

A APPENDIX

A.1 QUALITATIVE ANALYSIS

Weakly Supervised Neuro-Symbolic Module Network	GenBERT
1. Query: how many times did a game between the patriots versus colts result in the exact same scores?, Ans: 2 Num. of Passage Entities: Date(10), Number(9)	
D, N = <i>Entity-Attention</i> ('how many times') // D, N are the attention distribution over date and number entities D1, N1 = <i>Entity-Attention</i> ('did a game between the patriots versus colts result in the exact same scores', (D, N)) 'Number', 'Count' = <i>EntType-Operator-Selector</i> ('how many times', Query) Answer 2 = <i>Count</i> (N1)	Predicted AnsType: Decoded Decoder output: 2 Span extracted: "colts" Answer = 2
2. Query: how many people in chennai, in terms of percent population, are not hindu?, Ans: 19.3 Num. of Passage Entities: Date(2), Number(26)	
D, N = <i>Entity-Attention</i> ('how many people in chennai, in terms of percent population') D1, N1 = <i>Entity-Attention</i> ('are not hindu', (D, N)) 'Number', 'Negate' = <i>EntType-Operator-Selector</i> ('are not hindu', Query) 1 = <i>Count</i> (N) {80.7} = <i>Sample-Arbitrary-Arguments</i> (N1, 1) Answer = 19.3 = <i>Negate</i> ({80.7})	Predicted AnsType: Decoded Decoder output: 19.3 Span extracted: "80.7" Answer = 19.3
3. Query: how many more percent of the population was male than female?, Ans: 0.4 Num. of Passage Entities: Date(4), Number(29)	
D, N = <i>Entity-Attention</i> ('how many') D1, N1 = <i>Entity-Attention</i> ('more percent of the population was male', (D, N)) D2, N2 = <i>Entity-Attention</i> ('than female', (D, N)) 'Number', 'Difference' = <i>EntType-Operator-Selector</i> ('how many', Query) 50.2 = <i>Sample-1-Argument</i> (N1) 49.8 = <i>Sample-2-Argument</i> (N2) Answer = 0.4 = <i>Difference</i> ({50.2, 49.8})	Predicted AnsType: Decoded Decoder output: 3.2 Span extracted: "49.8" Answer = 3.2
4. Query: how many more, in percent population of aigle were between 0 and 9 years old than are 90 and older?, Ans: 9.8 Num. of Passage Entities: Date(0), Number(25)	
D, N = <i>Entity-Attention</i> ('how many more') D1, N1 = <i>Entity-Attention</i> ('in percent population of aigle were between 0 and 9 years old', (D, N)) D2, N2 = <i>Entity-Attention</i> ('than are 90 and older', (D, N)) 'Number', 'Difference' = <i>EntType-Operator-Selector</i> ('how many more', Query) 10.7 = <i>Sample-1-Argument</i> (N1) 0.9 = <i>Sample-1-Argument</i> (N2) Answer = 9.8 = <i>Difference</i> ({10.7, 0.9})	Predicted AnsType: Decoded Decoder output: 1.7 Span extracted: "0.9" Answer = 1.7
5. Query: going into the 1994 playoffs, how many years had it been since the suns had last reached the playoffs?, Ans: 3 Num. of Passage Entities: Date(3), Number(17)	
D, N = <i>Entity-Attention</i> ('going into the 1994 playoffs : how many years') D1, N1 = <i>Entity-Attention</i> ('had it been since the suns had last reached the playoffs', (D, N)) 'Date', 'Difference' = <i>EntType-Operator-Selector</i> ('going into the 1994 playoffs : how many years', Query) {1991, 1994} = <i>Sample-2-Argument</i> (D) Answer = 3 = <i>Difference</i> ({1991, 1994})	Predicted AnsType: Decoded Decoder output: 7 Span extracted: "1991" Answer = 7
6. Query: how many more points did the cats have in the fifth game of the AA championship playoffs compared to st. paul saints?, Ans: 3 Num. of Passage Entities: Date(3), Number(12)	
D, N = <i>Entity-Attention</i> ('how many') D1, N1 = <i>Entity-Attention</i> ('more points did the cats have in the fifth game of the AA championship playoffs', (D, N)) D2, N2 = <i>Entity-Attention</i> ('compared to the st. paul saints', (D, N)) 'Number', 'Difference' = <i>EntType-Operator-Selector</i> ('how many', Query) 5.0 = <i>Sample-1-Argument</i> (N1) 2.0 = <i>Sample-1-Argument</i> (N2) Answer = 3.0 = <i>Difference</i> ({5.0, 2.0})	Predicted AnsType: Decoded Decoder output: 3 Span extracted: "4 - 1 in the fifth game" Answer = 3
7. Query: how many total troops were there in the battle?, Ans: 40000 Num. of Passage Entities: Date(1), Number(3)	
D, N = <i>Entity-Attention</i> ('how many total troops') D1, N1 = <i>Entity-Attention</i> ('were there in the battle', (D, N)) 'Number', 'Sum' = <i>EntType-Operator-Selector</i> ('how many total troops', Query) 2 = <i>Count</i> (N1) {10000.0, 30000.0} = <i>Sample-Arbitrary-Arguments</i> (N1, 2) Answer = 40000.0 = <i>Sum</i> ({10000.0, 30000.0})	Predicted AnsType: Decoded Decoder output: 100000 Span extracted: "10000 korean troops" Answer = 100000

Weakly Supervised Neuro-Symbolic Module Network	NMN- <i>num</i>	GenBERT
8. Query: how many field goals did sebastian janikowski and kris brown both score each? Ans: 2 Num. of Passage Entities: Date(0), Number(9)		
D, N = <i>Entity-Attention</i> ('how many field goals') D1, N1 = <i>Entity-Attention</i> ('did sebastian janikowski and kris brown both score each', (D, N)) 'Number', 'Count' = <i>EntType-Operator-Selector</i> ('how many field goals', Query) Answer = 2.0 = <i>Count</i> (N1)	P1 = <i>Find-Passage-Attention</i> () P2 = <i>Filter-Passage-Attention</i> (P1) 2 = <i>Passage-Attn-To-Count</i> (P2) Answer = 2	Predicted AnsType: Decoded Decoder output: 2 Span extracted: "33 - yard" Answer = 2
9. Query: how many years was between the oil crisis and the energy crisis? Ans: 6 Num. of Passage Entities: Date(19), Number(14)		
D1, N1 = <i>Entity-Attention</i> ('was between the oil crisis and the energy crisis') D, N = <i>Entity-Attention</i> ('how many years', D1, N1) 'Date', 'Difference' = <i>EntType-Operator-Selector</i> ('how many years', Query) {1973, 1979} = <i>Sample-2-Argument</i> (D) Answer = 6.0 = <i>Difference</i> ({1973, 1979})	$year-diffs \in \mathbb{R}^{40}$ (<i>//</i> generated exhaustive output space of all differences) P1 = <i>Find-Passage-Attention</i> () 6 = <i>Year-Difference</i> (P1, <i>year-diffs</i>) Answer = 6.0	Predicted AnsType: Decoded Decoder output: 3 Span extracted: "1973" Answer = 3
10. Query: how many yards was the longest touchdown pass? Ans: 40 Num. of Passage Entities: Date(0), Number(5)		
D, N = <i>Entity-Attention</i> ('how many yards was the') D1, N1 = <i>Entity-Attention</i> ('longest touchdown pass', (D, N)) 'Number', 'Sum' = <i>EntType-Operator-Selector</i> ('how many yards was the', Query) 1 = <i>Count</i> (N) {40.0} = <i>Sample-Arbitrary-Argument</i> (N, 1) Answer = 40.0 = <i>Sum</i> ({40.0})	P1 = <i>Find-Passage-Attention</i> () N1 = <i>Find-Passage-Number</i> (P1) 40 = <i>Find-Max-Num</i> (N1) Answer = 40	Predicted AnsType: Extract-Span Decoder output: 43 Span extracted: "40" Answer = 40

Table 4: Example questions from DROP-*num* along with predictions of the Proposed model WNSMN and the best performing versions of the NMN-*num* and GenBERT baselines from Table II. Detailed elaborations of outputs of these three models below:

(i) **WNSMN** first parses the dependency structure in the query into a program-form. Next, for each step of the program, it generates an attention distribution over the date and number entities. *Entity-Attention* refers to that learnt entity-specific cross attention described in §2.1.1. It then performs the discrete reasoning by sampling an operation and specific entity-arguments, in order to reach the answer. *EntType-Operator-Selector* refers to the Entity-Type and Operator Predictor in Operator Sampling Network and *Sample-*-Argument* refers to the Argument Sampling Network described in §2.1.2. *Sum/Difference/Logical-Not* are some of the discrete operations that are executed to get the answer. In some of the cases, (e.g., Query 3.) despite wrong parsing the model was able to predict the correct operation even though the root clause did not have sufficient information. In Query 10., the correct operation is *Max*, but WNSMN reaches the right answer by sampling only the maximum number entity through the *Sample-Arbitrary-Argument* network and then applying a spurious *Sum* operation on it.

(ii) On the other hand, the steps of the program generated by **NMN-*num*** first compute or further filter attention distribution over the passage or entities which are then fed into the learnable modules (*Passage-Attn-To-Count*, *Year-Difference*) that predict the answer. In order to do so, it needs to precompute all possible outputs of numerical operations that generate new numbers for e.g. *year-diffs* in Example 9. Because of the relatively poorer performance of NMN-*num*, its outputs are only reported for the last 3 instances, which were cherry-picked based on NMN-*num*'s predictions.

(iii) **GenBERT** first predicts whether the answer should be decoded or extracted from passage span and accordingly uses the Decoder output or extracted span as the answer. By design, the modular networks provide a more interpretable output than the monolithic encoder-decoder model GenBERT.

A.2 IMPLEMENTATION & PSEUDO-CODE

The source-code and models pertaining to this work would be open-sourced on acceptance of this work. A detailed pseudo-code of the WNSMN algorithm is provided below.

Algorithm 1: WNSMN Algorithm

Input: (Query (q), Passage (p)) = x
Output (or Supervision): Answer(y) $\in \mathbb{R}$

Preprocessing:
 $[num_1, num_2, \dots, num_N] = Num = \text{Extract-Numbers}(p)$ *// Number and Date*
 $[date_1, date_2, \dots, date_D] = Date = \text{Extract-Dates}(p)$ *// Entity and Passage Mentions*

Inference:
 $[(q_1, ref_1), \dots (q_k, ref_k), \dots (q_l, ref_l)] = Program = \text{Query-Parsing}(q)$
for step $(q_k, ref_k) \in Program$ **do**
 $(\mathbf{A}_k^{num}, \mathcal{T}_k^{num}), (\mathbf{A}_k^{date}, \mathcal{T}_k^{date}) = \text{Entity-Attention}(q_k, p, ref_k, Num, Date)$ §2.1.1
end for
 $\mathcal{L}_{aux}^{num}, \mathcal{L}_{aux}^{date} = \text{Entity-Inductive-Bias}(\mathbf{A}^{num}, \mathbf{A}^{date})$ Equation (1)
 $\mathcal{L}_{aux} = \mathcal{L}_{aux}^{num} + \mathcal{L}_{aux}^{date}$
 $q_l = \text{Query Span Argument of Last Step}$ *// Program Arguments and Stacked Attention*
 $ref_l = \text{Reference Argument of Last Step}$ *// Map over Entities for Last Step*
 $\mathcal{T}^{num} = \{\mathcal{T}_k^{num} | k \in ref_l\}, \mathcal{T}^{date} = \{\mathcal{T}_k^{date} | k \in ref_l\}$
 $Operators = \{op_1, op_2, \dots, op_{k1}\} = \text{Operator-Predictor}(q_l, q)$ *// Operator and EntityType*
 $EntTypes = \{type_1, type_2, \dots, type_{k1}\} = \text{Entity-Type-Predictor}(q_l, q)$ *// Sampling*
 $Actions = \{\}$ *// Action Sampling for each Operator*
for $op, type \in (Operators, EntTypes)$ **do**
if $type$ **is** Number **then**
 $\mathcal{T} = \mathcal{T}^{num}$
else if $type$ **is** Date **then**
 $\mathcal{T} = \mathcal{T}^{date}$
end if
if op **is** diff **then**
if $|ref_l| == 2$ **then**
 $arg1 = \{arg1_1, arg1_2, \dots, arg1_{k2}\} = \text{Sample-1-Argument}(\mathcal{T}_0)$
 $arg2 = \{arg2_1, arg2_2, \dots, arg2_{k2}\} = \text{Sample-1-Argument}(\mathcal{T}_1)$
 $args = \{(a1, a2) | (a1, a2) \in (arg1, arg2)\}$
else if $|ref_l| == 1$ **then**
 $args = \{arg_1, arg_2, \dots, arg_{k2}\} = \text{Sample-2-Argument}(\mathcal{T}_0)$
end if
else if op **is** count **then**
 $args = \{count_1, count_2, \dots, count_{k2}\} = \text{Count-Network}(\sum_j \mathcal{T}_j)$
else
 $args = \{arg_1, arg_2, \dots, arg_{k2}\} = \text{Sample-Arbitrary-Argument}(\sum_j \mathcal{T}_j)$
end if
 $probs = \{(p^{type} * p^{op} * p) | p \in p^{arg}\} \in \mathbb{R}^{k2}$ *// p's refer to the corresponding probabilities*
 $answers = \{\text{Execute-Discrete-Operation}(type, op, arg) | arg \in args\} \in \mathbb{R}^{k2}$
 $actions = \{(prob, answer) | prob \in probs, answer \in answers\}$
 $Actions = Actions \cup actions$
end for

Training:
for $i \in \{1, \dots, N_{IML} + N_{RL}\}$ **do**
for $(x, y) \in \mathcal{D}$ **do**
 $\mathcal{A}(x) \leftarrow Actions$ sampled for input(x) *// Using above Algorithm*
 $R(x, a, y) \leftarrow \text{Exact Match Reward for action } a \text{ for instance } x \text{ with gold answer } y$
if $i \leq N_{IML}$ **then**
 $(\theta, \phi) \leftarrow \max_{\theta, \phi} J^{IML} \text{ over } (\mathcal{A}, R) + \min_{\phi} \mathcal{L}_{aux}$ J^{IML} from Equation (2)
else
 $(\theta, \phi) \leftarrow \max_{\theta, \phi} J^{RL} \text{ over } (\mathcal{A}, R) + \min_{\phi} \mathcal{L}_{aux}$ J^{RL} from Equation (3)
end if
end for
end for

A.3 QUALITATIVE INSPECTION OF WNSMN PREDICTIONS

Good Action: Action Resulting in exact match with gold answer	
Correct Action: Action Manually annotated to be correct	
Number of test instances (DROP-num Test)	5800
Number of instances with atleast 1 good action	4868
Number of instances with more than 1 good action	2533
Average number of good actions (where there is atleast 1 good action)	1.5
Average number of good actions (where there is more than 1 good action)	2.25
Number of instances where the top-1 action is good action	2956
Number of instances where top-1 is the only good action	2335 (79% of 2956)
Number of instances with possibility of top-1 action being spuriously good	620 (21% of 2956)
Number of instances manually annotated (out of possible cases of spurious top-1 action)	334 (out of 620)
Number of instances where top-1 action is found to be spurious	28 (8.4% of 334)
Avg Ratio of Probability of Top Action and Maximum Probability of all other spuriously good actions (if any)	4.4e+11

Table 5: Analysis of the predictions of WNSMN on DROP-num Test

Generic Observations/Notes

- Note: When the model selects a single number in the Argument Sampling network and the Operator sampled is not of type count, we forcefully consider the operation as a NO-OP. For example sum/min/max over a single number or date is treated as NO-OP.
- One potential source of spuriously correct answer is the neural ‘counter’ module which can predict numbers in [1, 10]. However, out of the cases where atleast one of the top-50 actions is a good action we observe that the model is able to learn when the answer is directly present as an entity or can be obtained through (non count) operations over other entities and when it cannot be obtained directly from the passage but needs to aggregate (i.e., count) over multiple entities. Table 8 below gives some examples of *hard* instances where the WNSMN Top-1 prediction was found to be correct.

True Reasoning	Model Prediction	Count
<i>negate</i> a passage entity i.e., 100 - number	the model was able to select <i>negate</i> of the correct entity as the top action.	34
<i>min/max</i> of a set of passage entities	the model instead directly sampled the correct minimum/maximum entity as a single argument and then applied NO-OP operation over it.	11
<i>select</i> one of the passage entities	the model was able to select the right entity and apply NO-OP on it as the top action.	18
<i>count</i> over passage entities	the model was able to put count as the top action and the spurious actions came much lower with almost epsilon probability	88
<i>difference</i> over passage entities (the same answer could be spuriously obtained by other non-difference operations over unrelated entites)	the model was able to put difference as the top action and the spurious actions came much lower with almost epsilon probability	89
<i>difference</i> over passage entities (the same answer could be spuriously obtained by difference over other unrelated entities)	the model was able to put difference over the correct arguments as the top action	66

Table 6: Case Study of the 306 instances manually annotated as Correct out of 334 instances

True Reasoning	Model Prediction	Count
difference of dates/months	count over years	4
sum(number1, count([number2]))	count over numbers	1
difference between entities	sum over two arguments (both arguments wrong)	1
difference between entities	difference over two arguments (both arguments wrong)	1
difference between entities	count over entities	1
difference between entities	sum over arguments (one correct) (correct action was taken in one of the other top-5 beams)	2
question is vague/incomplete/could not be answered manually	count or difference	2
counting over text spans (Very rare type of question, only 2 found out of 334)	wrong operator	2
miscellaneous	wrong operator	7
miscellaneous	correct operator wrong arguments (one correct)	2
miscellaneous	correct operator wrong arguments (all wrong)	5

Table 7: Case Study of the 28 instances manually annotated as Wrong out of 334 instances.

Question	Relevant Passage Excerpt	Model Prediction Analysis
How many printing had Green Mansions gone through by 1919?	"W. H. Hudson which went through nine printings by 1919 and sold over 20,000 copies.... "	Model was able to rank the operation sum([9,0]) highest. the <i>count-number</i> operator had near-epsilon probability, indicating that indeed it did not find any indication of the answer being 9 by counting entities over the passage. This is despite the fact that most of the "how many" type questions need counting.
The Steelers finished their 1995 season having lost how many games difference to the number of games they had won?	"In 1995, the Steelers overcame a 3-4 start (including a 20-16 upset loss to the expansion 1995 Jacksonville Jaguars season) to win eight of their final nine games and finished with an record, the second-best in the AFC".	Model had to avoid distracting numbers (3,4) and (20,16) to understand that the correct operation is difference of (9-8)
How many more field goals did Longwell boot over Kasay?	"26-yard field goal by kicker Ryan Longwell ... Carolina got a field goal with opposing kicker John Kasay. ... Vikings would respond with another Longwell field goal (a 22-yard FG) ... Longwell booted the game-winning 19-yard field goal "	Question needed counting of certain events and none of these appeared as numbers. Model was able to apply count over number entities correctly
How many delegates were women from both the Bolshevik delegates and the Socialist Revolutionary delegates?	"Of these mandatory candidates, only one Bolshevik and seven Socialist Revolutionary delegates were women."	Model was able to apply sum on the correct numbers, even though many of the "how many" type questions need counting
How many years in a row did the GDP growth fall into negatives?	"Growth dropped to 0.3% in 1997, -2.3% in 1998, and -0.5% in 1999."	Model had to understand which numbers are "negative". It also needed to understand to count the two events instead of taking difference of the years
At it's lowest average surface temperature in February, how many degrees C warmer is it in May?	"The average surface water temperature is 26-28 C in February and 29 C in May."	Passage had distractive unrelated numbers in the proximity but the model was able to select the lowest temperature out of (26,28) and then take difference of (29-26)
How many years ibefore the blockade was the Uqair conference taken place?	"Ibn Saud imposed a trade blockade against Kuwait for 14 years from 1923 until 1937... At the Uqair conference in 1922, ... "	Passage had other distracting unrelated numbers in the proximity but the model was able to select the correct difference operation

Table 8: Manual Analysis of a few *hard* instances (with Question and Relevant Passage Excerpt) where WNSMN top-1 prediction was found to be correct

A.4 BACKGROUND: NUMERICAL REASONING OVER TEXT

The most generic form of Numerical reasoning over text (NRoT) is probably encompassed by the machine reading comprehension (MRC) framework (as in [Dua et al. \(2019\)](#)), where given a long passage context, c , the model needs to answer a query q , which can involve generating a numerical or textual answer or selecting a numerical quantity or span of text from the passage or query. The distinguishing factor from general RC is the need to perform some numerical computation using the entities and numbers in the passage to reach the goal.

Discrete/symbolic reasoning in NRoT: In the early NRoT datasets [Hosseini et al. (2014); Roy & Roth (2015); Koncel-Kedziorski et al. (2016)] which deal with simpler math word problems with a small context and few number entities, symbolic techniques to apply discrete operations were quite popular. However, as the space of operations grow or the question or the context becomes more open-ended these techniques fail to generalize. Incorporating explicit reasoning in neural models as discrete operations requires handling non-differentiable components in the network which leads to optimization challenges.

Discrete reasoning using RL: Recently Deep Reinforcement Learning (DRL) has been employed in various neural symbolic models to handle discrete reasoning, but mostly in simpler tasks like KBQA, Table-QA, or Text-to-SQL [Zhong et al. (2017); Liang et al. (2018; 2017); Saha et al. (2019); Ansari et al. (2019); Neelakantan et al. (2017)]. Such tasks can be handled by well-defined components or modules, with well structured function-prototypes (*i.e.*, function arguments can be of specific variable-types *e.g.*, KB entities or relations or Table row/column/cell values), which can be executed entirely as a symbolic process. On the other hand, MRC needs more generalized frameworks of modular networks involving fuzzy forms of reasoning, which can be achieved by *learning* to execute the query over a sequence of learnable neural modules, as explored in [Gupta et al. (2020)]. This was inspired by the Neural Modular Networks which have proved quite promising for tasks requiring similar fuzzy reasoning like Visual QA [Andreas et al. (2016; 2015)].

SoTA models on DROP: While the current leaderboard-topping models already showcase quite superior performance on the reasoning based RC task, it needs closer inspection to understand whether the problem has been indeed fully solved.

Pre-trained Language Models: On one hand, the large scale pretrained language models [Geva et al. (2020)] use Transformer encoder-decoder (with pretrained BERT) to emulate the input-output behavior, decoding digit-by-digit for numeric and token-by-token for span based answers. However such models perform poorly when only trained on DROP and need additional synthetic dataset of numeric expressions and DROP-like numeric textual problems, each augmented with the *gold numeric expression* form.

Reasoning-free Hybrid Models: On the other hand, a class of *hybrid* neural models have also gained SoTA status on DROP by explicitly handling the different types of numerical computations in the standard extractive QA pipeline. Most of the models in this genre, like NumNet [Ran et al. (2019)], NAQANet [Dua et al. (2019)], NABERT+ [Kinley & Lin (2019)], MTMSN [Hu et al. (2019)] and NeRd [Chen et al. (2020)] do not actually treat it as a reasoning task; instead they precompute an exhaustive enumeration of all possible outcomes of numerical and logical operations (*e.g.*, *sum/diff, negate, count, max/min*) and augment the training data with knowledge of the query-type (depending on reasoning-type) and *all* the numerical expression that leads to the correct answer. This reduces the question-answering task to simply learning a multi-type answer predictor to classify into the reasoning-type and directly predict the numerical expression, thus alleviating the need for rationalizing the inference or handling any (non-differentiable) discrete operation in the optimization. Some of the initial models in this genre are NAQANet [Dua et al. (2019)] and NumNet [Ran et al. (2019)] which are respectively numerically aware enhancements of QANet [Yu et al. (2018)] and the Graph Neural Networks. These were followed by BERT-based models, NABERT and NABERT+ [Kinley & Lin (2019)], *i.e.* a BERT version of the former, enhanced with *standard numbers* and *expression templates* for constraining numerical expressions. MTMSN [Hu et al. (2019)] models a specialized multi-type answer predictor designed to support specific answer types (*e.g.*, count/negation/add/sub) with supervision of the arithmetic expressions that lead to the gold answer, for each type.

Modular Networks for Reasoning: NMN [Gupta et al. (2020)] is the first model to address the QA task through explicit reasoning by learning to execute the query as a specialized program over learnable modules tailored to handle different types of numerical and logical operations. However, to do so, it further needs to augment the training data with annotation of the *gold program* and *gold program execution* *i.e.* the *exact* discrete operation and numerical expression (*i.e.*, the numerical operation and operands) that leads to the correct answer for *e.g.*, the supervision of the gold numerical expression in Figure 1 is *SUM*(23, 26, 42). This is usually obtained through manual inspection of the data through regex based pattern matching and heuristics applied on the query language. However, because of the abundance of templated queries in DROP this pattern matching is in fact quite effective and noise-free, resulting in the annotations acting as strong supervision.

However such a manual intensive process severely limits the overall model from scaling to more general settings. This is especially true for some of the previous reasoning based models, NABERT+, NumNet and MTMSN which perform better on than NMN (infact achieve SoTA performance) on the full DROP dataset. But we do not consider them as our primary baselines, as, unlike NMN, these models (Hu et al. (2019); Efrat et al. (2019); Dua et al. (2019); Ran et al. (2019)) do not have any provision to learn in absence of the additional supervision generated through exhaustive enumeration and manual inspection. (Gupta et al., 2020) have been the first to train a modular network strong, *albeit* a more fine-grained supervision for a fraction of training data, and auxiliary losses that allow them to learn from the QA pairs alone. Consequently on a carefully-chosen subset of DROP, NMN showcased better performance than NABERT and MTMSN, when strong supervision is available only for partial training data.

Our work takes it further along the direction in two ways

- while NMN baseline can handle only 6 specific kinds of reasoning, for which they tailored the program generation and gold reasoning annotation, our model works on the full DROP-*num*, that involves more diverse kinds of reasoning or more open-ended questions, and requires evaluating on a subset $\times 7.5$, larger by training on $\times 4.5$ larger training data.
- while NMN generalized poorly on the full DROP-*num*, especially when only one or more types of supervision is removed, our model performs significantly better without any of these types of supervision.

Together, NMN and GenBERT are some of the latest works in the two popular directions (reasoning and language model based) for DROP that allow learning with partial no strong supervision and hence act as primary baselines for our model.

Since in this work we are investigating how neural models can incorporate explicit reasoning, we focus on only answering questions having numerical answer (DROP-*num*), where we believe the effect of explicit reasoning is more directly observable. This is backed up by the category-wise performance comparison of reasoning-free language model GenBERT (reported in Geva et al. (2020)) with other hybrid models (MTMSN and NABERT+) that exploit numerical computation required in answering DROP questions. While, on DROP-*num*, there is an accuracy gap of 33% between the GenBERT model and the hybrid models (when all are trained on DROP only), there is only a 2-3% performance gap on the subset having answers as single span, despite the latter also needing reasoning. This evinces that the performance gap is indeed due to exploiting explicit reasoning under such strong supervised settings.

A.4.1 LIMITATIONS OF NMN

The primary motivation behind our work comes from some of the limitations of the contemporary neural module networks, NMN and the reasoning-free hybrid models MTMSN, NABERT+, NumNet, NAQANet; specifically their dependence on the availability of various kinds of strong supervision. For that we first describe the nature of programmatic decompositions of queries used in the modular architectures in the closest comparable work of NMN.

NMN defined a program structure with modules like ‘find’, ‘filter’, ‘relocate’, ‘find-num’, ‘find-date’, ‘year-difference’, ‘max-num’, ‘min-num’, ‘compose-number’ etc., to handle a carefully chosen subset of DROP showcasing only 6 types of reasoning, (i.e. *Date-Difference*, *Count*, *Extract Number*, *Number Compare*). For e.g. for the query *Which is the longest goal by Carpenter?* the program structure would be $(MAX(FILTER(FIND('Carpenter'), 'goal')))$, where each of these operations are learnable networks. However to facilitate learning of such specialized programs and the networks corresponding to these modules, the model needs precomputation of the exhaustive output space for different discrete operation and also various kinds of strong supervision signals pertaining to the program generation and execution.

Precomputation of the Exhaustive Output-Space: For operations that generate a new number as its output (e.g., *sum/diff*), the annotation enumerates the set of all possible outputs by computing over all subsets of number or date entities in the passage. This simplifies the task by allowing the model to directly learn to optimize the likelihood of the arithmetic expression that lead to the final answer, without any need for handling discrete operations.

Program Supervision provides supervision of the query category out of the 6 reasoning categories, on which their program induction grammar is tailored to. With this knowledge they can directly use the

category specific grammar to induce the program (for e.g. $SUM(FILTER(FIND))$ in Fig II). Further all these models (NMN, MTMSN, NABERT+, NumNet, NAQANet) use the supervision of the query category to understand whether the discrete operation is of type count or add/sub or max/min. which includes the knowledge of the ‘gold’ discrete operation (i.e. count or max/min or add/sub) to perform.

Query Attention Supervision provides information about the query segment to attend upon in each step of the program, as the program argument for e.g. in Fig I, ‘Carpenter’ and ‘goal’ in the 1st and 2nd step of the program.

Execution Supervision: For operations that select one or more of the number/date entities in the passage, (for e.g. max/min), rule based techniques provide supervision of the subset of numbers or dates entities from the passage, over which the operation is to be performed.

These annotations are heuristically generated through manual inspection and regular expression based pattern matching of queries, thus limiting their applicability to a small subset of DROP only. Furthermore, using a hand-crafted grammar to cater to the program generation for each of their reasoning categories, hinders their generalizability to more open ended settings. While this kind of annotation is feasible to get in DROP, this is clearly not the case with other futuristic datasets, with more open-ended forms of query, thus calling for the need for other paradigms of learning that do not require such manually intensive annotation effort.

A.4.2 PRETRAINING DATA FOR GENBERT

While GenBERT (Geva et al. (2020)) greatly benefits from pre-training on synthetic data, there are few notable aspects of how the synthetic textual data was carefully designed to be similar to DROP. The textual data was generated for the same two categories *nfl* and *history* as DROP with similar vocabulary and involving the same numerical operations over similar ranges of numbers (2-3 digit numbers for DROP and 2-4 digit numbers for synthetic textual data). The intentional overlap between these two datasets is evident from the t-SNE plots (in Figure 6) of the pretrained Sentence-Transformer embedding of questions from DROP-*num* (blue) and the Synthetic Textual Data (red). Further, while the generalizability of GenBERT was tested on add/sub operations from math word problems (MWP) datasets ADD-SUB, SOP, SEQ, their synthetic textual data was also generated using the same structure involving world state and entities and verb categories used by Hosseini et al. (2014) to generate these MWP datasets. Such bias limits mitigates the real challenges of generalizability, limiting the true test of robustness of such language models for numerical reasoning.

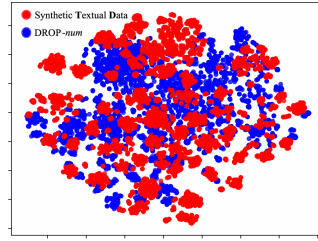


Figure 6: t-SNE of questions in DROP-*num*-Test and Synthetic Textual Data used in GenBERT models (TD and ND+TD)

A.5 QUERY PARSING: DETAILS

The Stanford Dependency parse tree of the query is organized into a program structure as follows

- **Step 1)** A node is constructed out of the subtrees rooted at each immediate child of the root, the left-most node is called the root-clause
- **Step 2)** Traversing the nodes from left to right, an edge is added between the left-most to every other node, and each of these are added as steps of the program with the node as the query span argument of that step and the reference argument as the incoming edges from past program steps
- **Step 3)** The terminal (leaf) nodes obtained in this manner are then further used to add a final step of the program which is responsible for handling the discrete operation. The query-span argument of this step is the root-clause, which often is indicative of the kind of discrete reasoning to perform. The reference arguments of this step are the leaf nodes obtained from Step 2).

Figure 7 provides some example queries similar to those in DROP along with their Dependency Parse Tree and the Simplified Representation obtained by constructing the nodes and edges as in Step 1) and 2) above, and the final program which is used by WNSMN. Note that in this simplified

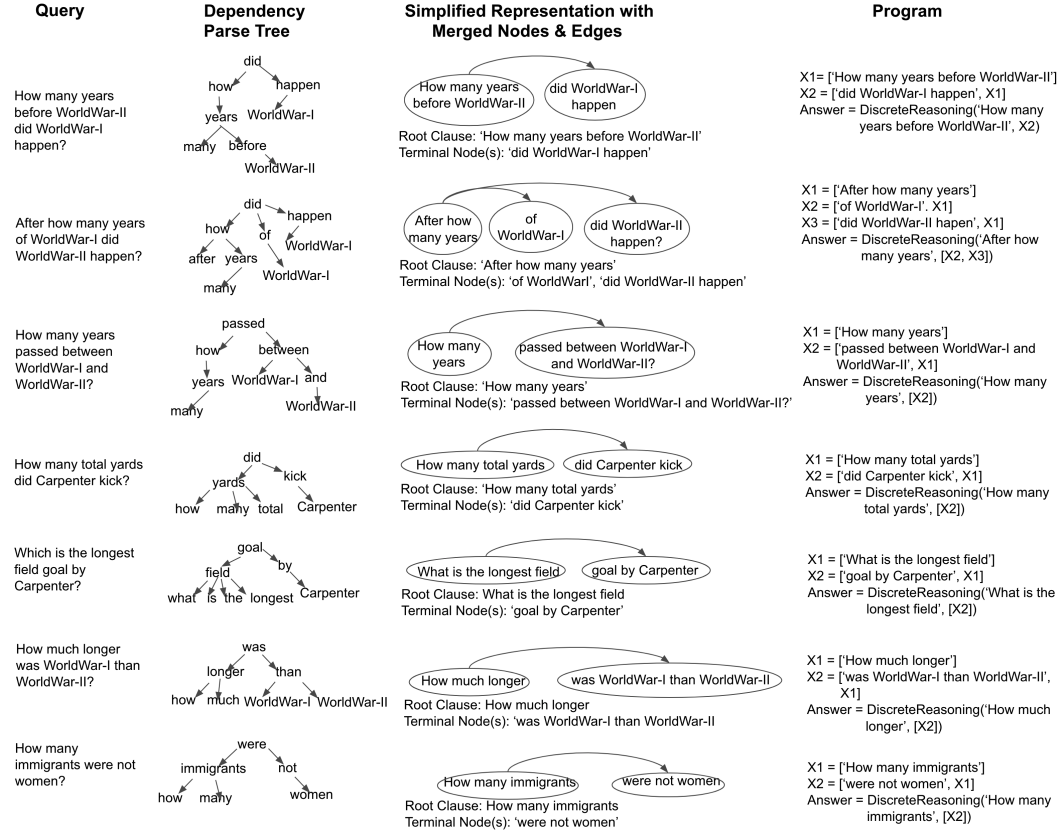


Figure 7: Examples of Programs for WNSMN obtained from the Dependency Parse Tree of the Query

representation of the parse tree the root-word of the original parse tree is absorbed in its immediate succeeding child. Also we simplify the structure in order to limit the number of reference arguments in any step of the program to 2, which in turn requires the number of terminal nodes (after step 2 of the above process) to be limited to 2. This is done in our left to right traversal by collapsing any additional terminal node into a single node.

A.6 RL FRAMEWORK: DETAILS

In this section we discuss some additional details of the RL framework and tricks applied in the objective function

Iterative ML Objective: In absence of supervision of the true discrete action that leads to the correct answer, this iterative procedure fixes the policy parameters to search for the *good* actions (where $\mathcal{A}^{good} = \{a : R(x, a) = 1\}$) and then optimizes the likelihood of the *best* one out of them. However, the simple, conservative approach of defining the best action as the most likely one according to the current policy can lead to local minima and overfitting issues, especially in our particularly sparse and confounding reward setting. So we take a convex combination of a conservative and a non-conservative selection that respectively pick the most and least likely action according to the current policy out of \mathcal{A}^{good} as best. Hyperparameter λ weighs these two parts of the objective and is chosen to be quite low ($1e^{-3}$), to serve the purpose of an epsilon-greedy exploration strategy without diverging significantly from the current policy.

$$J^{IML}(\theta, \phi) = \sum_x (1 - \lambda) \max_{a \in \mathcal{A}^{good}} \log P_{\theta, \phi}(a|x) + \lambda \min_{a \in \mathcal{A}^{good}} \log P_{\theta, \phi}(a|x)$$

Using Noisy Pseudo-Reward: In addition to using the REINFORCE objective to maximise the likelihood of actions that lead to the correct answer, we can also obtain different noisy *pseudo rewards* ($\in \{-1, +1\}$) for the different modules that contribute towards the action sampling (i.e.

the operator and the entity-type and different argument sampler networks). Towards this end, we define pseudo-reward for sampling an operator as the maximum of the reward obtained from *all* the actions involving that operator. Similarly, we can also define reward for predicting the entity-type (date or number) over which the discrete operation should be executed. Following the same idea, we also obtain pseudo rewards for the different argument sampling modules. For e.g. if the most likely operator (as selected by the Operator Sampler) is of type `count` and it gets a pseudo-reward of $+1$, then, in that case, we can use the reward obtained by the different possible outputs of the Counter network as a noisy pseudo-label supervision and subsequently add an explicit loss of negative log-likelihood to the final objective for the Counter module. Similar pseudo-reward can be designed for the Entity-Ranker module when the most likely operator sampled by the Operator Sampler needs arbitrary number of arguments. Treating the pseudo-reward as a noisy label can lead to a negative-log-likelihood based loss on output distribution from the Entity-Ranker, following the idea that the correct entities should atleast be ranked high so as to get selected when sampling any arbitrary number of entities.