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 **L**anguage **M**odels (LM) on a wide range of **N**atural **L**anguage (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

[*] The first two authors contributed equally. Work done during internship at Microsoft Research Asia.

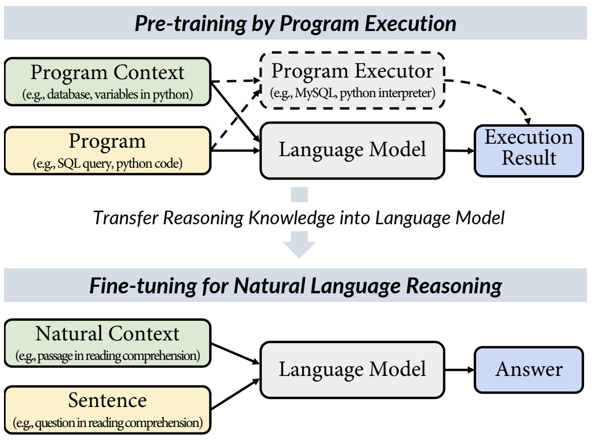
Pre-training by Program Execution
Program Context
(e.g., database, variables in python)
Program Executor
(e.g., MySQL, python interpreter)
Program
(e.g., SQL query, python code)
Language Model
Execution Result
Transfer Reasoning Knowledge into Language Model
Fine-tuning for Natural Language Reasoning
Natural Context
(e.g., passage in reading comprehension)
Sentence
(e.g., question in reading comprehension)
Language Model
Answer

Figure 1: Given a program context and a program as input, PoET pre-trains LMs to output the execution result. After fine-tuning on downstream tasks, PoET can boost LMs on reasoning-required scenarios. Explanations about program context, program, program executor and execution result can be found in § 3. More examples of natural context and sentence are in Table 1.

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

Type	Example	Dataset	Task
Numerical	**Question:** What is the difference in casualty numbers between Bavarian and Austrian? **Passage:** `[DOC]` The popular uprising included large areas of . . .	DROP (Dua et al., 2019)	Reading Comprehension (RC)
Logical	**Conclusion:** One employee supervises another who gets more salary than himself. **Fact:** `[DOC]` David, Jack and Mark are colleagues in a company. David supervises Jack, and Jack supervises Mark. David gets more . . .	LogiQA (Liu et al., 2020)	Reading Comprehension (RC)
Multi-hop	**Question:** At which university does the biographer of John Clare teach English Literature? **Passage:** `[DOC]` John Clare : John Clare was an English poet . . . `[DOC]` CMS College Kottayam : The CMS College is one . . .	HotpotQA (Yang et al., 2018)	Reading Comprehension (RC)
Hybrid	**Question:** What was the percentage change in gaming between 2018 and 2019? **Context:** `[TAB]` Server products and cloud services | 32, 622 | 26, 129 . . . `[DOC]` Our commercial cloud revenue, which includes Office . . .	TAT-QA (Zhu et al., 2021)	Question Answering (QA)
Quantitative	**Hypothesis:** Teva earns $7 billion a year. **Premise:** After the deal closes, Teva will generate sales of about $7 billion a year, the company said.	EQUATE (Ravichander et al., 2019)	Natural Language Inference (NLI)

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 the 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 (**Pro**gram **E**xecu**t**or), to investigate the learnability of language models from symbolic reasoning and transferrability 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[1].

2 Related Work

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

[1]The code is available at https://github.com/microsoft/ContextualSP

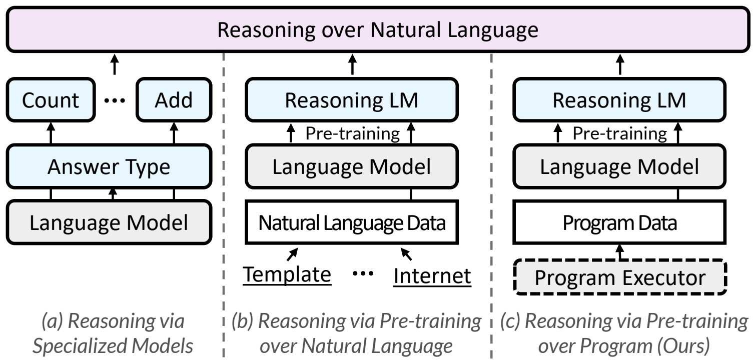

Figure 2: The illustration of different lines of reasoning, including **(a)** reasoning via specalized models, **(b)** reasoning via pre-training over natural language and **(c)** reasoning via pre-training over program (Ours).

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.

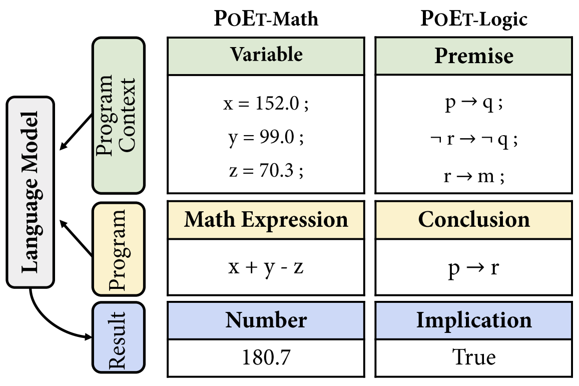

Figure 3: The illustration of PoEt-Math and PoEt-Logic. During pre-training, the concatenation of *program* and *program context* are fed into *language model* and the model is expected to output *result*.

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 a math expression containing up to 2 operators and 3 variables, and a program context that contains at most 30 floating-point variables[2]. The mathematical addition and subtraction operators are denoted by `+` and `-`, respectively. The values of variables vary from 0.0 to 1000.0. By random generation, we synthesize 4 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

[2]More discussion can be found in Appendix § A.

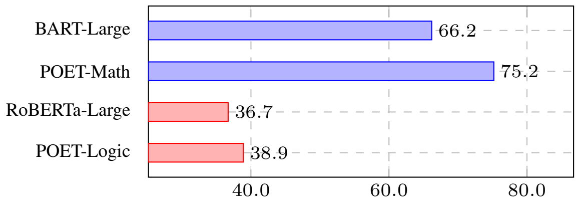

Figure 4: Fine-tuning EM performance [%] of different models on DROP (blue) and LogiQA (red).

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 $\{p \rightarrow q, p \rightarrow \neg q, \neg p \rightarrow \neg q, \neg p \rightarrow q\}$ 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 Bjørner, 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[3] 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-

[3]To clarify, 16% is not a specific-purpose design but a statistical result.

Type	Example SQL Program
Arithmetic	SELECT [COL]$_1$ - [COL]$_2$
Superlative	SELECT MAX([COL]$_1$)
Comparative	SELECT [COL]$_1$ WHERE [COL]$_2$ > [VAL]$_2$
Aggregation	SELECT COUNT([COL]$_1$)
Union	SELECT [COL]$_1$ WHERE [COL]$_2$ = [VAL]$_2$ OR [COL]$_3$ = [VAL]$_3$
Nested	SELECT [COL]$_1$ WHERE [COL]$_2$ IN (SELECT [COL]$_2$ WHERE [COL]$_3$ = [VAL]$_3$)

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.

soning 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 heavily 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 `IO` sequence tagging process to find the query results in the database, as shown on the left side of Figure 5.

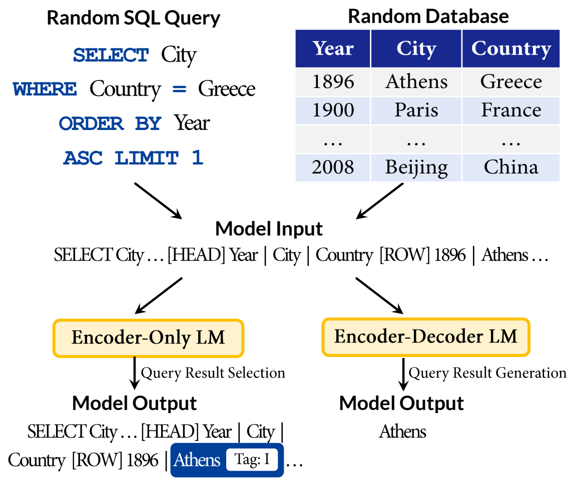

Figure 5: The illustration of PoEt-SQL pre-training tasks: query result selection for encoder-only and query result generation for encoder-decoder LMs.

Note that the tag `I` is for tokens in the query results (e.g., Athens), while `O` 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 WikiSQL (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.

Models	DROP$^{\heartsuit}$		HotpotQA$^{\heartsuit}$		TAT-QA$^{\heartsuit}$		SVAMP	EQUATE
	EM	F1	EM	F1	EM	F1	EM	EM
BART-Large	66.2	69.2	65.6	78.9	38.8	46.7	12.4	62.6
PoEt-SQL$_{\texttt{BART}}$	77.7 (+11.5)	80.6 (+11.4)	66.5 (+0.9)	79.7 (+0.8)	41.5 (+2.7)	49.6 (+2.9)	33.5 (+21.1)	66.5 (+3.9)
RoBERTa-Large	78.1	85.3	67.6	81.1	55.2	62.7	–	64.2
PoEt-SQL$_{\texttt{RoBERTa}}$	79.8 (+1.7)	87.4 (+2.1)	68.7 (+1.1)	81.6 (+0.5)	59.1 (+3.9)	65.9 (+3.2)	–	67.5 (+3.3)

Table 3: The main experimental results of different backbone models on test sets and dev sets (♡) of datasets with or without our PoEt-SQL. The results of PoEt are significantly better than the vanilla LMs ($p < 0.05$). Note the performance of RoBERTa and PoEt-SQL$_{\texttt{RoBERTa}}$ are reported on the subset of DROP where the answer is span(s).

Dataset	Models	EM	F_1
DROP$^{\heartsuit}$	*Specialized Models*		
	NumNet (Ran et al., 2019)	64.9	68.3
	MTMSN (Hu et al., 2019)	76.7	80.5
	NeRd (Chen et al., 2020c)	78.6	81.9
	NumNet+ (Ran et al., 2019)	81.1	84.4
	QDGAT (Chen et al., 2020a)	**84.1**	**87.1**
	Language Models		
	T5 (Yoran et al., 2022)	61.8	64.6
	GenBERT (Geva et al., 2020)	68.8	72.3
	PReasM (Yoran et al., 2022)	69.4	72.3
	PoEt-SQL$_{\texttt{BART}}$	77.7	80.6
	PoEt-Math+SQL$_{\texttt{BART}}$	**78.0**	**80.9**
TAT-QA$^{\heartsuit}$	TaPas (Herzig et al., 2020)	18.9	26.5
	NumNet+ (Ran et al., 2019)	38.1	48.3
	TagOp (Zhu et al., 2021)	55.2	62.7
	TagOp + PoEt-SQL$_{\texttt{RoBERTa}}$	**59.1**	**65.9**

Dataset	Models	EM	F_1
HotpotQA$^{\heartsuit}$	*Specialized Models*		
	DFGN (Qiu et al., 2019)	55.7	69.3
	SAE (Tu et al., 2020)	67.7	80.8
	C2F Reader (Shao et al., 2020)	68.0	81.2
	HGN (Fang et al., 2020)	**69.2**	**82.2**
	Language Models		
	BERT (Devlin et al., 2019)	59.1	73.4
	ReasonBERT (Deng et al., 2021)	64.8	79.2
	PoEt-SQL$_{\texttt{BART}}$	66.5	79.7
	SpanBERT (Joshi et al., 2020)	67.4	81.2
	PoEt-SQL$_{\texttt{RoBERTa}}$	**68.7**	**81.6**
EQUATE	BERT (Devlin et al., 2019)	51.8	–
	GPT (Radford et al., 2019)	55.8	–
	QREAS (Ravichander et al., 2019)	60.7	–
	PoEt-SQL$_{\texttt{BART}}$	66.5	–
	PoEt-SQL$_{\texttt{RoBERTa}}$	**67.5**	–

Table 4: The comparison of our models with baselines on test sets and dev sets (♡) of different datasets. LMs used by all baselines are in Large size, except for ReasonBERT. Bold numbers indicate the best results.

Dataset Setup We perform experiments on different datasets including DROP, HotpotQA, TAT-QA, and EQUATE. Table 1 shows examples of these datasets and their corresponding reasoning types. Furthermore, SVAMP (Patel et al., 2021), the challenging diagnostic dataset for probing *numerical reasoning*, is employed in our experiments to test the generalization capability of our fine-tuned models on DROP. We evalute models on their addition and subtraction subsets. More details about datasets can be found in Appendix § B.

Backbone Model RoBERTa (Liu et al., 2019) is elected as the backbone in encoder-only LMs, while BART (Lewis et al., 2020) is chosen as the backbone in encoder-decoder LMs. Afterward, We mark the RoBERTa model and the BART model trained under PoEt as PoEt-SQL$_{\texttt{RoBERTa}}$ and PoEt-SQL$_{\texttt{BART}}$, respectively. For PoEt-SQL$_{\texttt{BART}}$, we treat all datasets as generative tasks and fine-tune models to generate answers. As for PoEt-SQL$_{\texttt{RoBERTa}}$, the fine-tuning strategies on different datasets are slightly different, and more implementation details can be found in Appendix § C. Notably, on all datasets, our models are evaluated with official evaluation metrics EM and F1.

6.1 Experimental Results

Comparing to Vanilla LMs Table 3 presents an apple-to-apple performance comparison between PoEt-SQL models and their associated vanilla LMs. Across all instances, we observe significant performance increment on downstream NL reasoning tasks. Specifically, PoEt-SQL equips popular encoder-only and encoder-decoder models with an integrated package of reasoning skills, effectively improving their performance on five benchmark datasets. As a highlighted example, PoEt-SQL$_{\texttt{BART}}$ obtains 11.5% (DROP) and 21.1% (SVAMP) improvements on EM, compared with the vanilla BART. Since PoEt pre-training is carried purely on program context, whereas all downstream tasks are on natural context, our hypothesis that reasoning capability is transferable from program executors to NL scenarios gets verified.

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$_{\texttt{BART}}$ surpasses them by a large

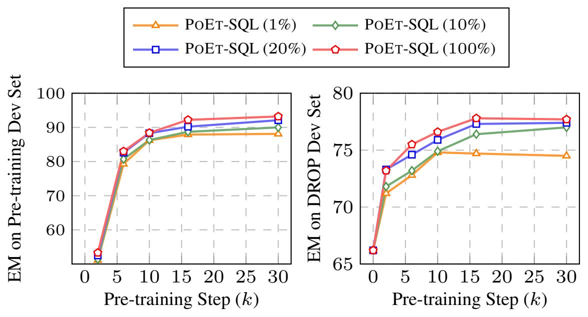

Figure 6: The EM performance [%] on the pre-training dev set (**Left**) and the downstream DROP dev set (**Right**) with different pre-training steps and scales. POET-SQL (x%) denotes the model trained with x% pre-training examples, while 100% corresponds to the model trained with the whole pre-training 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 T5-Large, POET-SQL$_{\text{BART}}$ initialized from BART-Large exceeds it by 8.3%. Furthermore, POET that learns from a mix of program executors (i.e., POET-Math+SQL$_{\text{BART}}$) achieves a slightly better performance than the single prgoram 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$_{\text{BART}}$ 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

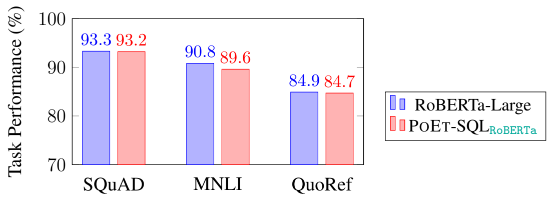

Figure 7: The performance comparison between RoBERTa-Large and POET-SQL$_{\text{RoBERTa}}$ on representative NLU tasks. On SQuAD and QuoRef, we report F$_1$, whereas on MNLI we report accuracy.

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$_{\text{RoBERTa}}$ 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$_{\text{RoBERTa}}$ 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

Settings	EM	F_1
PoEt-SQL$_{\text{BART}}$	77.7	80.6
Tuning Program		
↪ *w.* Nnatural program	77.2	79.9
↪ *w.* Unnatural program	76.9	79.7
Tuning Program Context		
↪ *w.* Natural program context	76.5	79.0

Table 5: The EM and F_1 of PoEt-SQL$_{\text{BART}}$ on the DROP dev set with respect to different naturalness of program and program context.

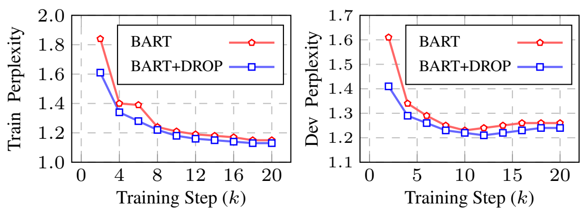

Figure 8: The train and dev perplexity of vanilla BART and BART pre-trained on DROP (BART+DROP) on the pre-training corpus of PoEt-SQL.

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 pre-training 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 indictaes that the model learns certain abstract reasoning capabilities rather than lexical associations.

Does pre-training on NL reasoning benefit model learning on program execution? **Yes**.

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

Models	DROP$^{\heartsuit}$		SVAMP
	EM	F1	EM
T5-11B	83.5	85.9	52.9
PoEt-SQL$_{\text{T5}}$	85.2 (+1.7)	87.6 (+1.7)	57.4 (+4.5)

Table 6: The experimental results of T5-11B and PoEt-SQL$_{\text{T5}}$ on test sets and dev sets ($\heartsuit$) of different datasets.

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 fo 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

Daniel Andor, Luheng He, Kenton Lee, and Emily Pitler. 2019. Giving BERT a calculator: Finding operations and arguments with reading comprehension. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5947–5952, Hong Kong, China. Association for Computational Linguistics.

Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. 2016. Neural module networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 39–48.

Akari Asai and Hannaneh Hajishirzi. 2020. Logic-guided data augmentation and regularization for consistent question answering. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 5642–5650, Online. Association for Computational Linguistics.

Tarek R. Besold, Artur S. d'Avila Garcez, Sebastian Bader, Howard Bowman, Pedro M. Domingos, Pascal Hitzler, Kai-Uwe Kühnberger, Luís C. Lamb, Daniel Lowd, Priscila Machado Vieira Lima, Leo de Penning, Gadi Pinkas, Hoifung Poon, and Gerson Zaverucha. 2017. Neural-symbolic learning and reasoning: A survey and interpretation. *CoRR*, abs/1711.03902.

Chandra Bhagavatula, Ronan Le Bras, Chaitanya Malaviya, Keisuke Sakaguchi, Ari Holtzman, Hannah Rashkin, Doug Downey, Wen tau Yih, and Yejin Choi. 2020. Abductive commonsense reasoning. In *International Conference on Learning Representations*.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

Giovanni Campagna, Agata Foryciarz, Mehrad Moradshahi, and Monica Lam. 2020. Zero-shot transfer learning with synthesized data for multi-domain dialogue state tracking. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 122–132, Online. Association for Computational Linguistics.

Kunlong Chen, Weidi Xu, Xingyi Cheng, Zou Xiaochuan, Yuyu Zhang, Le Song, Taifeng Wang, Yuan Qi, and Wei Chu. 2020a. Question directed graph attention network for numerical reasoning over text. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6759–6768, Online. Association for Computational Linguistics.

Shuang Chen, Qian Liu, Zhiwei Yu, Chin-Yew Lin, Jian-Guang Lou, and Feng Jiang. 2021. ReTraCk: A flexible and efficient framework for knowledge base question answering. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing: System Demonstrations*, pages 325–336, Online. Association for Computational Linguistics.

Wenhu Chen, Hongmin Wang, Jianshu Chen, Yunkai Zhang, Hong Wang, Shiyang Li, Xiyou Zhou, and William Yang Wang. 2019a. Tabfact: A large-scale dataset for table-based fact verification. In *International Conference on Learning Representations*.

Wenhu Chen, Hanwen Zha, Zhiyu Chen, Wenhan Xiong, Hong Wang, and William Yang Wang. 2020b. HybridQA: A dataset of multi-hop question answering over tabular and textual data. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1026–1036, Online. Association for Computational Linguistics.

Xinyun Chen, Chen Liang, Adams Wei Yu, Denny Zhou, Dawn Song, and Quoc V. Le. 2020c. Neural symbolic reader: Scalable integration of distributed and symbolic representations for reading comprehension. In *International Conference on Learning Representations*.

Xinyun Chen, Chang Liu, and Dawn Song. 2019b. Execution-guided neural program synthesis. In *International Conference on Learning Representations*.

Pradeep Dasigi, Nelson F. Liu, Ana Marasović, Noah A. Smith, and Matt Gardner. 2019. Quoref: A reading comprehension dataset with questions requiring coreferential reasoning. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5925–5932, Hong Kong, China. Association for Computational Linguistics.

Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: An efficient smt solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer.

Xiang Deng, Yu Su, Alyssa Lees, You Wu, Cong Yu, and Huan Sun. 2021. ReasonBERT: Pre-trained to reason with distant supervision. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 6112–6127, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Ming Ding, Chang Zhou, Qibin Chen, Hongxia Yang, and Jie Tang. 2019. Cognitive graph for multi-hop reading comprehension at scale. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2694–2703, Florence, Italy. Association for Computational Linguistics.

Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. 2019. DROP: A reading comprehension benchmark requiring discrete reasoning over paragraphs. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2368–2378, Minneapolis, Minnesota. Association for Computational Linguistics.

Kevin Ellis, Maxwell I. Nye, Yewen Pu, Felix Sosa, Josh Tenenbaum, and Armando Solar-Lezama. 2019. Write, execute, assess: Program synthesis with a REPL. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 9165–9174.

Yuwei Fang, Siqi Sun, Zhe Gan, Rohit Pillai, Shuohang Wang, and Jingjing Liu. 2020. Hierarchical graph network for multi-hop question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 8823–8838, Online. Association for Computational Linguistics.

Mor Geva, Ankit Gupta, and Jonathan Berant. 2020. Injecting numerical reasoning skills into language models. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 946–958, Online. Association for Computational Linguistics.

Nitish Gupta, Kevin Lin, Dan Roth, Sameer Singh, and Matt Gardner. 2019. Neural module networks for reasoning over text. *CoRR*, abs/1912.04971.

Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. 2021. Deberta: Decoding-enhanced bert with disentangled attention. In *International Conference on Learning Representations*.

Chadi Helwe, Chloé Clavel, and Fabian M. Suchanek. 2021. Reasoning with transformer-based models: Deep learning, but shallow reasoning. In *3rd Conference on Automated Knowledge Base Construction*.

Jonathan Herzig, Pawel Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Eisenschlos. 2020. TaPas: Weakly supervised table parsing via pre-training. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4320–4333, Online. Association for Computational Linguistics.

Minghao Hu, Yuxing Peng, Zhen Huang, and Dongsheng Li. 2019. A multi-type multi-span network for reading comprehension that requires discrete reasoning. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1596–1606, Hong Kong, China. Association for Computational Linguistics.

Yinya Huang, Meng Fang, Yu Cao, Liwei Wang, and Xiaodan Liang. 2021. DAGN: Discourse-aware graph network for logical reasoning. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5848–5855, Online. Association for Computational Linguistics.

Mandar Joshi, Danqi Chen, Yinhan Liu, Daniel S. Weld, Luke Zettlemoyer, and Omer Levy. 2020. SpanBERT: Improving pre-training by representing and predicting spans. *Transactions of the Association for Computational Linguistics*, 8:64–77.

Tushar Khot, Daniel Khashabi, Kyle Richardson, Peter Clark, and Ashish Sabharwal. 2021. Text modular networks: Learning to decompose tasks in the language of existing models. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1264–1279, Online. Association for Computational Linguistics.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *arXiv preprint arXiv:2205.11916*.

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

Moxin Li, Fuli Feng, Hanwang Zhang, Xiangnan He, Fengbin Zhu, and Tat-Seng Chua. 2022a. Learning to imagine: Integrating counterfactual thinking in neural discrete reasoning. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 57–69, Dublin, Ireland. Association for Computational Linguistics.

Yifei Li, Zeqi Lin, Shizhuo Zhang, Qiang Fu, Bei Chen, Jian-Guang Lou, and Weizhu Chen. 2022b. On the advance of making language models better reasoners. *arXiv preprint arXiv:2206.02336*.

Jian Liu, Leyang Cui, Hanmeng Liu, Dandan Huang, Yile Wang, and Yue Zhang. 2020. Logiqa: A challenge dataset for machine reading comprehension with logical reasoning. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 3622–3628. International Joint Conferences on Artificial Intelligence Organization. Main track.

Qian Liu, Bei Chen, Jiaqi Guo, Morteza Ziyadi, Zeqi Lin, Weizhu Chen, and Jian-Guang Lou. 2022. TAPEX: Table pre-training via learning a neural SQL executor. In *International Conference on Learning Representations*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692.

Augustus Odena, Kensen Shi, David Bieber, Rishabh Singh, Charles Sutton, and Hanjun Dai. 2020. Bustle: Bottom-up program synthesis through learning-guided exploration.

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

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 2021. Are NLP models really able to solve simple math word problems? In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2080–2094, Online. Association for Computational Linguistics.

Lin Qiu, Yunxuan Xiao, Yanru Qu, Hao Zhou, Lei Li, Weinan Zhang, and Yong Yu. 2019. Dynamically fused graph network for multi-hop reasoning. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 6140–6150, Florence, Italy. Association for Computational Linguistics.

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

Jack W. Rae, Sebastian Borgeaud, Trevor Cai, Katie Millican, Jordan Hoffmann, H. Francis Song, John Aslanides, Sarah Henderson, Roman Ring, Susannah Young, Eliza Rutherford, Tom Hennigan, Jacob Menick, Albin Cassirer, Richard Powell, George van den Driessche, Lisa Anne Hendricks, Maribeth Rauh, Po-Sen Huang, Amelia Glaese, Johannes Welbl, Sumanth Dathathri, Saffron Huang, Jonathan Uesato, John Mellor, Irina Higgins, Antonia Creswell, Nat McAleese, Amy Wu, Erich Elsen, Siddhant M. Jayakumar, Elena Buchatskaya, David Budden, Esme Sutherland, Karen Simonyan, Michela Paganini, Laurent Sifre, Lena Martens, Xiang Lorraine Li, Adhiguna Kuncoro, Aida Nematzadeh, Elena Gribovskaya, Domenic Donato, Angeliki Lazaridou, Arthur Mensch, Jean-Baptiste Lespiau, Maria Tsimpoukelli, Nikolai Grigorev, Doug Fritz, Thibault Sottiaux, Mantas Pajarskas, Toby Pohlen, Zhitao Gong, Daniel Toyama, Cyprien de Masson d'Autume, Yujia Li, Tayfun Terzi, Vladimir Mikulik, Igor Babuschkin, Aidan Clark, Diego de Las Casas, Aurelia Guy, Chris Jones, James Bradbury, Matthew Johnson, Blake A. Hechtman, Laura Weidinger, Iason Gabriel, William S. Isaac, Edward Lockhart, Simon Osindero, Laura Rimell, Chris Dyer, Oriol Vinyals, Kareem Ayoub, Jeff Stanway, Lorrayne Bennett, Demis Hassabis, Koray Kavukcuoglu, and Geoffrey Irving. 2021. Scaling language models: Methods, analysis & insights from training gopher. *CoRR*, abs/2112.11446.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas. Association for Computational Linguistics.

Qiu Ran, Yankai Lin, Peng Li, Jie Zhou, and Zhiyuan Liu. 2019. NumNet: Machine reading comprehension with numerical reasoning. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2474–2484, Hong Kong, China. Association for Computational Linguistics.

Abhilasha Ravichander, Aakanksha Naik, Carolyn Rose, and Eduard Hovy. 2019. EQUATE: A benchmark evaluation framework for quantitative reasoning in natural language inference. In *Proceedings of the 23rd Conference on Computational Natural Language Learning (CoNLL)*, pages 349–361, Hong Kong, China. Association for Computational Linguistics.

Hongyu Ren, Hanjun Dai, Bo Dai, Xinyun Chen, Michihiro Yasunaga, Haitian Sun, Dale Schuurmans, Jure Leskovec, and Denny Zhou. 2021. Lego: Latent execution-guided reasoning for multi-hop question answering on knowledge graphs. In *ICML*.

Michael Scriven. 1976. *Reasoning*. New York: McGraw-Hill.

Elad Segal, Avia Efrat, Mor Shoham, Amir Globerson, and Jonathan Berant. 2020. A simple and effective model for answering multi-span questions. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 3074–3080, Online. Association for Computational Linguistics.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2015. Neural machine translation of rare words with subword units. *CoRR*, abs/1508.07909.

Nan Shao, Yiming Cui, Ting Liu, Shijin Wang, and Guoping Hu. 2020. Is Graph Structure Necessary for Multi-hop Question Answering? In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7187–7192, Online. Association for Computational Linguistics.

Tianze Shi, Chen Zhao, Jordan Boyd-Graber, Hal Daumé III, and Lillian Lee. 2020. On the potential of lexico-logical alignments for semantic parsing to SQL queries. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1849–1864, Online. Association for Computational Linguistics.

Koustuv Sinha, Prasanna Parthasarathi, Joelle Pineau, and Adina Williams. 2021. UnNatural Language Inference. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 7329–7346, Online. Association for Computational Linguistics.

Shao-Hua Sun, Hyeonwoo Noh, Sriram Somasundaram, and Joseph Lim. 2018. Neural program synthesis from diverse demonstration videos. In *International Conference on Machine Learning*, pages 4790–4799. PMLR.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. 2019. CommonsenseQA: A question answering challenge targeting commonsense knowledge. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4149–4158, Minneapolis, Minnesota. Association for Computational Linguistics.

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.

Ming Tu, Kevin Huang, Guangtao Wang, Jing Huang, Xiaodong He, and Bowen Zhou. 2020. Select, answer and explain: Interpretable multi-hop reading comprehension over multiple documents. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 9073–9080. AAAI Press.

Eric Wallace, Yizhong Wang, Sujian Li, Sameer Singh, and Matt Gardner. 2019. Do NLP models know numbers? probing numeracy in embeddings. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5307–5315, Hong Kong, China. Association for Computational Linguistics.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2018a. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355, Brussels, Belgium. Association for Computational Linguistics.

Chenglong Wang, Kedar Tatwawadi, Marc Brockschmidt, Po-Sen Huang, Yi Xin Mao, Oleksandr Polozov, and Rishabh Singh. 2018b. Robust text-to-sql generation with execution-guided decoding. *ArXiv*, abs/1807.03100.

Shuohang Wang, Mo Yu, Jing Jiang, and Shiyu Chang. 2018c. A co-matching model for multi-choice reading comprehension. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 746–751, Melbourne, Australia. Association for Computational Linguistics.

Siyuan Wang, Wanjun Zhong, Duyu Tang, Zhongyu Wei, Zhihao Fan, Daxin Jiang, Ming Zhou, and Nan Duan. 2022. Logic-driven context extension and data augmentation for logical reasoning of text. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 1619–1629, Dublin, Ireland. Association for Computational Linguistics.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Chi, Quoc Le, and Denny Zhou. 2022. Chain of thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*.

Adina Williams, Nikita Nangia, and Samuel Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1112–1122, New Orleans, Louisiana. Association for Computational Linguistics.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Tomer Wolfson, Mor Geva, Ankit Gupta, Matt Gardner, Yoav Goldberg, Daniel Deutch, and Jonathan Berant. 2020. Break it down: A question understanding benchmark. *Transactions of the Association for Computational Linguistics*, 8:183–198.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2369–2380, Brussels, Belgium. Association for Computational Linguistics.

Ori Yoran, Alon Talmor, and Jonathan Berant. 2022. Turning tables: Generating examples from semi-structured tables for endowing language models with reasoning skills. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6016–6031, Dublin, Ireland. Association for Computational Linguistics.

Weihao Yu, Zihang Jiang, Yanfei Dong, and Jiashi Feng. 2020. Reclor: A reading comprehension dataset requiring logical reasoning. In *International Conference on Learning Representations (ICLR)*.

Rowan Zellers, Yonatan Bisk, Roy Schwartz, and Yejin Choi. 2018. SWAG: A large-scale adversarial dataset for grounded commonsense inference. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 93–104, Brussels, Belgium. Association for Computational Linguistics.

Jing Zhang, Bo Chen, Lingxi Zhang, Xirui Ke, and Haipeng Ding. 2021. Neural, symbolic and neural-symbolic reasoning on knowledge graphs. *AI Open*, 2:14–35.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2SQL: Generating structured queries from natural language using reinforcement learning. *arXiv*, abs/1709.00103.

Fengbin Zhu, Wenqiang Lei, Youcheng Huang, Chao Wang, Shuo Zhang, Jiancheng Lv, Fuli Feng, and Tat-Seng Chua. 2021. TAT-QA: A question answering benchmark on a hybrid of tabular and textual content in finance. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 3277–3287, Online. Association for Computational Linguistics.

Amit Zohar and Lior Wolf. 2018. Automatic program synthesis of long programs with a learned garbage collector. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 2098–2107.

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 Necessairty of Program Context

To verify the necessairty 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

Models	EM	F1
BART-Large	66.2	69.2
POET-Math *without* program context	67.4	70.5
POET-Math *with* 0 irrelevant variable	71.5	74.5
POET-Math *with* 10 irrelevant variables	74.6	77.5
POET-Math *with* 30 irrelevant variables	75.2	78.1

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.

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.

Dataset	Train		Dev	
	# Questions	# Docs	# Questions	# Docs
DROP	$77,409$	$5,565$	$9,536$	582
HotpotQA	$90,564$	$90,564$	$7,405$	$7,405$
TAT-QA	$13,215$	$2,201$	$1,668$	278
SVAMP	–	–	726	726
EQUATE	–	–	$9,606$	$9,606$
LogiQA	$6,942$	$6,942$	868	868

Table 8: The statistics of our experimental datasets.

B Experimental Setup

B.1 Dataset Setup

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 (Herzig 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$_{\text{RoBERTa}}$ 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$_{\text{RoBERTa}}$ 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 3×10^{-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 3×10^{-5} during pre-training.

POET-SQL For POET-SQL$_{\text{BART}}$ and POET-SQL$_{\text{RoBERTa}}$, 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 3×10^{-5} during pre-training. To save memory, each example in the pre-training corpus could at most contains 512 tokens. For POET-SQL$_{\text{T5}}$, 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 1×10^{-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 [4].

[4]http://github.com/microsoft/DeepSpeed

Models	**Number**	**Span**	**Spans**	**Date**	**Total**
		Previous Systems			
MTMSN (BERT)	81.1	82.8	62.8	69.0	80.5
NumNet+ (RoBERTa)	83.1	86.8*	86.8*	63.9	84.4
QDGAT (RoBERTa)	86.2	88.5*	88.5*	67.5	87.1
GenBERT	75.2	74.5	24.2	56.4	72.3
PReasM	64.4	86.6	78.4	77.7	72.3
		Original LMs			
RoBERTa-Large	–	86.4	79.9	–	–
BART-Large	63.6	79.6	74.6	62.1	69.2
T5-11B	83.2	90.2	85.8	84.9	85.8
		POET *Models*			
POET-SQL$_{\text{RoBERTa}}$	–	88.2	83.1	–	–
POET-SQL$_{\text{BART}}$	78.9	84.5	79.6	71.9	80.6
POET-SQL$_{\text{T5}}$	85.2	92.4	86.6	84.4	87.6

Table 9: Breakdown of model F_1 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.

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

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 Hyperpameters

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$_{\text{T5}}$ due to the limit of computation budgets.

DROP POET-SQL$_{\text{RoBERTa}}$ and RoBERTa-Large are trained with the subset of questions marked as "span" from the DROP dataset.t 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	Text	Table-Text	Total
	EM / F_1	EM / F_1	EM / F_1	EM / F_1
Arithmetic	50.1 / 50.1	43.8 / 50.0	55.6 / 55.6	51.5 / 51.5
Counting	66.7 / 66.7	– / –	90.0 / 90.0	81.3 / 81.3
Spans	67.4 / 80.6	54.2 / 80.8	79.2 / 84.8	71.4 / 82.6
Span	68.4 / 68.4	51.2 / 76.0	76.2 / 77.8	61.9 / 74.6
Total	56.5 / 58.0	51.1 / 75.0	69.0 / 70.7	59.1 / 65.9

Table 11: The EM performance of TAGOP (POET-SQL$_{\text{RoBERTa}}$) with respect to answer types and sources on the dev set of TAT-QA.

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$_{\text{BART}}$, POET-Math, the same below) and T5-11B (and POET-SQL$_{\text{T5}}$, 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$_{\text{RoBERTa}}$), 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$_{\text{RoBERTa}}$), T5-11B (and POET-SQL$_{\text{T5}}$) and BART-Large (and POET-SQL$_{\text{BART}}$).

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$_{\text{RoBERTa}}$) and BART-Large (and POET-SQL$_{\text{BART}}$), 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$_{\text{RoBERTa}}$). The fine-tuning procedure runs up to 10 epochs with a batch size of 24. The learning rate is set as 1×10^{-5}.

[5]https://github.com/NExTplusplus/TAT-QA

[6]https://github.com/yuweihao/reclor

D Fine-grained Analysis

DROP In Table 9 we report model F_1 scores by question type on DROP. Comparing three POET pre-trained models with their vanilla versions, we observe that: (i) POET-SQL$_{\text{BART}}$ 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$_{\text{RoBERTa}}$ only deals with span selection questions, and obtain 1.9%, 3.2% gain on *span, spans* questions, respectively. (iii) For the giant POET-SQL$_{\text{T5}}$, 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$_{\text{BART}}$ and POET-SQL$_{\text{RoBERTa}}$ 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$_{\text{RoBERTa}}$ 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$_{\text{RoBERTa}}$). Considering that the pre-training of POET-SQL$_{\text{RoBERTa}}$ is

Dataset	**Train**		**Dev**	
	# Questions	# Docs	# Questions	# Docs
SQuAD	77,409	5,565	9,536	582
MNLI	392,702	392,702	9,815	9,815
QuoRef	19,399	3,771	2,418	454

Table 12: PoEt on NL understanding experiment dataset statistics.

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$_{\texttt{RoBERTa}}$ 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).