What If: Generating Code to Answer Simulation Questions in Chemistry Texts

Gal Peretz  
Technion - Israel Institute of Technology  
Haifa, Israel  
galprz@cs.technion.ac.il

Mousa Arraf  
Technion - Israel Institute of Technology  
Haifa, Israel  
amosa@cs.technion.ac.il

Kira Radinsky  
Technion - Israel Institute of Technology  
Haifa, Israel  
kirar@cs.technion.ac.il

ABSTRACT
Many texts, especially in chemistry and biology, describe complex processes. We focus on texts that describe a chemical reaction process and questions that ask about the process’s outcome under different environmental conditions. To answer questions about such processes, one needs to understand the interactions between the different entities involved in the process and simulate their state transitions during the process execution under other conditions. We hypothesize that generating code and executing it to simulate the process will allow answering such questions. We, therefore, define a domain-specific language (DSL) to represent processes. We contribute to the community a unique dataset curated by chemists and annotated by computer scientists. The dataset is composed of process texts, simulation questions, and their corresponding computer codes represented by the DSL. We propose a neural program synthesis approach based on reinforcement learning with a novel state-transition semantic reward. The novel reward is based on the run-time semantic similarity between the predicted code and the reference code. This allows simulating complex process transitions and thus answering simulation questions. Our approach yields a significant boost in accuracy for simulation questions: we achieved 88% accuracy as opposed to 83% accuracy of the state-of-the-art neural program synthesis approaches and 54% accuracy of state-of-the-art end-to-end text-based approaches.

CCS CONCEPTS
- Information systems → Question answering  
- Applied computing → Chemistry

KEYWORDS
Simulation Questions, Code Generation, Chemistry, Neural Code Generation, Question Answering

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

SIGIR ’23, July 23–27, 2023, Taipei, Taiwan. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3539618.3591783

1 INTRODUCTION
Many texts, especially in chemistry and biology, describe complex processes. These describe a chemical reaction process and require answering questions asking about the process’s outcome under different environmental conditions. To answer these questions, one needs to understand the interactions between the different entities involved in the process and simulate their state transitions during the process execution under other conditions. Consider the following example:

"A mixture of 50 g (0.42 mole) of p-aminobenzonitrile, 146 ml of concentrated hydrochloric acid, and 106 ml of H2O was heated at 80° for 20 minutes and then cooled to 0°. ... The hexane was cooled and the resulting precipitate was filtered and air dried to yield 16 g (18%) of p-(5-acetyl-2-furyl) benzonitrile. ... Overheat the mixture to 83 degrees will result in 9% decrease in the outcome of p-(5-acetyl-2-furyl) benzonitrile.

Question: What will be the output of the process, if after 10 minutes of heating the mixture we increase the heat by 0.5 degrees per minute?"

This example describes chemical reactions between several components that yield 16 grams of p-(5-acetyl-2-furyl) benzonitrile. A model that attempts to answer the question following the text needs to extract the right information about the temperature and the heating duration described in the text, and to understand the output of the iterative process described in the question. Finally, it needs to perform a numerical computation and to check if the temperature reached the threshold of 83 degrees or not. We refer to these types of questions as Simulation Questions, as they require to simulate a process under other environment conditions.

In recent years, deep learning models have achieved state-of-the-art performance for question-answering (QA) and reading comprehension tasks. Pre-trained models like Roberta [20] and XLNet [32] are end-to-end models that achieved high performance on QA datasets like SQuAD [25] and SWAG [34]. Although these models achieved impressive results on various QA tasks, they mostly focus on span-based datasets, where the answer to the question must be a span from the text or questions that require simple commonsense reasoning. For simulation questions, one needs the ability to simulate the process at hand in addition to performing a numerical computation that would yield the final process product. Semantic parsing approaches, whose goal is to convert
between the semantic and syntactic similarity rewards and provide insights into the type of questions that NPS-SQA outperforms state-of-the-art QA models.

The contribution of our work is fourfold:

(1) We present a novel dataset for simulation questions of complex processes. It contains reference programs that can be executed to infer the right answers to the questions. As far as we know, this is the first question-answering dataset that requires deep understanding of the scientific processes’ dynamics and numerical commonsense reasoning capabilities. We contribute it to the community for further research.

(2) We define a domain-specific language (DSL) that is more concise than full featured programming languages like Python, but expressive enough with the ability to imitate the state of the process by defining variables, simulate iterative process using loops and examine multiple possibilities using if-conditions.

(3) We propose a novel reward function based on the state transition of the predicted and the reference programs to model the dynamics of the process. We use policy gradient training to train our model by rewarding it based on both the syntactic similarity and run-time similarity of the execution processes. We publish our code for the benefit of the community.

(4) We perform an empirical analysis comparing NPS-SQA and state-of-the-art deep-learning QA approaches showing significant boosts of 30%+ in accuracy compared to pretrained end-to-end question-answering QA models and 5%+ for neural program synthesis approaches trained on SimQA.

2 RELATED WORK

Question answering is a broad field that can be broken down into subdomains. Models that excel in one subdomain may perform poorly in others. Recently, pretrained models like ELMo[23], BERT[7], XLNET[32] and Roberta[20] have shown state-of-the-art performance on various QA tasks like SQUAD [25], RACE [16], GLUE [29] etc. However, these models perform poorly on tasks that require logical or numerical commonsense [19].

Semantic parsing approaches transform the text to a logical form that facilitates inference. These methods showed superior performance on tasks that require understanding of processes’ dynamics [3, 6, 8, 9, 11, 17, 18] and numerical commonsense reasoning [1, 5]. Semantic parsing approaches have two variations, weakly supervised approaches [3, 5, 6, 17, 18, 22] and fully supervised approaches. Our model is a fully supervised approach.

Fully supervised approaches use sequence-to-sequence architectures in a fully supervised manner to optimize the maximum marginal likelihood to a logical form given as a label (“the oracle”). For example, if the logical language is a computer program code, then the oracle is the predicted code or the target abstract syntax tree [8, 9, 11] corresponding to the text. Recent example of a semantic parsing model that achieved state-of-the-results on well-known benchmarks like Drop [10] and MathQA [1] is Neural Symbolic Reader(NeRD) [5] that has an encoder-decoder architecture and can be used as a weakly or fully supervised model.

1https://cutt.ly/FbcC0uu

Figure 1: A text describing a process and the corresponding code.

A mixture of 50 g (0.42 mole) of p-aminobenzonitrile, 146 ml of concentrated hydrochloric acid, and 106 ml of H2O was heated at 80° for 30 minutes... 

```
func simulation(){
  temp = 80;
  duration = 20;
  time_left = duration - 10;
  out = 16;
  if (temp > 83){
    return out * 0.91;
  }
  return out
}
```

What will be the output of the process if after 10 minutes of heating the mixture we increase the heat by 0.3 degree per minute?

Answer: 14.88
The aforementioned approaches deal with limited target DSL without variables, if branches and loops. Consider a representative question from MathQA: “A train running at the speed of 48 km / hr crosses a pole in 9 seconds . What is the length of the train?”. The logical form is "multiply(divide(multiply(48, const 1000), const 3600), 9)". Unlike the above example, chemistry texts contain a description of iterative processes and conditional events, therefore, often require more expressive DSL. Consider the complex process example in Fig. 1 that requires the notion of loops and if-clauses in addition to numerical comprehension of multiple entities.

The complexity of the generated DSL requires a more nuanced optimization. Unlike previous approaches that focused on optimizing the syntax of the target program, our approach leverages the run-time information of the oracle program during optimization. We devise a reinforcement learning approach and during optimization compare the predicted code to the oracle code run-time variable state. Our reward is therefore composed of both syntactic and run-time similarity. This helps the model to converge even when the DSL is complex and the search space is large.

3 PROBLEM DEFINITION
Let \( \Sigma \) denote an input text, \( \Gamma \) denote a simulation question, and \( y \in R \) to denote the correct answer. The question-answering task is to find \( \pi : (\Sigma, \Gamma) \rightarrow y \), which maps the text and the question to \( y \), the correct answer. In this work, we focus on simulation questions. A simulation question is a question that asks someone to imagine what might happen. Formally, we focus on problems that can be solved by the auxiliary task of \( \pi ' : (\Sigma, \Gamma) \rightarrow \lambda \), which maps the text and the question to a code, s.t. \( \lambda() = y \), i.e., when executing \( \lambda \) it produces \( y \).

4 METHODOLOGY
4.1 Domain Specific Language
We start by defining a language \( \lambda \), s.t. a text can be matched to it. We aim to define a concise version of a full-featured programming language to reduce the size of the search space of possible programs thus potentially leading to better convergence. We define this domain-specific language (DSL) to resemble c-syntax language, but with a less complex syntax while still retaining the capability of defining variables, “if” branches, and loops to manipulate a variable state. The following is the grammar definition of the new language:

\[
\begin{align*}
\text{Prog} \ p & := \text{func simulation}() \{ \ s \} \\
\text{Stmt} \ s & := \text{repeat}(n) \{ \ s \} \ | \ \text{if}(c) \{ \ s \} \\
& \quad | \ \text{return} \ e \ ; \ \ a; \ \ | \ s_1; s_2 \\
\text{Cond} \ c & := \text{identifier} \ \text{op}_c \ e \\
\text{Assign} \ a & := \text{identifier} = e \\
& \quad | \ \text{identifier} = e \ \text{op} \ e \\
\text{expr} \ e & := \text{identifier} \ | \ n \ | \ -n \ | \ r \ | \ r \\
\text{op}_c & := < \ | \ > \ | \ >= \ | \ <= \ | \ != \ | \ == \\
\text{op} & := + \ | \ - \ | \ * \ | \ \backslash \ | \ \backslash \\
\end{align*}
\]

4.2 Neural Program Synthesis for Question-Answering
Assume we have a labeled dataset of \((\Sigma_i, \Gamma_i, \lambda_i)\). One can refer to the problem as a translation task, i.e., given a text \(\Sigma_i\) and a question \(\Gamma_i\) generate the corresponding code, \(\lambda_i\), that answers the question \(\Gamma_i\). The common approach [24, 30] is to use an encoder-decoder architecture that maximizes the likelihood (MLE) of the corresponding text. In our setting that corresponding text is the reference program. Recently, it has been shown that an additional step of optimization, applied after the model is trained, can significantly boost results [14, 26, 31]. Usually, reinforcement learning is applied during this optimization step. We adopted the architecture of the encoder-decoder model and used Transformers (as suggested in Attention is all you need paper [28]) and trained the model using a cross-entropy loss. We treat the trained decoder from the previous step (optimized for MLE) as a model-free agent and further optimize the code generation process using reinforcement learning.

We use beam search to sample \(N\) possible programs that have the highest sequence probability, calculated by the multiplication of the probability for each token that constructs the program. We denote the reward function as \(Q : (\lambda, \hat{\lambda}) \rightarrow R\) and apply it to the predicted programs. We take the \(S\) programs with the highest value, as determined by \(Q\). Because the target function is non-smooth we cannot use it as a loss function and backpropagate the gradient thus, we use the REINFORCE [13, 14] algorithm to optimize the...
Algorithm 1 Training

Input: N, S, B

1: batch ← []
2: model ← pre-train MLE model
3: for (Σ_i, G_i, λ_i) in dataset do
4:     n_top ← beam_search(model, Σ_i, G_i, N)
5:     s_top ← [{Q(λ, λ_i) for λ in n_top}]
6:     s_top ← sort(s_top) # sort by Q value
7:     batch ← batch + s_top[:S] # take top s
8:     if length(batch) = B then
9:         L ← -1 \[ \sum_{t \in batch} Q \log p(\hat{\lambda}_t | \Sigma_i, G_i) \]
10:        L.backward()
11:        batch ← []
12:    end if
13: end for

We consider a reward function, Q, that will measure both the syntactic similarity between \( \lambda_i \) and \( \hat{\lambda}_i \), and the semantic similarity. We denote the syntactic similarity as \( Q_{syntactic} \) (Section 4.3) and the semantic as \( Q_{semantic} \) (Section 4.4). We then set the final Q as the weighted sum of the syntactic and semantic scores:

\[
Q = y Q_{syntactic} + (1 - y) Q_{semantic}
\]  

4.3 Syntactic Neural Program Synthesis

Recent research has shown that an additional step of BLEU-score optimization, applied after a model is trained, can produce dramatic improvements in performance [14, 26, 31]. Intuitively, training the encoder-decoder architecture using cross-entropy loss for each token optimizes the max likelihood estimation (MLE) for each token separately, while BLEU-score optimization optimizes the sequence as a whole. The BLEU-score can give a quantitative estimation of the syntactic similarity of the predicted code with the reference code. Therefore, we define: \( Q_{syntactic} = BLEU(\hat{\lambda}_i, \lambda_i) \).

4.4 Semantic Neural Program Synthesis

Unlike translation-based approaches, we hypothesize that when generating code, it might not be sufficient to only require syntactic similarity to the reference program. A small syntax mistake can lead to a wrong output, syntax errors, or run-time errors. Consider the example presented in Figure 3. The left program simulates an iterative process that eventually outputs the sum of an arithmetic series while the right program calculates the sum of a geometric series. The two programs are close syntactically but semantically different. Therefore, we aim to capture the semantics of the code execution alongside the syntax.

A computer program stores data in variables. The contents of these variables during the code, at any given point in the program’s execution, is called a program’s state. We define semantic similarity of code programs as the similarity of the program’s states during code execution. Formally, we define \( S_t \) = \{\( s_1^t, s_2^t, ..., s_n^t \)\} as the state of the predicted program and \( \hat{S}_t \) = \{\( \hat{s}_1^t, \hat{s}_2^t, ..., \hat{s}_m^t \)\} as the state of the reference program at time \( t \). Note that if the program didn’t define \( s_i \) yet we treat its value as \( \epsilon \), a special symbol that defines that the variable has not been used.

We define \( I = (s_1, s_2, ..., s_m) \) and \( \hat{I} = (\hat{s}_1, \hat{s}_2, ..., \hat{s}_k) \) as the instructions of the predicted program and the oracle program correspondingly. We use function \( E : (S_t, \sigma_t) \rightarrow S_{t+1} \) to execute the instruction in time \( t \) and compute the next state at time \( t+1 \). We use these definitions to define a semantic reward function that rewards each state that is aligned with the reference code state.

Unlike \( Q_{syntactic} \) that measures the syntactic similarity between the generated code and \( \hat{\lambda}_i \), \( Q_{semantic} \) should capture the similarity of the states of the generated code and the reference program \( \lambda_i \). We aim to integrate the run-time semantic estimation into the training process. To evaluate the similarity of the code states at each step, \( \lambda_i \) and the generated code are executed and a reward is calculated:

\[
T_{max} = \max(|I|, |\hat{I}|), T_{min} = \min(|I|, |\hat{I}|)
\]

\[
Q_{semantic} = \frac{1}{T_{max}} \sum_{t=0}^{T_{min}} \begin{cases} 1, & \forall i \in [n], \hat{s}_i^t \in E(S_t, \sigma_t), s_i^t \in E(\hat{S}_t, \hat{\sigma}_t) \\ s_i^t = \hat{s}_i^t, & 0, \text{ otherwise} \end{cases}
\]

After the execution of both programs, the reward is estimated by the number of matched state transitions. Algorithm 2 presents the semantic reward algorithm. The input to the algorithm is the reference code and the predicted code. The state transitions after each line are recorded during code execution. A check is then performed if the two programs modified the variables in the same way and returned the same values.

Algorithm 2 Semantic Reward

Input: \( \lambda_i, \hat{\lambda}_i \)

1: r_state ← execute_and_get_state(\( \lambda_i \))
2: p_state ← execute_and_get_state(\( \hat{\lambda}_i \))
3: correct_states ← 0
4: \( T_{min} \leftarrow \min(\text{len}(r\_state), \text{len}(p\_state)) \)
5: for \( t = 0 \) to \( T_{min} \) do
6:     if all(r_state[t] = p_state[t]) then
7:         correct_states ← correct_states + 1
8:  end if
9:  end for
10: \( T_{max} \leftarrow \max(\text{len}(r\_state), \text{len}(p\_state)) \)
11: return correct_states/T_{max}
Note that we require that all the variables will have the same values for each execution timestamp, but we can think on other variations that allow equality only for part of the variables, or allow the state to be equal on different stages of the execution and not only in the same timestamp. We leave these experiments to future research.

5 EXPERIMENTAL METHODOLOGY

5.1 Datasets

The chemical reactions’ dataset contains descriptions of reactions between chemicals from the US patents published between 1976-2016. Each reaction is annotated with the corresponding components, their quantities and actions that affected the process outcome. Consider this example from the US Patent dataset:

“A mixture of 50 g (0.42 mole) of p-aminobenzonitrile, 146 ml of concentrated hydrochloric acid, and 106 ml of H2O was heated at 80 °C for 20 minutes and then cooled to 0°. The hexane was cooled and the resulting precipitate was filtered and air dried to yield 16 g (18%) of p-(5-acetyl-2-furyl) benzonitrile.”

The example describes the process’s components (yellow), their quantities (green) and the actions (red) that were taken to cause the components to interact with each other to yield 16g of p-(5-acetyl-2-furyl) benzonitrile. We used the US patents dataset to create the SimQA datasets – question-answering datasets on texts that contain descriptions of chemical processes and reactions. Chemists were asked to curate a set of questions and answers over entities, actions and quantities. We provide two datasets based on those questions, SimQA-v1 and SimQA-v2. For SimQA-V2, the chemists were only allowed to create questions about the reaction described. Below is an example of such a question from the above text:

“How many milliliters of water we need to add to yield 25 grams of the product if we want to preserve the same ratio between them?”.

For SimQA-V1, the chemists were allowed to add additional plausible information about the reaction to allow the creation of more advanced questions. The context is added to the dataset. Below is an example of such context:

Additional Context: If the temperature exceeds 100 degrees when heating the mixture it causes the water to be evaporated at a rate of 2.3 milliliters per minute.

and the corresponding question:

How many milliliters of water should we add if we increase the temperature by 2 degrees per minutes for 10 minutes and we want to cancel the effect of the evaporation?

The datasets contain 6187 training examples (3158 in SimQA-V1 and 3029 in SimQA-v2) and 1089 test examples (555 in SimQA-V1 and 534 in SimQA-V2).

6 BASELINES AND METRICS

To evaluate whether we could benefit from solving the auxiliary problem of finding the program that answers the question instead of answering the question directly, we compare against state-of-the-art question-answering models: Roberta [20], XLnet [32] and Longformer [2]. We use pre-trained models and fine-tune them on the SimQA training datasets.

To compare our approach to other semantic parsing approaches we used Seq2Prog [1], an encoder-decoder architecture based on LSTM models which leverages attention mechanism to decode the right program, and NeRd [5] (its fully-supervised version) that also uses encoder (the “reader”) decoder (the “programmer”) architecture based on BERT and LSTM models that achieved state-of-the-art performance on MathQA [1] and Drop [10] datasets – both are a well-known benchmark for commonsense inference and logical reasoning tasks.

For the NPS-SQA, after hyper-parameters tuning (separate validation set), we chose to use a beam width of 32 sequences. We set \( N = 8, S = 4 \) and \( B = 32 \). All the models were evaluated using the accuracy metric, i.e., the percentage of correct answers.
7 EXPERIMENTS AND RESULTS

The nature of our task is a regression task where the model aims to answer the question directly with a numeric answer. We adapt state-of-the-art (SOTA) models, such as Roberta, to a regression setting by adding a dense linear layer on top of the output vector corresponding to the [CLS] token with dropout that will serve as the final regression layer. Tables 1-2 report the open-ended-question format results. SOTA models in question-answering, as Roberta [20], achieve close to 0 accuracy with MSE above 15000, whereas models that perform code generation, such as NPS-SQA, reach significantly lower MSE and high accuracy.

Table 1: Open-ended question results SimQA V1

| Model                  | MSE   | Accuracy |
|------------------------|-------|----------|
| Roberta                | 15949 | <0.001   |
| NPS-SQA (This Work)    | 1189  | 0.8801   |

Table 2: Open-ended question results SimQA V2

| Model                  | MSE   | Accuracy |
|------------------------|-------|----------|
| Roberta                | 26541 | <0.001   |
| NPS-SQA (This Work)    | 1900  | 0.8431   |

To further compare the performance of NPS-SQA with SOTA we perform an analysis of the models on multiple-question answer datasets, where SOTA algorithms have shown higher performance in other tasks [12]. We created a multiple-answer dataset as follows: in addition to the true answer, we generate additional three possible answers for each question to match the multiple-choice task requirements: we generated at random an answer close in its value to the correct answer to confuse the model if its calculations were slightly off, and the other two answers were randomly generated to be far off from the correct answer in order to catch answers far from the correct answer. This allows us to compare against state-of-the-art models for the end-to-end approach. We draw the reader attention that code-generating models, such as Seq2Prod, Nerd and NPS-SQA in this setting continue to generate the numeric answer directly and are not optimized for multiple answers questions.

Table 3 presents the main results of our approach on the two SimQA datasets. Statistically significant results are shown in bold. We observe that NPS-SQA outperforms current state-of-the-art methods significantly with a boost of 33% - 36% in accuracy against the end-to-end models. This shows the merit of using code generation as an auxiliary task. Additionally, we observe a boost of 4% - 5% compared to other semantic parsing approaches. This shows the merit of code execution as part of the run-time similarity between the oracle program and the predicted program during the training process. The SimQA-v1 dataset contains questions that require an understanding of the process and chain of events that lead to the final product. As we show in 8.3 this is the strength of NPS-SQA which explain the better performance of NPS-SQA on SimQA-v1 over SimQA-v2.

8 THE SYNTACTIC AND SEMANTIC TRADE-OFF

The reward function of $Q$ is composed of both syntactic and semantic similarity (Eq. 2). To better understand the right balance between the syntactic reward, $Q_{syntactic}$, and the semantic reward, $Q_{semantic}$, we experimented with different values of $\gamma$. Figure 4 indicates that the optimal value for $\gamma$ is around 0.5 on both datasets. This strengthens the hypothesis that both syntactic and semantic similarity help improve the performance.

8.1 Syntactic Reward

We observe that for $\gamma = 1$, where only the syntactic similarity is used during the reward, the model reaches 0.8253 and 0.8082 for SimQA-v1 and SimQA-v2 respectively which match the performance of other approaches based on syntactic similarity.

8.2 Semantic Reward

We observe that for $\gamma = 0$, where only the semantic run-time code similarity is used during the reward, the model reaches 0.8313 and 0.8112 for SimQA-v1 and SimQA-v2 respectively. This shows that even if we use only state-score optimization after MLE optimization we can boost the performance of the model. The state-score also optimizes the model prediction for the sequence like the Syntactic Reward but also add another layer of run time information to the training phase.

Table 3: Main results

| Model                  | SimQA v1 | SimQA v2 |
|------------------------|----------|----------|
| Roberta                | 0.5233   | 0.5112   |
| XLnet                  | 0.5421   | 0.5102   |
| Longformer             | 0.5291   | 0.4991   |
| Seq2Seq [1]            | 0.8013   | 0.7921   |
| Seq2Prog [1]           | 0.8279   | 0.8021   |
| NeRd [5]               | 0.8261   | 0.8041   |
| NPS-SQA (This Work)    | 0.8801   | 0.8431   |

Figure 4: The performance of the model with the syntactic-semantic reward for different $\gamma$ (gamma) values.
8.3 Strengths of NPS-SQA

Mapping the text and the question to code as an intermediate step helps the model to find the chain of events that lead to the correct output. We observed that this approach performs well when dealing with questions that require some complex numerical calculation or execution of iterative loops and if branches. Formally, we define a Complex Simulation Question as a question that requires a code that includes at least one if branch and one loop branch. This code needs to modify the state of the variables more than 15 times. An example of such a question is:

**Context:** A mixture of 5-hydroxy-1-tetralone (2.00 g) and sodium hydroxide (493 mg) was warmed in ethanol (40 mL) to 50° C. ... If the temperature passes 56 degrees when heating ethanol for more than 5 seconds it will result in a loss of 2.04 grams in the final products for each additional second.

**Question:** How many grams the process would yield if we increase the temperature when heating ethanol by 2 degrees every 2 seconds for 16 seconds?

We extracted a subset of 443 examples with these characteristics and compared the performance of NPS-SQA to Roberta, NeRd and Seq2Prog models after training on the other examples.

Table 6 shows that our approach outperforms these models by a large margin on those questions that are prevalent when dealing with texts that describe scientific processes.

| Model      | Accuracy - Complex QA |
|------------|-----------------------|
| Roberta    | 0.53                  |
| Seq2Prog   | 0.82                  |
| NeRd       | 0.82                  |
| NPS-SQA    | 0.90                  |

(2) “What Will be the Outcome” Questions. We observed that in outcome questions, our model is able to recognize the structure present in the context. By introducing a change in the question relative to a pre-existing condition (such as an increase in temperature), our model was able to successfully handle these types of questions as well.

**Context:** A mixture of tert-butyl3-morpholine-4-carboxylate (18f, 2.3 g) and ammonium acetate (4.1 g) was heated to 12 ° C. in toluene (40 ml) for 18 h. The mixture was allowed to cool and was evaporated. The residue was partitioned between aq. sodium bicarbonate and DCM. The dried extracts were evaporated and the residue purified on silica gel. Elution with DCM:EtOH:NH3: 400:8:1 gave a light brown solid (1.42 g). If the temperature exceed 24 degrees when heating toluene it will result in 13% decrease in the final products.

**Question:** What will be the outcome of the process in grams if the temperature increases by 19 degrees?

Table 6: Performance for “What Will be the Outcome” complex questions

| Model      | Accuracy - Complex QA |
|------------|-----------------------|
| Roberta    | 0.59                  |
| Seq2Prog   | 0.80                  |
| NeRd       | 0.79                  |
| NPS-SQA    | 0.92                  |

8.4 Limitations of NPS-SQA

The code generation approach works well on complex questions that involve state transitions or complex numeric calculations. However, if the question is simple the performance boost compared to baselines is lower. We define a Simple Simulation Question as a question that requires a code that includes no more than one if branch and no loop branches. This code is allowed to change the state no more than 5 times. For example:
Context: Prepared as described in example 7A, are refluxed at 160° C. for 16 hours in a mixture of 9 ml of dimethylformamide, 1.3 ml of 85% formic acid and 0.5 ml concentrated hydrochloric acid. After the reaction m... with benzene. 4 g of the title product are obtained. If the temperature exceed 175 degrees when heating dimethylformamide it hurts the process.

Question: What will be the outcome of the process in grams if the temperature increases by 14 degrees?

We gathered 361 simple questions examples and compared the performance of NPS-SQA against the same baselines after training on the other examples. For such questions, NPS-SQA still outperforms the performance of the state-of-the-art end-to-end model but matches the performance of the semantic parsing approaches (see Table 7).

Table 7: Performance for simple questions

| Model     | Accuracy - Simple QA |
|-----------|-----------------------|
| Roberta   | 0.6027                |
| Seq2Prog  | 0.8215                |
| NeRd      | 0.8247                |
| NPS-SQA   | 0.8231                |

Additional family of inaccuracies we have identified are attributed to misattribution of values to code variables. Below are a few qualitative examples:

(1) Value Misattribution Errors We have observed multiple cases of errors arising from value misattribution of text values to code variables. Consider the example below.

Context: A 2 liter glass flask equipped with a thermometer... an aqueous solution of sodium bisulfite was added until a clear solution was formed. 100 g of methanol and in total 400 g of 50% aqueous sulfuric acid were added. Flash distillation of the reaction mixture resulted in a two phase distillate. Fractionation of the lower phase (120 g) gave 85.5 g of CF3O(CF2)3OCHFCF2COOCH3 (bp 34-35 °C./6 mbar; yield 50%). If we heat methanol to temperature higher than 95 degrees, it causes the methanol to be evaporated at a rate of 1 gram per minute.

Question: How many grams of methanol should we add if we increase the temperature from 55 degrees by 5 degrees per minute for 22 minutes and we want to cancel the effect of the evaporation?

Figure 5 displays both the generated and target codes. In this example, our model’s output deviated significantly from the target result. By examining the code, it appears that the confusion may have been caused by the model adding the value after the word ‘total’ (400g/0.5) instead of 100g of methanol.

(2) Unit of Measure Conversion Errors Another type of errors occurs to a common case of units of measurement conversion. Consider the example below:

Context: A mixture of 273 mg (0.94 mmole) azetidinone from Step A, 26.3 mg paraformaldehyde and 178 mg (0.56 mmole) cesium carbonate was stirred in 20 ml dry tetrahydrofuran at ambient temperature 16.5 hours under nitrogen. A mixture of 430 1/4 pyridine and 2.56 ml acetic anhydride was added to the reaction mixture and the stirring continued 5 more hours. The solvents were removed in vacuo to give 604 mg crude product which was chromatographed on a silica gel flash column in hexane-ethyl acetate 3/1. This gave 102 mg (30%) of 1-acetoxymethyl-4-p-nitrophenylthi-3-n-propylazetidin-2-one. stirring the mixture longer after adding tetrahydrofuran will cause a loss of 21.96 milligrams in the final product for each hour above the original time.

Question: How many milligrams the process would yield if we stir the reaction for 31 hours?

As we can observe from the example in Figure 6, the generated code is similar to the target code, except for the value assigned to the output solution. Instead of utilizing the quantity of the crude product, the model erroneously used the quantity of the net product (102 mg instead of 604 g). Additionally, the model did not take into account the different
units of measurement, leading to a significant difference in the reward function.

8.5 Visualization and Model Explainability

Another advantage of the code generation approach is the fact that we can explain the model’s predictions. Using the attention layers of our model, we can understand why the model chose to use loops and if branches or why it chose to perform a certain manipulation to the variable state. Figure 7 illustrates one such example. The different colored tokens in the text and the question mark the top 5 tokens that have the highest attention value when predicting the corresponding colored token in the generated code. We can see that when the “repeat” token was generated, the model paid attention to the “per” and “evaporation” tokens which indicate that it learns to simulate the iterative evaporation process by means of a “repeat” loop. When the model generated the token “if” it paid attention to the tokens in the text related to temperature. Which means it learned that it should understand whether the temperature has passed a certain threshold.

9 CONCLUSIONS

In this work, we explored the problem of simulation question-answering. We presented an approach that tackles this problem by generating a code that simulates the process described in the text and the environment conditions of the question. We automatically construct a dataset of descriptions of chemical processes. Chemists then curate possible questions and answers for those process descriptions. Each description and question were mapped to codes of the process described. We train an encoder-decoder model that optimizes both for syntactic similarity and semantic code similarities. Once the code is generated it is executed and outputs the answer to the question. We show that such approach significantly boosts performance by more than 30% in accuracy as compared to the state-of-the-art QA. We study the semantic-syntactic trade-off and identify that both are needed for successful question-answering and process simulation. We implemented a reward function that incentivizes both syntactic similarity to the target code and similarity in variable values during run-time. Additionally, we identify that questions that require complex simulation of process, with if-branches and loops, especially benefit from the intermediate task of code generation and execution. We also show the potential of explainability of the model using the generated code. Our model has shown to be capable of extracting parameter values from text, even if they’re not explicitly stated. For instance, “dropped 5 grams of methanol a minute for 10 minutes” becomes $5 \times 10 = 50$ in the generated code. However, the model sometimes makes errors in value detection, such as confusing values detected by a whole order of magnitude for example “mg” and “grams”. This potential source of confusion should be addressed in future work. We also wish to study the integration of external facts and world knowledge to code generation. In this work, all the information that is needed to answer the question can be found in the text itself without the need to combine knowledge from external sources. For example, the model does not know that the boiling temperature of water is 100°C if it was not written in the text explicitly. Additional such knowledge can significantly improve broad simulation-question answering.

REFERENCES

[1] Aida Amini, Saadia Gabriel, Shanchuan Lin, Rik Konnel-Kedzierski, Yejin Choi, and Hannaneh Hajishirzi. 2019. MathQA: Towards Interpretable Math Word Problem Solving with Operation-Based Formalisms. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL 2019).

[2] Iz Beltagy, Matthew E. Peters, and Arman Cohan. 2020. Longformer: The Long-Document Transformer. CoRR (2020).

[3] Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic Parsing on Freebase from Question-Answer Pairs. In Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, EMNLP 2013.

[4] Jonathan Berant, Vivek Srikumar, Pei-Chun Chen, Abby Vander Linden, Brittany Harding, Brad Huang, Peter Clark, and Christopher D. Manning. 2014. Modeling Biological Processes for Reading Comprehension. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014.

[5] Xinyun Chen, Chen Liang, Adams Wei Yu, Denny Zhou, and Quoc V. Le. 2020. Neural Symbolic Reader: Scalable Integration of Distributed and Symbolic Representations for Reading Comprehension. In 8th International Conference on Learning Representations, ICLR 2020.

[6] Pradeep Dasigi, Matt Gardner, Shikhar Murty, Luke Zettlemoyer, and Eduard Hovy. 2019. Iterative Search for Weakly Supervised Semantic Parsing. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies NAACL 2019.

[7] 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 (NAACL 2019).

[8] Li Dong and Mirella Lapata. 2016. Language to Logical Form with Neural Attention. In Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016. The Association for Computer Linguistics.
Li Dong and Mirella Lapata. 2018. Coarse-to-Fine Decoding for Neural Semantic Parsing. In Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018.

Pheeran Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. 2019. DROP: A Reading Comprehension Benchmark requiring Discrete Reasoning Over Paragraphs. In Proc. of NAACL 2019.

Robin Jia and Percy Liang. 2016. Data Recombination for Neural Semantic Parsing. In Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016. The Association for Computer Linguistics.

Namrata Kadam and M. Anand Kumar. 2022. Multiple Choice Question Answering Using Attention Based Ranking and Transfer Learning. In 2022 IEEE Region 10 Symposium (TENSYMP).

Mattis Manfred Kämmerer. 2019. On Policy Gradients. CoRR (2019).

Yaser keneshloo, Tian Shi, Naren Ramakrishnan, and Chandan K. Reddy. 2020. Deep Reinforcement Learning for Sequence-to-Sequence Models. IEEE Transactions on Neural Networks and Learning Systems (2020).

Chloé Kiddon, Luke Zettlemoyer, and Yejin Choi. 2016. Globally Coherent Text Generation with Neural Checklist Models. In Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016. The Association for Computational Linguistics.

Guokun Lai, Qihe Xie, Hanxiao Liu, Yiming Yang, and Eduard H. Hovy. 2017. RACE: Large-scale Reading Comprehension Dataset from Examinations. In Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017.

Chen Liang, Jonathan Berant, Quoc V. Le, Kenneth D. Forbus, and Ni Lao. 2017. Neural Symbolic Machines: Learning Semantic Parsers on Freebase with Weak Supervision. In Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017.

Chen Liang, Mohammad Norouzi, Jonathan Berant, Quoc V. Le, and Ni Lao. 2018. Memory Augmented Policy Optimization for Program Synthesis with Generalization. CoRR (2018).

Bill Yuchen Lin, Seyeon Lee, Rahul Khanna, and Xiang Ren. 2020. Birds have four legs?! NumerSense: Probing Numerical Commonsense Knowledge of Pre-Trained Language Models. In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP 2020). Association for Computational Linguistics.

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 (2019).

Daniel Lowe. 2017. Chemical reactions from US patents (1976-Sep2016). (2017).

Sewon Min, Danqi Chen, Hannaneh Hajishirzi, and Luke Zettlemoyer. 2019. A Discrete Hard EM Approach for Weakly Supervised Question Answering. 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 2019.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep Contextualized Word Representations. In Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018.

Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know What You Don’t Know: Unanswerable Questions for SQuAD. In Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics ACL 2018.

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, EMNLP 2016.

Marc Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. 2016. Sequence Level Training with Recurrent Neural Networks. In 4th International Conference on Learning Representations, ICLR 2016.

Niket Tandon, Bhavana Dalvi, Keisuke Sakaguchi, Peter Clark, and Antoine Bosselut. 2019. WIQA: A dataset for ‘what if...’ reasoning over procedural text. In Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems, NeurIPS 2017.

Alex Wang, Anampret Singh, Julian Michael, Felix Hall, Ober Levy, and Samuel R. Bowman. 2019. GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding. In 7th International Conference on Learning Representations, ICLR 2019.

Yan Wang, Xiaojing Li, and Shuming Shi. 2017. Deep Neural Solver for Math Word Problems. In Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017.

Lijun Wu, Li Zhao, Tao Qin, Jianhua Lai, and Tie-Yan Liu. 2017. Sequence Prediction with Unlabeled Data by Reward Function Learning. In Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017.

Zhulin Yang, Zhihong Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. 2019. XLM-Net: Generalized Autoregressive Pretraining for Language Understanding. In Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems, NeurIPS 2019.

Ziyu Yao, Xinjun Li, Jianfeng Gao, Brian M. Sadler, and Huan Sun. 2019. Interactive Semantic Parsing for If-Then Recipes via Hierarchical Reinforcement Learning. In The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, (EAAI 2019).

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, EMNLP 2018.