NT5?! Training T5 to Perform Numerical Reasoning

Peng-Jian Yang[a*], **Ying Ting Chen**[a*], **Yuechan Chen**[a], **Daniel Cer**[a, b]
{lesterpjy, chentim, sonyachan, dcer}@berkeley.edu

[a]University of California
Berkeley, CA

[b]Google Research
Mountain View, CA

Abstract

Numerical reasoning over text (NRoT) presents unique challenges that are different from the conventional task of reading comprehension (RC) in NLP. While T5 is known for its substantial amount of pre-training on RC, its ability on NRoT remains untested. In this paper, NRoT skills are injected into T5 through a series of experiments. We first perform a series of preliminary experiments prior to executing our training that consists of five schedules. Preliminary experiments test hypotheses including model scales, question type hinting, data shuffling methods, digit-by-digit tokenization, and learning rate schedules. The final model that results from the five experimental training schedules is a T5-Small pre-trained on three datasets designed to build NRoT and RC skills into T5, before fine-tuning on DROP CAT and our gold dataset, DROP. We show that our training improves DROP's adjusted F1 performance (a numeracy-focused score) from 45.90 to 70.83. Our model outperforms the best model proposed in the original DROP paper (47.01), and closes in on GenBERT (72.4), a custom BERT-Base model with significantly more parameters trained on the same two synthetic datasets we use to inject NRoT.

1 Introduction

Numerical reasoning over text (NRoT) in NLP, which differs from extractive tasks, is unique in that answers require mathematical skills to solve. An example from the Discrete Reasoning Over Paragraphs (DROP) dataset(Dua et al., 2019) in Table 1 shows that the answer, 86.9%, can only be derived with subtraction, and cannot be found anywhere in the context or question.

Symbolic and distributed approaches are commonly used to achieve NRoT. A symbolic model "translates" text to the mathematical expressions required to solve NRoT questions. The current state-of-the-art performance on DROP is achieved by a symbolic model, QDGAT-ALBERT (Chen et al., 2020). A distributed model, on the other hand, directly generates the final numerical answers by "internalizing" the mathematical equations and steps for derivation, as illustrated by Table 1. A pro and con analysis between the two approaches is beyond the scope of this research but can be found in *Injecting Numerical Reasoning Skills into Language Models* (Geva et al., 2020). In this work, we focus solely on the distributed approach. We show evidence that the Text-to-Text Transfer Transformer (T5) (Raffel et al., 2019), even at its smallest scale, is able to internalize NRoT through the distributed approach.

Question: How many percent of the population is not Irish?
Context: As of the census of 2000, there were 218,590 people, 79,667 households, and 60,387 families residing in the county.[...] 1.91% of the population were Race (United States Census) or Race (United States Census) of any race. 22.5% were of German people, 13.1% Irish people, 9.8% Italian people, 9.2% English, 8.1% "American" and 6.0% Polish ancestry.
Symbolic Model: 100% - 13.1%
Distributed Model: 86.9%

Table 1: Illustrated difference between symbolic and distributed models. In this work, we train T5 using the distributed approach.

Here we briefly describe the structure of this paper. In Section 2, we summarize some works related to NRoT in general. Section 3 begins by describing datasets used for experimentations, including NUM and TXT, the synthetic datasets. Section 4 describes the standard metrics for evaluating performance in DROP. We then go over the experiments designed to test fundamental hypotheses and identify hyperparameters in Section 5. The methodologies we applied for exploring the efficacies of numeracy injection are described in Section 6. These results are then discussed and analyzed in Section 7 and Section 8 §8 Error Analysis. Finally, in Section 9 we summarize our findings and potential improvements.

2 Related Works

Discrete Reasoning Over Paragraphs (DROP), introduced by AllenNLP in 2019(Dua et al., 2019), is

[*] Equal contribution

a crowdsourced, adversarially-created 96k question benchmark on multi-span extraction and discrete reasoning that requires the specific task of NRoT to perform well on. In the paper, a numerically-aware QANet model (NAQANet) which generates arithmetic expressions with a limited set of operations is proposed as a baseline. Introduced by (Geva et al., 2020), GenBERT is a BERT-Base model customized with specialized heads for handling discrete reasoning and NRoT. GenBERT's use of synthetic numerical and textual data inspires our work on training T5 without architectural changes. The current state-of-the-art model, QDGAT-ALBERT, uses a directed graph attention network between a RoBERTa based representation extractor and a prediction module for discrete reasoning (Chen et al., 2020). This novel approach has the benefit of isolating the discrete reasoning required for solving numerical questions, but increases the overall model size. A tangential line of work exists for analyzing the mathematical reasoning ability of models over text (Wallace et al., 2019; Ravichander et al., 2019), and on arithmetic problems (Saxton et al., 2019; Lample and Charton, 2019).

3 Datasets

Figure 1 summarizes the six datasets we explore. We choose DROP as our gold data specifically for its unique coverage on NRoT. DROP consists of four types of questions, which can be answered using the context provided. Approximately 61% of the examples in DROP are "number" questions that require mathematical skills to solve. The other three types are "single-span" (32%; answer is one single continuous span in the context), "spans" (6%; answer consists of multiple discrete spans in the context), and "date" (2%; answer can also be found in the context but with its format being a day, month, year, or combination). Note that all four question types in DROP can require NRoT skills. For example, "What date occurred first: The Siege of Nice or Polin succeeding Antoine de Rincon?" (date) requires the skill of ordering, and "What races had more than 700 births?" (spans) requires the skill of comparison. Numerical questions, however, are unique for two characteristics: All numerical questions require NRoT skills (which might or might not be required for date/span/spans), and the answers of which (consisting of only digits) do not show up in the context or question, as demonstrated by Table 1 (date/span/spans, by contrast, always have their answers in the context). DROP CAT is the same as DROP, but with labels replaced by the four question types. The use and performance of DROP CAT are detailed in Section 6.

NUM consists of near 1M synthetically generated questions on the seven types of numerical skills required to solve numerical questions in DROP. TXT, building on NUM, includes 2M plus examples generated based on the seven numerical skills in NUM. All examples in TXT mimic the DROP format. The construction of the two datasets are detailed in the Appendix A.

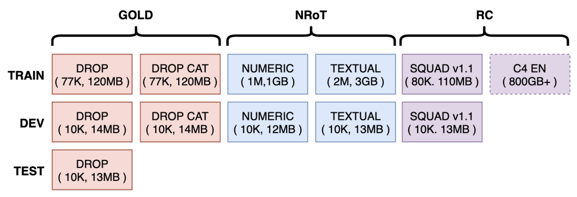

Figure 1: Datasets used in our work to inject T5 with NRoT skills.

Lastly, we include SQuAD v1.1 as an attempt to strengthen RC, which we find lagging in performance at the end of NRoT training. Our ideal plan is to multitask-train on C4 EN but depart from the idea given the resource constraints described in the Appendix A.

4 Performance Evaluation

In addition to NRoT questions, we chose DROP for its unique emphasis on evaluating numeric answers. DROP employs two official metrics: a numeracy-focused, macro-averaged F1, and Exact-Match (EM).

F1 is set to 0 whenever there is a number mismatch between gold and predicted in numerical and single-span questions, regardless of other word overlaps. In the case of spans, a single F1 score is computed using macro-average after each individual span within the gold answer is evaluated independently. EM in DROP, on the other hand, simply implements SQuAD's EM and normalizes predictions prior to performing evaluation (e.g. whitespace removal, lowercase transformation, etc.). Lastly, each example in DROP could include more than one validated answer. In the presence of multiple gold answers, both EM and F1 will take a max over all computed scores.

Note that the only way to evaluate DROP's test set is by submitting predictions to the leaderboard. F1 and EM are returned with no additional information; hence we are unable to perform error analysis or evaluate performance by question type like we do for the development set.

5 Preliminary Experiments

Preliminary experiments prior to training are designed to test our hypotheses. For details on how we have tuned our hyperparameters and learning rates, refer to Appendix A.

Baseline Model (T5-Small) We begin by building a barebone baseline model using T5-Small trained directly on DROP for 60 epochs with a constant learning rate of 0.001. We shuffle the training examples, set the batch size to 256 (largest possible size for training on Colab Pro's TPUs without causing memory leaks), and save our best model on minimal loss using DROP's validation set. T5's transfer learning is proven to be strong: Our baseline achieves a F1 score of 44.46 comparable to the 47.01 of NAQANet (a custom, numerically-aware model based on QANet proposed in the DROP paper (Dua et al., 2019)).

Scaled Baseline (T5-Base) T5 offers five model scales: small, base, large, 3b, and 11b. To test our hypothesis that model scale is effective in improving NRoT performance, we train a T5-Base using exactly the same hyperparameters as our baseline model. The "barebone" T5-Base predicts extremely well on DROP, outperforming all models including NAQANet proposed in the original DROP paper. T5-Base also sees a strong F1 score boost on numerical questions over our baseline (13.09 points, as shown in Table 2). Note that unless specified otherwise, all works in this paper are done using T5-Small.

Question-Type Hinting We hypothesize that T5 would predict better if the model is aware of the four question types in DROP. In particular, numeric questions would benefit because they require specific skills that are significantly different from RC. To test our hypothesis, we train T5 to recognize DROP's question types using DROP CAT, a custom DROP dataset with gold answers replaced by the four question types: n (number), d (date), ss (single-span), and s (spans). We multitask-train our model on DROP and DROP CAT for 100 epochs using our baseline's hyperparameters. The results are encouraging: Numeric and spans questions improve with relatively low deterioration on other question types (Table 2).

Encouraged by the results of DROP CAT, we further hypothesize that numeric questions's performance could improve even more if T5 is allowed to focus on just classifying whether an answer requires NRoT skills. To test our hypothesis, we create a simplified Drop CAT by changing the answers from the four question types to "math" vs. "no_math," and retrain our model using the same hyperparameters. However, the new model underperforms across all question types, and our fine-tuning schedules will hint on all four question types based on the observation.

Digit-by-Digit Tokenization Research shows that numerical computation benefits from the direct use of digits(Wallace et al., 2019) although T5's SentencePiece tokenizer does not apply special treatments on numbers. We thereby hypothesize that tokenizing numbers digit by digit (e.g. 100 $\rightarrow$ ['1', '0', '0']) would help T5 internalize the math skills required for NRoT. To test our hypothesis, we customize SentencePiece by adding the 10 numerical digits (0 to 9) as special tokens. We then rerun our baseline model using the new digit-by-digit tokenizer. We show in Table 2 that performances on number and spans improve although there is a slight drop on span. The unexpected deteriation on date is concerning (discussed in Section 8), but given our focus on NRoT, we implement digit-by-digit tokenization across all models in training.

Shuffle by Context versus Example We hypothesize that shuffling by contexts instead of examples would improve performance because of each of the contexts in DROP consists of multiple questions. T5 is able to learn closely related questions in batches if we group questions under the same context during shuffling. To test the hypothesis, we load examples sequentially from DROP's raw JSON, partition them into batches of size 256, and shuffle only by batch without altering the order of examples within. While the setup does not guarantee all questions under the same context to be partitioned into the same batch, we reason that an experiment as such is a good starting-point approximation and would execute a more precise context-based shuffling if the performance improves. As shown by Table 2, we observe a significant drop on T5's capability to perform numeric reasoning while RC remains (RC-focused) remain relatively unchanged. Despite the large improvement on date (our focus is on NRoT), we decide to move forward with the traditional wisdom and shuffle by examples in all training schedules.

Temperature-Scaled Mixing Our training schedules take advantage of T5's multitasking and mix multiple datasets using temperature-scaled ratios (detailed in Appendix A). After experimenting with different temperatures, we decide to simply set temperature to 1 across all training schedules except for E5. As evidenced by our training results (Table 4), the decision is mainly based on the fact that we observe no overfitting to the larger datasets, NUM (1M examples) and TXT (2M examples) at the end of NRoT pre-training, likely due to the fact that NUM and TXT are two synthetic datasets generated by closely following the specific formats and NoRT skills found in DROP. SQuAD v1.1 and DROP CAT, on the other hand, come with comparable sizes and structures as DROP by default so overfitting due to size disparity is less likely to be an issue. In sum, task interference or negative transfer, as described in the original T5 paper, is not an issue in our training, and setting temperature to 1 allows us to easily track down how many times T5 goes over an entire dataset. Note: The temperature of E5 (our attempt to multitask-train on all datasets at the same time), 10, is determined by the computing resources we have available on Colab Pro. Each epoch in E5 completes a full run of all examples in DROP, and we try to set a temperature as low as possible to include as many examples as possible from the other datasets.

6 Training Schedules

We now describe our training which consists of two stages: pre-training on NRoT/RC, and fine-tuning on DROP/DROP CAT. Building on the preliminary experiments, we complete a total of five schedules to test multiple methodologies and hypotheses (E1 to E5, as summarized by Figure 3). We will first describe the high-level specifications and rationales of each experi-

Model	overall	number	date	span	spans
T5-Small (Baseline)	44.64	31.83	53.28	67.42	55.44
T5-Base	55.69	44.92	52.52	75.12	66.19
DROP + DROP CAT	49.55	39.51	50.68	67.15	60.76
DROP + DROP CAT S	47.55	36.95	47.62	66.12	59.83
with Digit Tokenization	47.08	35.78	50.15	66.65	60.63
Shuffle by Batch	38.71	23.02	65.86	65.86	54.99

Table 2: 1. In the F1 performance of T5-Small and T5-Base, except for date, performance improves across all question types with an increased model scale. 2. F1 performance improves with the help of question-type hinting using DROP CAT (baseline is without multitasking on DROP CAT). The simplified DROP CAT, however, does not produce the same benefits. 3. F1 on numeric answers improves with digit-by-digit tokenization (baseline is without digit-by-digit tokenization). The drop of performance on date is concerning and is discussed in Section 8. 4. F1 performance deteriorates when shuffling is done by batches instead of examples (baseline shuffle by examples).

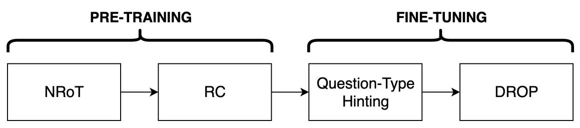

Figure 2: High-level overview of our training schedules. The full schedule is detailed in Figure 3.

ment, followed by discussions on hyperparameters and sequential/continuous training.

E1 and E2 E1/ E2 are designed to test the performance of saving best models by different validation data. E1 is validated on the DROP dev set in pre-training while E2 is validated on the dev sets of NUM and TXT. We hypothesize that E1 would perform better but are surprised by the results.

E3 and E4 E3/ E4 attempt to strengthen T5's RC by building on top of E2's training on NRoT. We observe only minor improvement on T5's RC at the end of pre-training on NRoT in E2: F1 on numeric questions increases from 31.83 to 63.18 while F1 on span barely improves from 67.42 to 68.56. E3 aims to fix the issue with the addition of SQuAD v1.1 in pre-training. In E4, we further test if performance would benefit by multitask training SQuAD v1.1 together with DROP and DROP CAT in the fine-tuning stage.

E5 E5 is our attempt to take full advantage of T5's multitasking. Based on T5's original paper, we hypothesize that multitasking would be the best way to achieve optimal performance. Unfortunately, we are only able to partially test our hypothesis due to the limited computing resources (documented in Appendix A). For example, each epoch in E5 covers all examples in DROP but only a small portion of TXT. Even with 80 epochs, our training only manages to learn over TXT roughly 3 times, a far cry from the 30 epochs in E1 to E4.

Sequential and Continuous Training All five training experiments involve sequential training. In addition, due to the strict policy of Colab Pro (i.e. up to 24 hours of training), we are unable to finish all 30 epochs of training on Textual in E1/E2/E3 and 80 epochs of multitask training in E5 in one run. As a result, we break our training into continuous training runs. For example, in E3 Phase 2, we complete the first 15 epochs of training on TXT (16 hours), and complete the next 15 epochs by starting with the model at the end of the 15th epoch. The learning rate decay schedule is manually adjusted so that it continues to decay from the point at the end of the half of training. The best model out of all 30 epochs (by the minimum loss on validation data) then becomes the starting point for the training in the next phase. The flow described above is summarized in Figure 4.

7 Results

Comparing experiment performance: The overall results of our five training schedules, E1 to E5, are summarized in Table 3, and decomposed in Table 4.

E1 versus E2 E1 and E2 differ in the validation data used in pre-training. E1 saves its best model by DROP's development set while E2 by Numeric and Textual's. A surprising finding is that E2 outperforms E1 by a large margin at the end of Phase 1 (i.e. end of training on NUM). In particular, E2 achieves a F1 score on numeric questions (41.37) that is nearly 5 points higher than that of E1 (36.97) without sacrificing performance on other question types. The observation is puzzling: Why does validating on DROP lead to a lower performance on DROP?

We reason that the gap is caused by the difference between the loss on dev (used to determine the best model during training) and DROP's actual metrics (used to evaluate the best model's performance on dev at the end of training). As detailed in Section 4, DROP's evaluation metrics are unique in that there is a strong emphasis on numeracy: an example can receive a zero F1 score if the number does not match. As a result, a best model determined on the minimum loss of dev during training does not necessarily translate to the best performance on DROP's numeracy-focused metrics.

E1's performance, however, does catch up with E2 at the end of pre-training (i.e. end of Phase 2). This is likely due to the "synergy" between NUM and TXT

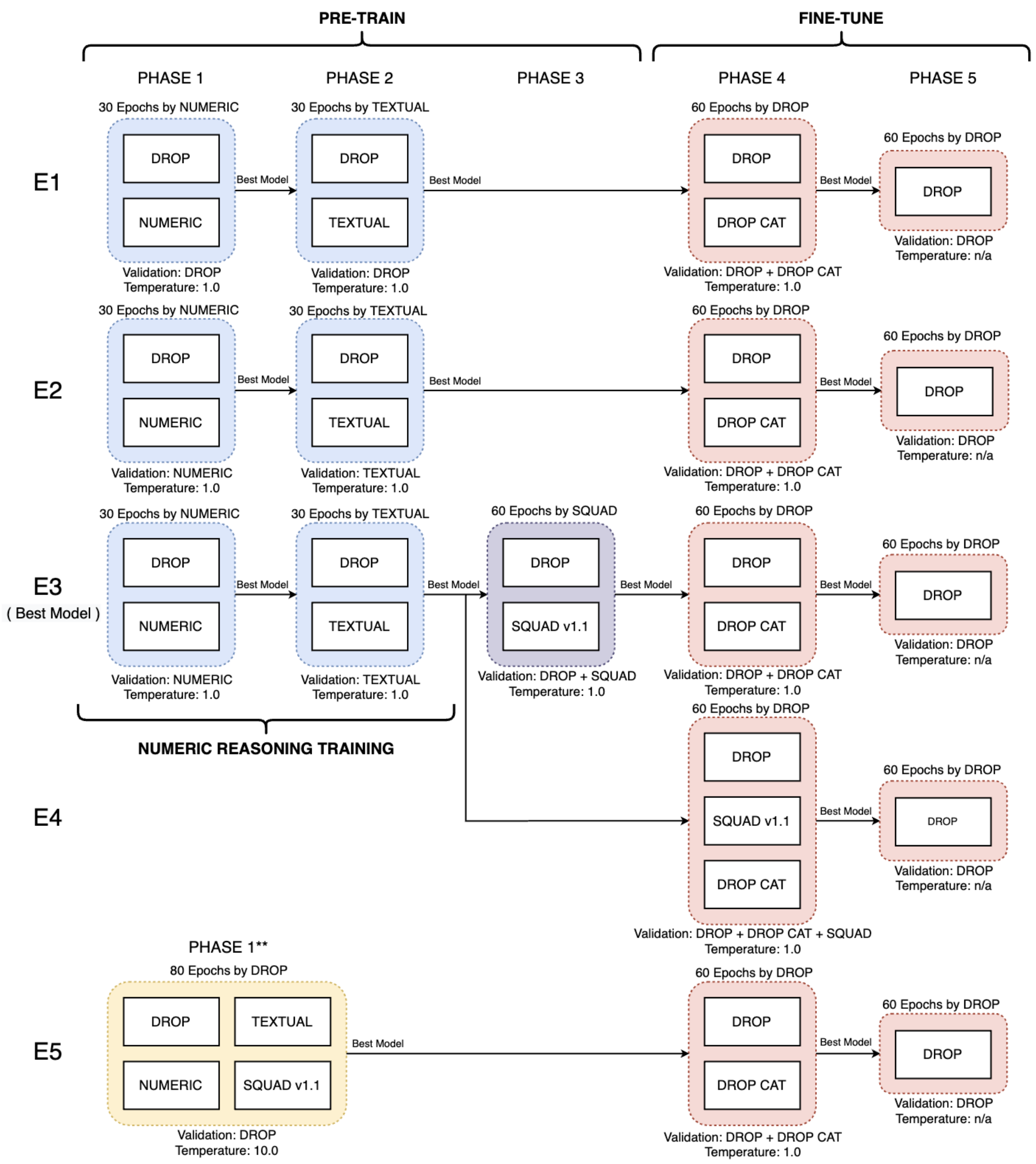

Figure 3: Summary for our five training schedules. We define pre-training as training on NRoT and RC. Fine-tuning, on the other hand, focuses mainly on learning DROP. E1 to E3 are pre-trained sequentially on NUM, TXT, and SQuAD before fine-tuning on DROP and DROP CAT. E4 moves the RC training from pre-training to fine-tuning by combining SQuAD, DROP CAT, and DROP with the help of T5's multitasking. E5 is our attempt to multitask-train on all datasets prior to fine-tuning.

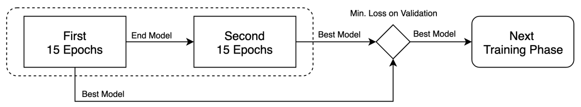

Figure 4: Illustration on how sequential and continuous training are done across multiple datasets.

	Development		Test	
Model	EM	F_1	EM	F_1
Baseline	41.12	44.64	41.97	45.90
E1	65.00	68.53	-	-
E2	66.04	69.60	-	-
E3	**66.87**	**70.31**	**67.00**	**70.83**
E4	66.41	69.80	-	-
E5	63.10	66.47	-	-
NAQANet	46.20	49.24	44.07	47.01
GenBert	68.8	72.3	68.6	72.4
QDGAT	-	-	87.04	90.10

Table 3: Performance summary for our baseline, training experiments (E1 to E5; our best model is in), and select benchmarks. NAQANet is the highest-performing model proposed in DROP's original paper. GenBERT is a modified BERT-base model fine-tuned on NUM and TXT. QDGAT is the current state of the art on DROP's leaderboard.

that closes the gap described above. The construction of the two synthetic datasets, NUM and TXT, and reason behind the "sunergy" is detailed in Appendix A. Regardless, we hypothesize that our model should benefit from validating directly on DROP's metrics during training with at least some degree of reduced performance variance. Unfortunately, we are unable to test the hypothesis because of the computing resources required. Evaluating on DROP's metrics requires the conversion of SentencePiece tokens back to words, resulting in a validation that takes more than one hour at the end of each epoch; 30 epochs would take 30 plus hours to run.

E3 (Best Model) versus E4 E3/E4 attempt to resolve the issue of weak RC performance at the end of the pre-training of E1/E2. Models from E1 and E2 beat our baseline by a large margin, showing strong evidence that T5 is able to learn NRoT using the two synthetic datasets, NUM and TXT. However, while E1 and E2 achieve significant improvement on numerical questions, we observe only slight improvement on other question types (which involve extractive tasks) at the end of pre-training on NRoT. We reason that the low improvement is caused by the mismatch of the languages used to phrase questions and contexts between DROP and TXT. As such, E3 is designed to close the gap by improving T5's extractive RC capability with the inclusion of SQuAD v1.1.

SQuAD v1.1 is a Q&A dataset with a format similar to DROP (i.e context-question pairs). While not an ideal dataset to train T5's RC (our original plan to multitask-train on C4 or Wikipedia is not doable due to the shortage of computing resources), E3 does see improvement across all question types at the end of the training on SQuAD v1.1. The performance of E3 at the end of training on SQuAD (Phase 3), for example, sees an average improvement of 3.06 points in F1 on extractive tasks (date, span, and spans) over E2 at the end of training on TXT (Phase 2).

With the success of E3, we further test the hypothesis that by multitask-training SQuAD together with DROP CAT and DROP in Phase 4, T5's learning on extractive tasks will be "closer" to what's required to comprehend DROP. However, we find only weak evidence to validate our hypothesis (a mere F1 improvement of 0.25 on span at the end of Phase 4), and the improvement comes at the expense of minor deteriorated performance on numeric questions. E3 eventually becomes our best model although its performance difference from E4 can be seen as neglectable.

E5 E5 tests the hypothesis that multitask-training is an optimal way for T5 to acquire NRoT skills. The experiment is an approximation due to our limited computing resources as discussed in Section 6. Nevertheless, when compared to our best model (E3), the fine-tuned performance in E5 underperforms on numeric questions by a large margin (nearly 6-point difference in F1). We argue that this is not a strong indication that a single-phase multitasking on all datasets is unsuitable for NRoT training. Note that the performance of E5 on other types of questions, which also require some degree of numerical reasoning (as explained in Section 3), are on par with the results produced by our best model (E3). E5's performance on date, in particular, even slightly outperforms our best model. In addition, each epoch in E5's pre-training only covers a small portion of NUM and TXT, and with a temperature of 10, 80 epochs only translate to only five and three full runs of NUM and TXT respectively (1M and 2M examples respectively). We believe that with enough computing resources and settings on mixture temperature and validation data, multitasking could potentially produce results on par with or even exceeding our other sequential models. Unfortunately, this is all speculation as we are unable to perform a more precise experiment to test our hypothesis.

On the other hand, E5 reveals a major disadvantage of multitasking. Compared to E3, E5 takes roughly the same amount of training time (50 hours) to complete, but is much more difficult to run and experiment with (e.g. you can't observe "work-in-progress" results as in a sequentially trained model with different phases). In the face of computing resource shortage and research time constraint, we thereby advocate for the sequential approach we have developed in our best model.

Best Model (E3) versus GenBERT and Baseline E3 is our best model with an overall performance of 66.8 EM and 70.3 F1 on the dev set, and similarly a

67.0 EM and 70.8 F1 on the test data. This puts our model on the 21st place on the DROP leaderboard, right behind GenBERT at the 20th place. Notably, our model is based on the smallest scale of T5 with significantly fewer parameters than GenBERT, and achieves the performance without modifying T5's underlying architecture. The encoder-decoder T5-small model has just 60 million parameters, compared to the 110 million parameters of GenBERT in its encoder alone (a modified BERT-BASE). When comparing E3 with our baseline, we show strong evidence that our training schedule improves T5's ability to perform both NRoT, a relatively weaker aspect of T5's pre-training from C4, and RC on DROP. Overall, Table 4 shows that E3 is able to significantly boost the performance on numeric questions, an increase of F1 from 31.83 to 70.39, while slightly maintaining or improving on other types of questions in DROP. Overall, E3 achieves a 59.62% increase on EM and 54.33% on F1 over the baseline on DROP's test data. In addition, our preliminary experiment on model scale shows evidence that the performance could be further improved by scaling up T5.

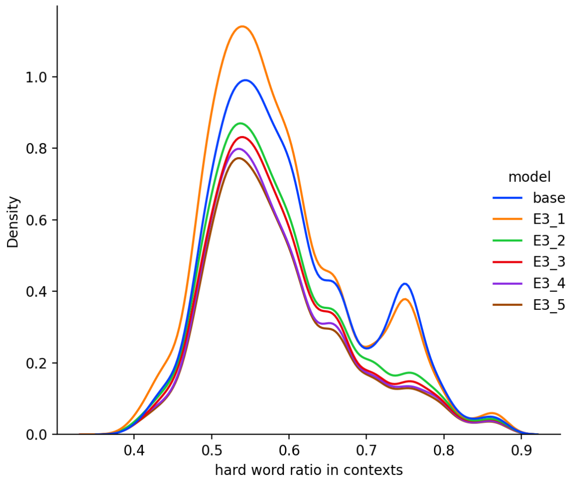

Figure 5: Hard word ratio KDE plot for contexts of examples in the DROP dev set where EM does not equal to 1 for each phase of E3, and baseline. Hard word is defined as the words not in the Dale-Chall easy word list.

8 Error Analysis

To better understand the achievement and limitations of our best model, the errors of E5 Phase 5 on the development set are analyzed. In 38 of 100 errors sampled from number questions, E5 Phase 5 have at least one partial digit match. Of the total 86 errors on date questions, 39 questions require numerical calculation, as the answers do not exist in the context (non-extraction task). And 9 of the remaining 47 questions with answers in the context, the model wrongly performs numerical calculations, instead of simply extracting the correct answer. With a sample of 100 span and spans errors, 49 of the questions contain either reasoning skills not covered in the pre-training phases, or higher level reasoning skills that require several degrees of inference. This is compared to the 43% shown by Dua et al. (2019), although our criteria vary due to the manual evaluation nature. With the data driven framework we have proposed, many of these errors can be addressed with pre-training datasets that cover more complicated calculations and reasoning skills.

Incorrect predictions (EM=0) made by the baseline model for four randomly sampled examples (one from each question type), which E3 Phase 5 answered correctly are summarized in Appendix A Table 5. These errors in prediction made by baseline reconfirms the requirement of both discrete reasoning and RC in scoring well on DROP. While it is time consuming and inconsistent to manually evaluate mistakes made in a sample, which seems to be the standard practice on DROP, we can instead evaluate the RC portion with standard metrics to get a sense of how the models are learning. Figure 5, shows that by training T5-small on a combination of digit-tokenized DROP and NUM, in E1 Phase 1, there is an immediate negative impact. As previously discussed in Section 7, negative impacts at this stage are expected. By phase 2, TXT is able to dramatically improve the performance over the baseline. However, the Phase 2 curve is suddenly missing the peak between 0.7 and 0.8. In this range, just 44 contexts contribute to the peak difference between the questions that Phase 1 answered wrongly, and Phase 2 correctly. Of these contexts, the 30 most common non-stopwords words that are also not in the easy word list are all related to the census, such as ”race”, ”housing”, ”population”, etc. This suggests that for this portion of the contexts that are relatively harder, TXT was able help overcome a ”barrier of understanding”. We observe that models continue to improve incrementally after Phase 2 in all experiments. Further analysis with tools such as parse trees will allow us to analyze more deeply this barrier that differentiates the questions.

9 Conclusion

T5, a model known for its numerous pre-training on C4, remains untested for its ability on discrete reasoning. In this work, we propose a framework to train T5 to perform NRoT using the distributed approach. We choose DROP as our gold dataset for its unique emphasis on numeric examples and evaluation metrics. Our preliminary experiments show evidence that digit-by-digit tokenization, question-type hinting, and model scaling are effective in training T5 on DROP. Our best model (pre-trained on NUM/TXT/SQuAD v1.1 and fine-tuned on DROP/DROP CAT) demonstrates strong improvement on performance on NRoT without compromising T5's strong RC ability. The current state of the art, QDGAT (F1: 90.10) and human performance on DROP (96.42), however, are significantly higher than our achieved F1 score at 70.31 on the test set. In addition, our experi-

		Number		Date		Span		Spans		**Overall**	
Model	Phase	EM	F1	EM	F1	EM	F1	EM	F1	EM	F1
Baseline	-	31.79	31.83	43.95	53.28	62.09	67.42	26.98	55.44	41.12	44.64
E1	1	36.97	36.99	43.95	51.63	59.45	64.67	27.69	55.79	43.52	46.95
E1	2	63.25	63.27	42.04	51.42	63.03	68.29	29.63	56.97	60.83	64.26
E1	4	67.03	67.07	42.68	51.43	65.19	70.80	31.57	58.89	63.95	67.49
E1	5	68.72	68.78	43.31	50.36	65.09	70.62	32.10	60.05	65.00	68.53
E2	1	41.37	41.38	40.76	48.94	61.31	66.45	29.81	58.73	46.86	50.32
E2	2	63.16	63.18	44.59	52.76	63.23	68.56	29.81	58.85	60.90	64.42
E2	4	67.73	67.75	45.86	53.80	65.16	70.61	33.69	61.18	64.54	68.02
E2	5	69.78	69.83	42.68	51.23	66.00	71.47	34.22	62.53	66.04	69.60
E3	$3^{\star}$	65.61	65.68	45.22	55.38	65.94	71.23	34.39	62.75	63.52	67.06
E3	4	68.65	68.69	45.22	53.87	66.54	72.01	36.68	63.47	65.71	69.17
E3	5	**70.34**	**70.39**	45.22	53.85	66.75	72.35	**37.74**	63.43	**66.87**	**70.31**
E4	$4^{\star\diamond}$	65.90	65.94	45.22	54.31	66.48	71.73	35.80	62.55	63.95	67.35
E4	5	69.44	69.47	45.22	53.17	**67.45**	**72.60**	35.63	63.08	66.41	69.80
E5	$1^{\dagger}$	56.84	56.86	42.68	50.44	64.58	69.72	33.51	61.78	57.62	61.04
E5	4	63.73	63.81	**49.04**	**56.28**	65.97	71.24	36.16	63.65	62.54	65.99
E5	5	64.43	64.49	45.86	52.99	66.48	71.61	36.51	**63.80**	63.10	66.47

Table 4: Decomposed and overall EM and F_1 scores on different answer types in the development set of DROP for each phase and experiment. High scores for each type are in bold. $\star$Notice that E3 and E4 begin training using the weights learned in E2 phase 2. $\diamond$E4 trains with SQuAD in addition to DROP and DROP CAT. $\dagger$E5 does not undergo early phase training as mentioned earlier.

ments and training schedules are limited by the computing resources available. As such, we are optimistic that our proposed training framework is subject to room for significant performance improvement.

Acknowledgments

We thank Dr. Daniel Cer, for his generosity in both his precious time, and the valuable knowledge he shared, for which this project greatly benefited from. We also thank the W266 instructors team for their hard work answering questions on Piazza, and making the course possible.

References

Kunlong Chen, Weidi Xu, Xingyi Cheng, Zou Xiaochuan, Yuyu Zhang, Le Song, Taifeng Wang, Yuan Qi, and Wei Chu. 2020. Injecting numerical reasoning skills into language models. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*.

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 *North American Association for Computational Linguistics*.

Mor Geva, Ankit Gupta, and Jonathan Berant. 2020. Injecting numerical reasoning skills into language models. In *Association for Computational Linguistics*, volume arXiv:2004.04487.

Mohammad Javad Hosseini, Hannaneh Hajishirzi, Oren Etzioni, and Nate Kushman. 2014. Learning to solve arithmetic word problems with verb categorization. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*.

Guillaume Lample and François Charton. 2019. Deep learning for symbolic mathematics. In *International Conference on Learning Representations*.

Telmo Pires, Eva Schlinger, and Dan Garrette. 2019. How multilingual is multilingual bert? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2019. Exploring the limits of transfer learning with a unified text-to-text transformer. volume arXiv:1910.10683.

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*.

David Saxton, Edward Grefenstette, Felix Hill, and Pushmeet Kohli. 2019. Analysing mathematical reasoning abilities of neural models. In *International Conference on Learning Representations*.

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*.

A Appendices

Hyperparameters Five input prefixes (context, answer_me, calculate, and classify_me) are used for labeling the datasets during training. An example input from DROP is: "answer_me: How many more men did Ming Rui have than cavalry? context: In March 1768, Ming Rui began his retreat, pursued by a Burmese army of 10,000 men and 2000 cavalry. . . " Note that we place the question before context in order to avoid the cutoff when the input exceeds our encoder's max length. Our preset encoder and decoder max lengths (512, T5's maximum allowance, and 54 respectively) cover 100% of context and gold answers in all datasets except for DROP (of which 3.89% of contexts are cut off in training).

All experiments are performed using T5-Small. Except for E5, the mixture temperature is set to 1 and training steps are calculated accordingly such that one epoch covers all examples in each of the datasets used for training exactly once. E5 (a model simultaneously multitask-trained on four different datasets) is the only exception because of the limited computing resources we have. The temperature of E5 is set to 10 and the training steps are calculated such that one epoch is roughly a full run of DROP's training set.

To make best use of the TPUs in Colab Pro, our batch sizes (always a multiple of 8) are set to the highest possible values without causing memory leakage. We found batch sizes of 256 and 64 appropriate for T5-Small and T5-Base respectively. Building on preliminary experiments, we follow the same learning rate schedule in Section 5, and enforce the digit-by-digit tokenization across all models. We also convert data to TFRecord files as a workaround to Hugging Face's bug on data streaming through its transformar APIs, and shuffle by example prior to training. Lastly, we execute our sequential and continuous training schedules according to Figure 2.

Learning Rate We experiment multiple learning rate schedules on the development set, including constant rates (1e-3, 1e-4, 1e-6) with Adam as the optimizer, and various warm-up/decay rates ([1e-6, 1e-10] and [1e-3, 1e-7] respectively). In the end, we settle on the following parameters: warm-up starting rate (1e-8), warm-up ending rate (1e-4), decay rate after warm-up (1e-3). We allocate 10% of the total epochs to warm-up, during which the rate increases linearly by batch. Once the warm-up ends, the learning rate decays by epoch according to the following schedule:

$$\text{LR} = \frac{\text{LR}}{(1 + \text{decay rate}(\text{epoch} - \text{warmup epoch}))} \quad (1)$$

Synthetic Datasets Here we briefly discuss the two synthetic datasets by Geva et al: NUM and TXT. NUM is a dataset consisting of 1M mathematical expressions synthetically generated using six different types of templates that correspond to various numeric operations. An example of the "combination" templates is "s1 f1 s2 f2 s3 f3", which can translate to a synthetic example of "517.4 - 17484 - 10071.75 + 1013.21." The symbol "s1" defines the sign for the first digit in the mathematical expression and "f1" defines the first digit to be a float type. Another example is the "max/min/avg" template, "o(f1, f2, f3)," where "o" symbolizes the operation of min/max/avg. The symbols are converted to their associated numbers or operations randomly together with correct answers during data generation.

TXT is a dataset consisting of 2M Q&A examples synthetically generated by building on top of the mathematical skills found in NUM. In addition, the data generation is based on multiple sentence, question, and context templates that are created by following the "world state" framework created by Hosseini et al. (2014) to build math word problems. The extracted templates are sentences with "world states" which consist of countable *entities*, *entity containers* (owners of the entities), and *verbs* (which describe the changes of entities and entity containers) that can be filled. Synthetic context-question pairs with answers can then be randomly generated by filling in the extracted templates with domain-specific vocabularies that correspond to the world states. For more details on the procedures and method of the generation of the Textual data, please refer to Geva et al. (2020), and Hosseini et al. (2014).

Temperature-Scaled Mixing We hypothesize that multitasking would improve performance because in our work, T5 needs to find a way to "not forget" the transferred RC ability while pre-trained on the NRoT-specific examples (1M NUM and 2M TXT) that are significantly larger than DROP (77K). To test our hypothesis, we multitask-train our model by mixing different datasets with temperature-scaled mixing, a strategy utilized in the original work of T5 (Raffel et al., 2019) and *How multilingual is Multilingual BERT?*(Pires et al., 2019) This scaling technique is designed to mitigate the issue of a model overfitting to the larger dataset during multitasking due to the magnitude of disparity between datasets included. The formula for calculating the temperature scaling rate is as following:

$$r_m = min(length_m \times scale)^{\frac{1}{T}} \quad (2)$$

where T is the temperature used across datasets for adjusting mixing proportions, $length_m$ is the size of $dataset_m$, $scale$ modifies the scales of datasets, and r_m is the unnormalized ratio for $dataset_m$. This calculated rate, r_m, is then normalized for the sampling ratio of each dataset. When $T = 1.0$, this approach is equivalent to examples-proportional mixing (i.e. datasets get samples proportionally relative to their original sizes), and no adjustment is done to mitigate the disparity between dataset sizes. The proportions of datasets become closer to equal-mixing as T increases, resulting in a 1-to-1 ratio of sampling. The greater the T becomes, the more adjustment is done to close the disparity of sizes between datasets, effectively lowering the ratios of larger datasets in the mix.

Contribution to the Hugging Face Community We plan to contribute our notebooks to the Hugging Face (HF) community given the current shortage of learning resources on training T5 with HF's TensorFlow APIs. In particular, we build our models and custom workflow by overcoming the following technical challenges: Limited computing resources, HF's bugs and limited support on T5's multitasking, and Lack of learning resources on T5 using HF's TensorFlow APIs. The scale of our training schedules demand parallel TPUs/GPUs. Unfortunately, we were unable to get approved for anything more than a single basic GPU from AWS and GCP. In the end, the only reasonable computing resource we have is Colab, but even with the paid Pro plan, we are banned from the service multiple times over the course of our training. In addition, our progress was slowed down by HF's bugs on transformer APIs (which prevent us from streaming data using TensorFlow's generator) and lack of support for T5 (e.g. multitask training). Lastly, of the entire HF community today, we find one single notebook (with limited documentation) on training T5 using HF's TensorFlow APIs (although resources for T5 on PyTorch are abundant).

context	question	prediction	ground truth
The 2010 United States Census reported that Lassen County had a population of 34,895. The racial makeup of Lassen County was 25,532 (73.2%) White (U.S. Census), 2,834 (8.1%) African American (U.S. Census), 1,234 (3.5%) Native American (U.S. Census), 356 (1.0%) Asian (U.S. Census), 165 (0.5%) Pacific Islander (U.S. Census), 3,562 (10.2%) from Race (United States Census), and 1,212 (3.5%) from two or more races. Hispanic (U.S. Census) or Latino (U.S. Census) of any race were 6,117 persons (17.5%).	How many more percent of people were white than Native American?	68.8	69.7
According to , the total population was in compared to 6,077,000 in 1950, and around 1,700,000 in 1900. The proportion of children below the age of 15 in 2010 was 42.5%, 54.9% between the ages of 15 and 65, and 2.7% was 65 years or older. Worldometers estimates the total population at 48,466,928 inhabitants, a 29th global rank.	Which year had the highest total population, 1950 or 1900?	1900	1950
In the county, the population is spread out with 24.60% under the age of 18, 6.40% from 18 to 24, 22.80% from 25 to 44, 25.40% from 45 to 64, and 20.80% who are 65 years of age or older. The median age is 42 years. For every 100 females there are 95.90 males. For every 100 females age 18 and over, there are 90.50 males.	What age group has the highest percentage of the population; 25 to 44 or or 45 to 64?	25 to 44	45 to 64
There were 20,928 births in 2006. Of these, 19,757 (94.40% of the births, 95.19% of the population) were to Non-Hispanic Whites. There were 22 births to American Indians (0.11% of the births and 0.54% of the population), 177 births to Asians (0.85% of the births and 0.68% of the population), 219 births to Hispanics (1.05% of the births and 0.88% of the population) and 753 births to Blacks and others (3.60% of the births and 3.56% of the population).	What races had more than 700 births?	American Indians, Asians	Non-Hispanic Whites, Blacks

Table 5: Wrong predictions on one example of each question type made by the baseline model. The best model(E3 phase 5) has answered these questions correctly(EM=1).