reasoning skills at the same time. For example, when ablating all programs involving math operations from the pre-training corpus of POET-SQL, it shows poor performance on DROP. The second limitation is that POET still employs instantiated program templates rather than probabilistic context-free grammars to synthesize programs. The latter usually offers a more diverse range of programs that may contribute to the generalization of the pre-trained language models, but are often more complex.

Acknowledgement

We would like to thank Frank F. Xu, Zhengbao Jiang, Bill Yuchen Lin, Shuyan Zhou, Zhiruo Wang, Libo Qin, Jize Jiang, and Jonathan Livengood for fruitful discussion. We also thank all the anonymous reviewers for their constructive feedback and insightful comments.

References


Tarek R. Besold, Artur S. d’Avila Garcez, Sebastian Bader, Howard Bowman, Pedro M. Domingos, Pas-


Yonglong Tian, Andrew Luo, Xingyuan Sun, Kevin Ellis, William T. Freeman, Joshua B. Tenenbaum, and Jiajun Wu. 2019. Learning to infer and execute 3d shape programs.


Shuohang Wang, Mo Yu, Jing Jiang, and Shiyu Chang. 2018c. A co-matching model for multi-choice read-


### A Program Context Analysis

POET emphasizes the importance of program context for reasoning transferability, owing to the analogy between the program to program context and the sentence to natural context drawn in Figure 1. To investigate it, we explore the effect of different program context design choices on reasoning transferability by conducting experiments on well-designed POET-Math variants.

#### A.1 The Necessaritity of Program Context

To verify the necessity of program context, we experiment with POET-Math without program context, i.e., a variable-free POET-Math variant whose program context is empty. Taking the example of POET-Math in Figure 3, the program is transformed into $152.0 + 99.0 - 70.3$. The experimental results are shown in Table 7. One can see that there
A.2 The Variables Design in Program Context

In the pre-training task of PoET-Math, the program context is several floating-point variables. These variables include necessary variables (i.e., variables required by the program) and irrelevant variables. The irrelevant variables exist to make the program context closer to the natural context which generally contains irrelevant sentences. For example, given the program $a + b$ and the program context $a = 1; b = 2; c = 3; d = 4;$, variables $c$ and $d$ are what we refer to as irrelevant variables. This is motivated by the fact that passages are usually full of irrelevant information regarding a specific question in NL downstream tasks.

In this section, we explore impacts on pre-training effectiveness brought by numbers of irrelevant variables. Empirically, we experiment on pre-training with 0, 10, 30 irrelevant variables. The total length of 30 irrelevant variables approaches the maximum input length of pre-trained LMs, and thus we do not try more settings. The experimental results are shown in Table 7. As observed, (i) models can still learn numerical reasoning during pre-training where the program context is free from irrelevant variables, though less effective. (ii) the setting of 30 irrelevant variables brings BART-Large more performance improvement than the setting of 10 irrelevant variables. Considering there are plenty of lengthy passages in the DROP dataset, we therefore hypothesize that the noise level brought by irrelevant variables in the program context during pre-training should be made closer with the counterpart in the natural context during fine-tuning.

Table 7: The DROP performance with different numbers of irrelevant variables in PoET-Math pre-training.

is a dramatic performance drop in the variant compared to PoET-Math, verifying the importance of program context.

Table 8 presents some statistics about our experimental datasets. Below we introduce each dataset in detail.

**DROP** A reading comprehension benchmark to measure numerical reasoning ability over a given passage (Dua et al., 2019). It contains three subsets of questions: span, number, and date, each of which involves a lot of numerical operations. Unlike traditional reading comprehension datasets such as SQuAD (Rajpurkar et al., 2016) where answers are always a single span from context, several answers in the span subset of DROP contains multiple spans. The number and date answers are mostly out of context and need generative-level expressiveness.

**HotpotQA** An extractive reading comprehension dataset that requires models to perform multi-hop reasoning over different passages (Yang et al., 2018). It contains two settings (i) Distractor: reasoning over 2 gold paragraphs along with 8 similar distractor paragraphs and (ii) Full wiki: reasoning over customized retrieval results from full Wikipedia passages. We experiment with its distractor setting since retrieval strategy is beyond our focus in this work.

**TAT-QA** A question answering benchmark to measure reasoning ability over hybrid context, i.e., passages and tables (Zhu et al., 2021). It is curated by combing paragraphs and tables from real-world financial reports. According to the source(s) the answers are derived from, the dataset can be divided into three subsets: Table, Text and Table-Text(both).

**EQUATE** The first benchmark dataset to explore quantitative reasoning under the task of natural language inference (Ravichander et al., 2019). As a test-only dataset, it requires fine-tuned models on
MNLI to perform zero-shot natural language inference tasks over quantitative statements described in (premise, hypothesis) pairs to reach final entailment decisions.

LogiQA A multi-choice reading comprehension dataset that evaluates the logical reasoning ability, whose questions are designed by domain experts (Liu et al., 2020). It contains four types of logical reasoning, including categorical reasoning, disjunctive reasoning, conjunctive reasoning, and conditional reasoning.

SVAMP A challenging math word problem dataset (Patel et al., 2021). It is designed specifically to hack models who leverage spurious patterns to perform arithmetic operations without true understanding of context. We only keep addition and subtraction problems in accordance with our pre-training coverage.

B.2 Baseline Setup

We summarize the baseline methods in short below, and refer readers to their papers for more details. (i) On DROP, we include two families of models for comparison: specialized models such as NumNet(+) (Ran et al., 2019), MTMSN (Hu et al., 2019), NeRd (Chen et al., 2020c), QDGAT (Chen et al., 2020a) and language models such as GenBERT (Geva et al., 2020) and PReaM (Yoran et al., 2022). (ii) Similarly, on HotpotQA (Distractor), specialized model baselines include DFGN (Qiu et al., 2019), SAE (Tu et al., 2020), C2F Reader (Shao et al., 2020) and the SOTA model HGN (Fang et al., 2020). The language model baselines consist of BERT (Devlin et al., 2019), SpanBERT (Joshi et al., 2020) and ReasonBERT (Deng et al., 2021). (iii) On TAT-QA, we adopt the official baselines, including TAPAS (Hertzig et al., 2020), NumNet+ V2 and the SOTA model TagOP (Zhu et al., 2021). (iv) On EQUATE, we compare our methods with BERT (Devlin et al., 2019), GPT (Radford et al., 2019) and Q-REAS (Ravichander et al., 2019). (v) On LogiQA, we compare our methods with Co-Matching Network (Wang et al., 2018c) and the SOTA model DAGN (Huang et al., 2021).

C Implementation Details

C.1 PoET-SQL\textsubscript{RoBERTa} on Different Datasets

On DROP, we cast the span selection task as a sequence tagging problem following Segal et al. (2020). On TAT-QA, we in-place substitute the RoBERTa-Large encoder in TAGOP (Zhu et al., 2021) with our PoET-SQL\textsubscript{RoBERTa} to verify its effectiveness, and keep the rest of the components unchanged. On HotpotQA, we train two classifiers independently to predict the start and end positions of the answer span, as done in Devlin et al. (2019). On EQUATE, we train a classifier to perform sequence classification on concatenated premise-hypothesis pairs. Notably, we follow the official setup to train LMs on the MNLI dataset (Williams et al., 2018) and evaluate their zero-shot performance on EQUATE. On SVAMP, the encoder-only model is not suitable since the answers are out-of-context.

C.2 Pre-training Details

By default, we apply AdamW as pre-training optimizer with default scheduling parameters in fairseq. The coefficient of weight decay is set as 0.05 to alleviate over-fitting of pre-trained models. Additionally, we employ fp16 to accelerate the pre-training.

PoET-Math The pre-training procedure lasts for 10,000 steps with a batch size of 512. After the warm up in the first 2000 steps, the learning rate arrives the peak at $3 \times 10^{-5}$ during pre-training.

PoET-Logic The pre-training procedure lasts for 5,000 steps with a batch size of 512. After the warm up in the first 1000 steps, the learning rate arrives the peak at $3 \times 10^{-5}$ during pre-training.

PoET-SQL For PoET-SQL\textsubscript{BART} and PoET-SQL\textsubscript{RoBERTa}, the pre-training procedure lasts for 50,000 steps with a batch size of 512. After the warm up in the first 5000 steps, the learning rate arrives the peak at $3 \times 10^{-5}$ during pre-training. To save memory, each example in the pre-training corpus could at most contains 512 tokens. For PoET-SQL\textsubscript{TS}, the pre-training procedure lasts for 20,000 steps with a batch size of 512. After the warm up in the first 2000 steps, the learning rate arrives the peak at $1 \times 10^{-5}$ during pre-training. The maximum input length in each example is truncated to 384 tokens to increase the batch size.

C.3 Fintuning Details

We implement our models based on transformers (Wolf et al., 2020), fairseq (Ott et al., 2019) and DeepSpeed.\footnote{http://github.com/microsoft/DeepSpeed}
Table 9: Breakdown of model F1 score by answer types on the dev set of DROP. Some works only report overall span type performance (marked by *), and single-span is non-separable from multi-span performance. Bold and underlined numbers indicate the best and second-best results, respectively.

<table>
<thead>
<tr>
<th>Models</th>
<th>RTE-Q</th>
<th>NewsNLI</th>
<th>RedditNLI</th>
<th>NR ST</th>
<th>AWPNI</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Previous Systems</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAJ</td>
<td>57.8</td>
<td>50.7</td>
<td>58.4</td>
<td>33.3</td>
<td>50.0</td>
<td>50.4</td>
</tr>
<tr>
<td>BERT</td>
<td>57.2</td>
<td>72.8</td>
<td>49.6</td>
<td>36.9</td>
<td>42.2</td>
<td>51.8</td>
</tr>
<tr>
<td>GPT</td>
<td>68.1</td>
<td>72.2</td>
<td>52.4</td>
<td>36.4</td>
<td>50.0</td>
<td>55.8</td>
</tr>
<tr>
<td>Q-REAS</td>
<td>56.6</td>
<td>61.1</td>
<td>50.8</td>
<td>63.3</td>
<td>71.5</td>
<td>60.7</td>
</tr>
<tr>
<td><strong>Original LMs</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BART-Large</td>
<td>68.1</td>
<td>76.2</td>
<td>65.0</td>
<td>53.7</td>
<td>49.7</td>
<td>62.6</td>
</tr>
<tr>
<td>RoBERTa-Large</td>
<td>69.3</td>
<td>75.5</td>
<td>65.6</td>
<td>60.1</td>
<td>50.7</td>
<td>64.2</td>
</tr>
<tr>
<td><strong>POET Models</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>POET-SQL\textsuperscript{BART}</td>
<td>72.3</td>
<td>75.2</td>
<td>64.8</td>
<td>70.7</td>
<td>49.5</td>
<td>66.5</td>
</tr>
<tr>
<td>POET-SQL\textsuperscript{RoBERTa}</td>
<td>75.3</td>
<td>75.5</td>
<td>68.1</td>
<td>69.2</td>
<td>50.5</td>
<td>67.5</td>
</tr>
</tbody>
</table>

Table 10: The EM performance of different models on all subsets of the EQUATE benchmark. Bold and underlined numbers indicate the best and second-best results, respectively.

**Passage Retrieval in HotpotQA** Since the total length of the original passages in HotpotQA is too long to fit into memory, we train a classifier to filter out top-3 passages, as done in previous work (Deng et al., 2021). Specifically, a RoBERTa-Large model is fine-tuned to discriminate if an input passage is required to answer the question. The Hits@3 score of the classifier on HotpotQA is 97.2%.

**Numerical Design in DROP and SVAMP** As noticed by previous works, sub-word tokenization methods such as byte pair encoding (Sennrich et al., 2015) potentially undermines the arithmetic ability of models. Instead, the character-level number representation is argued to be a more effective alleviation (Wallace et al., 2019). Additionally, the reverse decoding of numbers is proposed as a better way of modelling arithmetic carry (Geva et al., 2020). Therefore, we employ these design strategies on DROP and SVAMP.

**C.4 Fine-tuning Hyperparameters**

By default, we apply AdamW as fine-tuning optimizer with default scheduling parameters on all datasets. To ensure statistical significance, all fine-tuning procedures are run with three random seeds, except for T5-11B and POET-SQL\textsuperscript{TS} due to the limit of computation budgets.

**DROP** POET-SQL\textsuperscript{RoBERTa} and RoBERTa-Large are trained with the subset of questions marked as "span" from the DROP dataset. Since a gold answer may occur multiple times in the passage, we optimize over the sum of negative log probability for all possibly-correct IO sequences where each one of gold answers is included at least once,
Table 11: The EM performance of TAGOp (PoET-SQL\textsubscript{RoBERTa}) with respect to answer types and sources on the dev set of TAT-QA.

<table>
<thead>
<tr>
<th></th>
<th>Table</th>
<th>Text</th>
<th>Table-Text</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>EM / F</td>
<td>EM / F</td>
<td>EM / F</td>
<td>EM / F</td>
</tr>
<tr>
<td>Arithmetic</td>
<td>50.1 / 50.1</td>
<td>43.8 / 50.0</td>
<td>55.6 / 55.6</td>
<td>51.5 / 51.5</td>
</tr>
<tr>
<td>Counting</td>
<td>66.7 / 66.7</td>
<td>- / -</td>
<td>90.0 / 90.0</td>
<td>81.3 / 81.3</td>
</tr>
<tr>
<td>Spans</td>
<td>67.4 / 80.6</td>
<td>54.2 / 80.8</td>
<td>79.2 / 84.8</td>
<td>71.4 / 82.6</td>
</tr>
<tr>
<td>Span</td>
<td>68.4 / 68.4</td>
<td>51.2 / 76.0</td>
<td>76.2 / 77.8</td>
<td>61.9 / 74.6</td>
</tr>
<tr>
<td>Total</td>
<td>56.5 / 58.0</td>
<td>51.1 / 75.0</td>
<td>69.0 / 70.7</td>
<td>59.1 / 65.9</td>
</tr>
</tbody>
</table>

as done in Segal et al. (2020). The fine-tuning procedure runs up to 25,000 steps with a batch size of 64, with the learning rate of 7.5 × 10⁻⁶. As for BART-Large (and PoET-SQL\textsubscript{BART}, PoET-Math, the same below) and T5-11B (and PoET-SQL\textsubscript{T5}, the same below), they are trained with the whole DROP dataset. For BART-Large, the fine-tuning procedure runs up to 20,000 steps with a batch size as 128 and a learning rate as 3 × 10⁻⁵. For T5-11B, due to the computational budget, the fine-tuning procedure only lasts for 10,000 steps with a batch size of 32, and the learning rate is 1 × 10⁻⁵.

**TAT-QA** In the experiment of TAT-QA, we employ the official implementation and the default hyperparameters provided in TAGOp ⁵. The fine-tuning procedure runs up to 50 epochs with a batch size of 48. For modules introduced in TAGOP, the learning rate is set as 5 × 10⁻⁴, while for RoBERTa-Large (and PoET-SQL\textsubscript{RoBERTa}), the learning rate is set as 1.5 × 10⁻⁵.

**HotpotQA** The fine-tuning procedure runs up to 30,000 steps with a batch size of 64. The learning rate is 1 × 10⁻⁵. Overlong inputs are truncated to 512 tokens for both RoBERTa-Large (and PoET-SQL\textsubscript{RoBERTa}), T5-11B (and PoET-SQL\textsubscript{T5}) and BART-Large (and PoET-SQL\textsubscript{BART}).

**EQUATE** The fine-tuning procedure runs up to 20,000 steps on MNLI with a batch size of 128 for both RoBERTa-Large (and PoET-SQL\textsubscript{RoBERTa}), and BART-Large (and PoET-SQL\textsubscript{BART}), with learning rate is 1 × 10⁻⁵. After fine-tuning, models are directly evaluated on EQUATE.

**LogiQA** In the experiment of LogiQA, we employ the open-source implementation and the default hyperparameters provided in ReClor ⁶ (Yu et al., 2020) to fine-tune RoBERTa-Large (and PoET-SQL\textsubscript{RoBERTa}). The fine-tuning procedure runs up to 10 epochs with a batch size of 24. The learning rate is set as 1 × 10⁻⁵.

**D Fine-grained Analysis**

**DROP** In Table 9 we report model F₁ scores by question type on DROP. Comparing three PoET pre-trained models with their vanilla versions, we observe that: (i) PoET-SQL\textsubscript{BART} outperforms the vanilla BART-large with a wide margin in all types of questions, i.e. number (15.3%), date (9.8%), span (around 5%). (ii) PoET-SQL\textsubscript{RoBERTa} only deals with span selection questions, and obtain 1.9%, 3.2% gain on span, spans questions, respectively. (iii) For the giant PoET-SQL\textsubscript{T5}, we also observe 2% improvement on number questions, 2.2% on span and 0.8% on spans questions. These model-agnostic performance boost on DROP reveals the extra numerical reasoning knowledge models learned from SQL program executors.

**EQUATE** Table 10 presents performance breakdown by subsets of EQUATE (Ravichander et al., 2019), where we compare PoET-SQL\textsubscript{BART} and PoET-SQL\textsubscript{RoBERTa} with their vanilla versions and previous baselines. For both models, we observe around 10% acc improvement on the NR ST subset, where numerical comparison and quantifiers are especially emphasized. Stable performance improvement was also observed in both pre-trained models on the RTE-Q subset, where arithmetics and ranges are primary focus. Interestingly, PoET-SQL\textsubscript{RoBERTa} alone demonstrate improvement on RedditNLI (emphasizes approximation and verbal quantitative reasoning) subset. Performance on other subsets are approximately comparable between PoET pre-trained models and vanilla models, suggesting that PoET does not harm intrinsic abilities of language models.

**TAT-QA** Table 11 shows the detailed experimental results of TAGOp (PoET-SQL\textsubscript{RoBERTa}). Considering that the pre-training of PoET-SQL\textsubscript{RoBERTa} is

---

⁵https://github.com/NExTplusplus/TAT-QA ⁶https://github.com/yuweihao/reclor
only performed on table-like texts (i.e., the flatten sequence of databases), it is highly non-trivial for our model to generalize to such a hybrid scenario containing both tables and passages, again illustrating the transferability of reasoning capabilities.

E NL Understanding Performance

Dataset Details  We fine-tune PoET-SQL$_{4RoBERTa}$ on (i) SQuAD v1.0: (Rajpurkar et al., 2016): one of the most classical single-span selection RC benchmarks measuring understanding over natural language context; (ii) MNLI (Williams et al., 2018): a large-scale NLI dataset measuring cross-domain and cross-genre generalization of NLU. Notably, our model is evaluated on the matched setting for the purpose of simplicity. (iii) QuoRef (Dasigi et al., 2019): A Wikipedia-based multi-span selection RC benchmark with a special emphasis on coreference resolution. All dataset Statistics are shown in Table 12.

Implementation Details (i) On SQuAD, we cast the span selection task as a sequence tagging problem following Segal et al. (2020). (ii) On MNLI-matched, we train both models to perform sequence classification on concatenated premise-hypothesis pairs. (iii) On Quoref, we cast the span(s) selection task as an IO sequence tagging problem following Segal et al. (2020).

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Train</th>
<th>Dev</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td># Questions</td>
<td># Docs</td>
</tr>
<tr>
<td>SQuAD</td>
<td>77, 409</td>
<td>5, 565</td>
</tr>
<tr>
<td>MNLI</td>
<td>392, 702</td>
<td>392, 702</td>
</tr>
<tr>
<td>QuoRef</td>
<td>19, 399</td>
<td>3, 771</td>
</tr>
</tbody>
</table>

Table 12: PoET on NL understanding experiment dataset statistics.