Sanitizing Synthetic Training Data Generation for Question Answering over Knowledge Graphs

Trond Linjordet
University of Stavanger
Stavanger, Norway
trond.linjordet@uis.no

Krisztian Balog
University of Stavanger
Stavanger, Norway
krisztian.balog@uis.no

ABSTRACT

Synthetic data generation is important to training and evaluating neural models for question answering over knowledge graphs. The quality of the data and the partitioning of the datasets into training, validation and test splits impact the performance of the models trained on this data. If the synthetic data generation depends on templates, as is the predominant approach for this task, there may be a leakage of information via a shared basis of templates across data splits if the partitioning is not performed hygienically. This paper investigates the extent of such information leakage across data splits, and the ability of trained models to generalize to test data when the leakage is controlled. We find that information leakage indeed occurs and that it affects performance. At the same time, the trained models do generalize to test data under the sanitized partitioning presented here. Importantly, these findings extend beyond the particular flavor of question answering task we studied and raise a series of difficult questions around template-based synthetic data generation that will necessitate additional research.

CCS CONCEPTS

• **Information systems** → **Question answering**; *Test collections*;
• **Computing methodologies** → Neural networks.

KEYWORDS

Knowledge graph question answering, synthetic data, template-based data generation, information leakage

ACM Reference Format:
Trond Linjordet and Krisztian Balog. 2020. Sanitizing Synthetic Training Data Generation for Question Answering over Knowledge Graphs. In *The 2020 ACM SIGIR International Conference on the Theory of Information Retrieval (ICTIR '20), September 14–17, 2020, Virtual Event, Norway*. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3409256.3409836

1 INTRODUCTION

Synthetic data generation can benefit neural models by producing adequate volumes of training data. Knowledge graph question answering (KGQA)—the problem of mapping natural language questions to SPARQL queries—is a task where deep neural models have recently been introduced. Neural models for KGQA by their high-volume data requirements bring about the need for synthetic data generation. However, unlike for other tasks like ad hoc document retrieval [4], query clarification terms [15], or query autocompletion [24], synthetic data generation for KGQA has so far been developed in a template-based manner. This raises a number of interesting methodological questions.

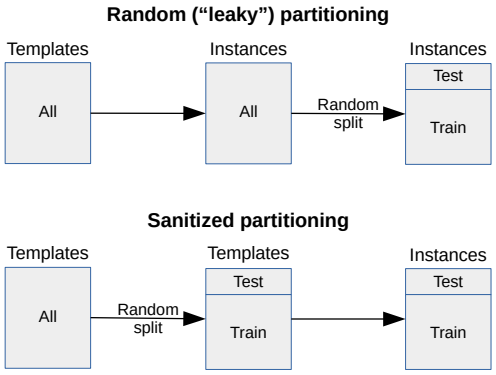

Figure 1: Illustration of leaky and sanitized partitioning for template-based synthetic data generation.

In particular, we consider a hypothesis that training models on template-based synthetic data instances may result in *information leakage* if the partitioning of synthetic data into training, validation, and test splits is not done carefully. If the training and test splits are randomly partitioned without regard for the underlying templates, it is possible that a significant portion of the performance seen in trained models is not coming from correct generalizations. Instead, some portion of the observed performance may come from memorizing the underlying patterns of the finite set of templates, which are common across the training, validation, and testing splits. This *leaky* partitioning condition is illustrated in the top part of Fig. 1. To explore the hypothesis, we devised an alternative, *sanitized* partitioning scheme, illustrated in the bottom part of Fig. 1.

As a guide to intuition, we can imagine that the KGQA models trained on template-based instances will "see" through the instance to the underlying template, which is therefore considered *seen* with respect to the trained model. The question of the trained models' ability to generalize can then be cast as a question of how the trained models perform on instances generated from *unseen* templates.

ICTIR '20, September 14–17, 2020, Virtual Event, Norway

ACM ISBN 978-1-4503-8067-6/20/09…$15.00
https://doi.org/10.1145/3409256.3409836

We address three research questions:

- **RQ1:** Is the performance of trained neural KGQA models affected by whether testing templates are *seen* or *unseen*?
- **RQ2:** Is the ability to generalize to instances based on *unseen* templates affected by the volume of training data used?
- **RQ3:** Can the proportion of *unseen* templates to *seen* templates affect the trained models' ability to generalize?

Specifically, we look at complex KGQA, which is a variant of KGQA where the formal query represents a multi-relation subgraph on the knowledge graph (KG). We investigate the properties of synthetic data in the context of neural network models, using the largest KGQA dataset that exists to date, DBpedia Neural Question Answering (DBNQA) [21]. We empirically compare three neural machine translation (NMT) architectures that represent a specific family of neural network architectures, recurrent neural networks (RNNs), which were shown to be effective on this task [11, 33, 34].

In the *leaky* partitioning, instances are randomly assigned to splits without regard for underlying templates. This *leaky* partitioning is both convenient and provides the models with the maximum volume and variety of training instances. In the *sanitized* partitioning, templates are partitioned into train and test splits, and instances are then partitioned so that test instances will only be those generated from *unseen* templates. If the synthetic data is not generated in a *sanitized* manner initially, the *sanitized* partitioning requires additional processing to achieve: first templates must be partitioned into test and training splits; then the generated instances must be matched with the templates they were generated from; and finally, the instances must be allocated to test and training splits, accordingly. Nevertheless, this approach helps minimize information leakage when testing model performance.

Empirically, we observe the expected loss of performance on sanitized test splits compared to leaky validation splits (RQ1).

When adjusting the volume of sanitized training data, we see small but consistent performance increase in response to increased training data, even with respect to instances from unseen templates (RQ2). Likewise, when the proportion of templates assigned to the training split is increased, the performance on sanitized test splits is low, but does show improvement (RQ3). These results indicate that while information leakage and memorizing the patterns of seen templates account for a lot of the performance observed under the leaky partitioning, some generalization does happen from instances of seen templates to instances of unseen templates.

To summarize, the most important contribution of this study is the identification of the problem of information leakage in template-based synthetic generation approaches. Using a large KGQA dataset, we show that that random partitioning as in [46] is indeed leaky, giving a misleading impression of the performance of the trained models. The significance of our finding, however, extends beyond KGQA, as it applies to any template-base data generation approach, and raises a set of interesting questions around training models with synthetic data using fair conditions. We present a novel dataset partitioning scheme that provides a facility to quantify the generalized learning achieved by models trained on template-generated synthetic data.

2 BACKGROUND

To satisfy the need for large volume datasets to train deep learning models, various approaches have been explored to enhance the collection of real data points, such as data augmentation [12, 32] and synthetic data generation [28]. One way to accomplish synthetic data generation is engineering with domain knowledge a reliable model from which to sample data points, e.g., creating computer 3D models to sample images [3]. Learning generative models from a small initial dataset is another way to establish a source of synthetic data. Two prominent approaches towards machine learning generative models are generative adverserial networks [20] and variational autoencoders [23].

The distinction between synthetic and augmented data can become ambiguous in some cases, as augmented data means taking data from real measurements and changing the data in some way that preserves key qualities of the data point while challenging the model to learn the preserved relationships. On the one hand "synthetic" data could be considered to include all data that are not the result of direct measurement[1] , which would subsume data augmentation. On the other, "synthetic data" implies that the data is constructed to represent an underlying distribution beyond simply transforming original data with certain invariances. Following the above distinction, DBNQA [21] may represent an ambiguous case, as semantically, the generated instances are all novel as in synthetic data, but syntactically, they are variations that serve to reinforce the shared pattern, as in data augmentation. Our work hopefully elucidates this further.

Generating synthetic data to train machine learning models has been done for a large number of tasks. Within the field of information retrieval (IR), synthetic data generation has been explored to train models for various tasks, including ad hoc document retrieval [4], suggesting NLQs to clarify search intent from query terms [15], and query auto-completion [24].

Synthetic data generation has also been used for question answering (QA) tasks [2, 19, 43, 47]. Much effort has focused on the machine reading comprehension (MRC) variant of QA, where questions should be answered in the context of a prose paragraph. For example, Golub et al. [19] looked at how to improve transfer learning, fine-tuning a model (pre-trained on one source domain MRC dataset) with synthetic MRC data generated from the target domain corpus of context paragraphs. The common approach, also taken by Alberti et al. [2], is to use neural language models to select answer spans from paragraphs, and to generate questions conditioned on the answer and paragraph.

We focus on one particular flavor of QA, KGQA, and the datasets and synthetic data generation approaches for this task are discussed in more detail in Sect. 3.1.

3 KGQA DATASETS AND APPROACHES

Knowledge-graph question answering (KGQA) is the task of, given a natural language query q, predicting a formal query f that executes on a knowledge graph (KG) $\mathcal{K}$ to return the correct answer a, and where f also correctly represents the meaning of q.

[1] McGraw-Hill Dictionary of Scientific and Technical Terms. Retrieved November 29, 2009.

Table 1: Datasets for complex KGQA.

Dataset	KG	Size	Generation	Manual post-processing
QALD-{1-9}[2]	DBpedia	~50-500 each	Manual	N/A
QALD-7-train [39]	DBpedia	517	Manual	Programmatically filtered
LC-QuAD [37]	DBpedia	5 000	Hand-made templates	Paraphrasing† and reviews‡ of NLQs
LC-QuAD 2.0 [17]	DBpedia, Wikidata	30 000	Hand-made templates	Paraphrasing† and reviews† of NLQs
ComplexWebQuestions [36][3]	Freebase	34 689	Hand-made templates	Paraphrasing† of NLQs
DBNQA [21][4]	DBpedia	894 499	Extracted templates	Reviews‡ of generated templates

† Non-experts, ‡ experts.

In the field of KGQA, a small number of datasets make up the basis for most of the research, as detailed in Sect. 3.1. Given some dataset, there is a relatively greater variety of approaches to build or train KGQA models, some of which are highlighted in Sect. 3.2. In this context, the scope of the present work is explained in Sect. 3.3.

3.1 Datasets

From 2013 and onwards, KGQA research was conducted on datasets typically on a scale of hundreds of data points or more [6–8, 10, 31, 35, 44]. However, a number of these datasets were exclusively or primarily aimed at simple KGQA [8, 31]. As part of the chosen scope in the present work, we consider only datasets for complex KGQA.[5] We further consider only those KGQA datasets where each data point consists of a question-query pair, in other words a natural language question (NLQ) and a logical form or formal query representing the NLQ with respect to the KG. The relevant datasets for complex KGQA found in recent literature (since 2013) are summarized in Table 1. In some cases, the complex KGQA datasets were constructed from simple KGQA datasets, such as ComplexWebQuestions [36], which was constructed from WebQuestionsSP [45]. In the following, these datasets are described in a bit more detail.

A series of datasets, from the Question Answering over Linked Data (QALD) challenges[6], were almost exclusively created manually at small scale. Within this initiative, the re-use and revision of data from previous years has been common. Out of the various QALD datasets, QALD-7-train [39] is highlighted due to its use among the seed data from which Hartmann et al. [21] extracted templates.

The LC-QuAD [37] dataset was created from a set of 38 [7] hand-made abstract query subgraphs extending at most two hops from a seed entity. These were instantiated with whitelisted entities and predicates, and a template for expressing the query as an NLQ was correspondingly populated. This tentative template-based NLQ was paraphrased by crowdsourced non-experts to improve the grammar of the question. The resulting paraphrased NLQs were then reviewed and revised by experts.

The LC QuAD 2.0 [17] dataset was created in a similar manner, except the initial set of 22 query subgraph templates were constructed not from geometric constraints but from consideration of pre-existing QA datasets. The template-based NLQs were paraphrased by crowdsourcing non-experts, the results were likewise paraphrased, and a third round of crowdsourcing verified whether or not the two paraphrases of the NLQ were identical in meaning.

The ComplexWebQuestions [36] dataset was created by taking the simple KGQA question-query pairs from WebQuestionsSP [44] and constructing templates to add constraints to each data point to produce complex KGQA data. The NLQs were extended with manually constructed predicate-specific templates. Tentative template-based NLQs were then paraphrased by crowdsourced non-experts.

Finally, the DBNQA [21] dataset was constructed by extracting templates of paired NLQ and SPARQL queries, one from each seed data point selected from QALD-7-train [39] and LC-QuAD [37]. The templates derived from QALD-7-train were extracted manually. Meanwhile, templates could be extracted semi-automatically from LC-QuAD: first a script exploited the indicated surface forms in the NLQs, and then the resulting templates were reviewed by SPARQL experts. For each entity URI or surface form in the seed data, corresponding placeholders were inserted in the templates. The templates were then instantiated using the results of the executable SPARQL templates applied to a DBpedia endpoint to find entities for the placeholders.

The datasets listed in Table 1 indicate a trend towards scalable instance generation to economically generate larger volumes of question-query pair data for complex KGQA. The progression started with fully manual dataset generation, moving to using hand-made templates for automated instance generation, and now with DBNQA the automated template generation from pre-existing seed datasets. While these changes in automation have increased the scale of available datasets, the need for manual post-processing also increases. Consequently, it becomes economically desirable to divide the post-processing work into (i) work that requires expert knowledge of the formal query language, and (ii) work that can be adequately performed by crowdsourced non-experts. The non-experts only need adequate natural language skills to ameliorate the grammatical artifacts of template-based NLQ generation.

3.2 Approaches

Various approaches have been taken for developing complex KGQA systems. One important distinction is between neural and non-neural approaches. The former represents a recent trend, focusing on the development of suitable neural architectures for end-to-end learning, while the latter tends to decompose the KGQA task into a sequence of discrete subtasks and create purpose-built solutions for each.

[5] While some of the excluded KGQA datasets, such as WebQuestions [44] and WebQuestionsSP [45], may include some complex queries, the majority of their queries were simple.

[6] https://project-hobbit.eu/, https://github.com/ag-sc/QALD

[7] The published file only contains 35 templates.

3.2.1 Non-neural Approaches. As if to indicate the pervasive shift towards neural approaches, Chakraborty et al. [11] refer to non-neural KGQA approaches as "traditional" [7, 30, 38]. Diefenbach et al. [14] consider all KGQA tasks to consist of distinct stages, all of which must be solved by the KGQA system. For example, the message-passing architecture QAmp [40] can be considered a hybrid of neural and non-neural approaches, but is not an end-to-end neural system. QAmp uses neural components (RNN classifiers, word embeddings) in its question interpretation stage, in addition to index-based methods. In the answer inference stage, the components are using probabilistic graphical models approaches. Another hybrid approach was devised by Abujabal et al. [1], who separated the semantic parsing for KGQA into various components: an offline neural component that learned from KGQA data to generate the syntactic template components that were used compositionally to parse the semantics of an NLQ into a formal query; another component that generates candidate queries; and a component to rank the candidate queries to output the inferred formal query.

3.2.2 Neural Architectures. Perhaps the most common way neural networks are used end-to-end in KGQA is to treat the formal query lanuage, e.g., SPARQL, as a target language in neural machine translation (NMT). As KGQA is cast as a semantic parsing task, this makes sense, although the strict syntax of the formal query language differs from the more varied and flexible syntax in natural languages.

As shown by Yin et al. [46], a number of NMT KGQA architectures can be successfully trained on KGQA data, i.e., NLQ and SPARQL query pairs. Besides the architectures mentioned in Sect. 3.3, Yin et al. [46] tested the following architectures on KGQA datasets, including DBNQA:

- Two variants of Google's NMT architectures from Wu et al. [42] were tested, *GNMT-4* and *GNMT-8*, which are respectively 4- and 8-layer LSTM-based RNNs with a bi-directional encoding layer.
- *LSTM_Luong*, the LSTM-based 4-layer RNN with local attention, introduced by Luong et al. [27] was also tested.
- A single CNN-based architecture was tested, *ConvS2S* [18].
- The *Transformer* [41] architecture was also tested.

Neural KGQA is surveyed in greater detail by Chakraborty et al. [11], including classification and ranking approaches, especially for simple KGQA, as well as machine translation approaches, which may be more suitable for complex KGQA. Various neural machine translation (NMT) approaches to KGQA have been investigated in previous work Dong and Lapata [16], Jia and Liang [22], Liang [25], Soru et al. [33, 34], Yih et al. [44, 45].

3.3 Scope

Out of the available approaches to KGQA, we consider only neural machine translation (NMT) architectures, which are interpreted as performing *semantic parsing* on the NLQ q to produce a semantically equivalent formal (SPARQL) query f that also executes on the target knowledge graph $\mathcal{K}$, retrieving the correct answer a.

In the present work, we limit ourself to a particular baseline architecture and its variants, to ensure the comparability of the obtained results. We note that the same experiments can be performed with additional architectures in the future. Specifically, the baseline architecture is taken from [46] (originally from [33, 34]), as well as two attention-based variations of this architecture, which in [46] performed well on DBNQA. The work of Yin et al. [46] was taken as a starting point because it was the only work that considered a variety of NMT architectures applied to the largest available complex KGQA dataset, DBNQA [21]. The selection was made both due to the high performance on the randomly partitioned DBNQA, as well as the fact that these models were implemented in the same framework, Tensorflow.

NSpM baseline The baseline architecture is here referred to as **NSpM baseline** following [33, 34, 46]. However, it is a basic Tensorflow NMT architecture, with 2 layers, 128 units per layer, a dropout rate of 20%, and optimizing on the BLEU metric.

NSpM+Att1 The second architecture is called **NSpM+Att1**, again following [46], and it differs from NSpM baseline only in that a global Bahdanau attention mechanism is added [5]. Since the type of global Bahdanau attention mechanism utilized in [46] was not further specified, the present work selected a "normed" variant.

NSpM+Att2 The third architecture is called **NSpM+Att2**, again following [46], and it differs from NSpM baseline only in that a local Luong attention mechanism is added [27]. Since the type of local Luong attention mechanism utilized in [46] was not further specified, the present work selected a "scaled" variant. This architecture had the second-best performance of the 8 architectures evaluated in [46], second only to the Convolutional sequence-to-sequence architecture **ConvS2S**, and even that difference was relatively slight.

4 METHODOLOGY

We are looking at a dataset [21] where instances were generated with templates extracted from seeds [37, 39]. The evaluation of this synthetic dataset was done with randomly partitioned training, validation, and test splits [46]. This random partitioning did not avoid allocating instances generated from the same template to different splits. Thus, models trained and evaluated on this random partitioning would see "familiar" instances in the validation and test splits, i.e., instances generated from the same template as instances used for training that model. Could this have created an information leakage, whereby the trained models have memorized a finite set of underlying templates—those *seen* during training—rather than learning to generalize from training instances to previously *unseen* patterns?

To answer this question, we have designed a method to *sanitize*[8] the existing dataset, and ensure that a held-out test split contains only instances generated from a held-out split of templates. Since we are working with a pre-existing dataset with no labelling or index of which template generated each instance, the sanitation process is inevitably somewhat uncertain and depends on constructing a reasonable set of rules to identify which template generated an instance. We make a best effort to recover this information, that is, the originating template for each instance.

[8]This term might be value-laden and undescriptive of the mechanics employed, but reflects the intention to remove or minimize and contamination due to information leakage.

Table 2: Examples of seed, template, and instance.

Seed	NLQ	*Is Peter Piper Pizza in the pizza industry?*
	SPARQL	`ASK WHERE {<http://dbpedia.org/resource/Peter_Piper_Pizza> <http://dbpedia.org/ontology/industry> <http://dbpedia.org/resource/Pizza>}`
Template	NLQ	*Is <B> in the <A> industry?*
	SPARQL	`SELECT DISTINCT ?a, ?b WHERE {?b <http://dbpedia.org/ontology/industry> ?a}`
Instance 1	NLQ	*Is robot comics in the publishing industry?*
	SPARQL	`ASK WHERE {<http://dbpedia.org/resource/Robot_Comics> <http://dbpedia.org/ontology/industry> <http://dbpedia.org/resource/Publishing>}`
Instance 2	NLQ	*Is tiger aircraft in the aerospace industry?*
	SPARQL	`ASK WHERE {<http://dbpedia.org/resource/Tiger_Aircraft> <http://dbpedia.org/ontology/industry> <http://dbpedia.org/resource/Aerospace>}`

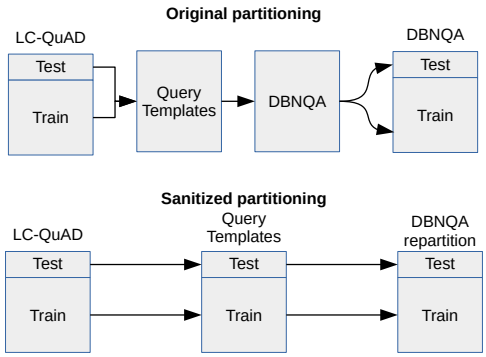

Figure 2: Illustration of original and sanitized DBNQA partitioning.

We can consider that the two approaches taken, shown in Fig. 2, the fully random partitioning by Yin et al. [46], and the sanitized partitioning in the present work, may represent two extremes in how template-based synthetic data should be treated. This perspective is further developed in Sect. 6.

4.1 Preliminaries

The pipeline utilized by Hartmann et al. [21] to generate a large volume of KGQA training data can be considered as three discrete stages, which we refer to as *seeds*, *templates*, and *instances*. First, a small high-quality KGQA dataset consisting of question-query pairs is taken as the seed dataset $s \in \mathcal{S}$ from which are extracted templates $t \in \mathcal{T}$, capturing the underlying pattern of the seed data points, but replacing certain parts of the seed data points with placeholder tokens or URIs, for NLQ and SPARQL forms, respectively. The templates are then instantiated into concrete data points, replacing the placeholders in a template with appropriate terms (entity labels) or entity URIs. Each template can be used to generate an arbitrary number of such new instances $i \in \mathcal{I}$, bounded only by the availability of unique paths (subgraphs) on the knowledge graph that fit the path(s) of the template. The different stages are illustrated in Table 2, with a pair of NLQ and SPARQL forms for each stage. The examples are chosen such that the template is derived from the seed, and the instances are both generated from the same template. Two example instances are shown to illustrate the similarities of instances generated from the same template.

Table 3: Overview of dataset splits used in our experiments. Five different random splits of original DBNQA were used with these proportions. Sanitized-1 DBNQA and Sanitized-2 DBNQA were based on 20% and 10% test splits in the LC-QuAD seed set, respectively.

Dataset	Train	Validation	Test
Original DBNQA	715 600 (80.0%)	89 449 (10.0%)	89 450 (10.0%)
Sanitized-1 DBNQA	659 313 (74.8%)	73 257 (8.3%)	148 397 (16.8%)
Sanitized-2 DBNQA	726 355 (82.4%)	80 706 (9.2%)	73 906 (8.4%)

4.2 Original DBNQA

The DBNQA dataset is provided without any canonical partitions [21]. Researchers are free to randomly partition the dataset into training, validation, and testing splits. This was done by Yin et al. [46], who reported allocating 80%-10%–10% to the respective splits. However, their unique partitioning is not recoverable from their paper or code repository. The present work randomly partitioned DBNQA in the same proportions, but used specific random seeds. In order to ensure that any differences were not due to random chance, this random partitioning was done five times with different random seeds each time, and the resulting partitions were used in Experiment 1 (see Sect.5.1) to separately train models of each of the three architectures discussed in Sect. 3.2.

4.3 Sanitized DBNQA

In order to investigate the question of information leakage via templates, the train-and-test-splits partitioning of the major part of the seed dataset, LC-QuAD, was used to coordinate a partitioning of the LC-QuAD-based templates dataset previously used in generating DBNQA. Subsequently, the partitioned templates were used to partition the instances dataset, DBNQA. This repartitioning is illustrated in the bottom half of Fig.2. Templates were assigned to

the template test split $\mathcal{T}_{\text{test}} \subset \mathcal{T}$ if the NLQ forms of both the seed and template were identical except where the template placeholders allow a contiguous sequence of tokens in the seed, and if the predicates in the seed SPARQL are the same as those in the template SPARQL. Similarly, instances were assigned to the instance test split $\mathcal{I}_{\text{test}}$ if the NLQ forms match as above, and if all the predicates in the template SPARQL are also in the instance SPARQL, in the same order.

The datasets were also de-duplicated at each stage. The original and sanitized instance datasets are summarized in Table 3. Sanitized-1 DBNQA was based on the canonical split of LC-QuAD into an 80% training split and a 20% test split. Sanitized-2 DBNQA was based on a 90%-10% split of LC-QuAD. Only after repartitioning in this systematic manner based on template matching is the instance training split itself randomly partitioned into a 90% training split and a 10% validation split. Thus, the test split is sanitized with respect to the training split, while the validation split is not. By evaluating trained models on both the test split and validation split, we illustrate the information leakage via templates caused by a purely random partitioning of an instance dataset like DBNQA. Having thus repartitioned the instances, we trained KGQA NMT models as in Yin et al. [46], both reproducing the approach taken by Yin, et al. with the original DBNQA partitionings described in Sect. 4.2, as well as on the repartitioned datasets, to compare the results of training with and without information leakage across the dataset splits.

5 EXPERIMENTS

This section presents a series of experiments we performed on the original and sanitized DBNQA collections, to answer our research questions. Model performance is evaluated in terms of the metrics BLEU [29] and perplexity [9], for comparison with results reported by Yin et al. [46]. These measures are commonly used in machine translation evaluation, especially BLEU, which reflects how well the predicted output matches with the ground truth. Perplexity reflects the degree of surprise caused by the model's predictions compared to the ground truth.

5.1 Sanitized data partitioning

Is the performance of trained models affected by the sanitized data partitioning? We addressed this question with our first experiment, which compared models trained on either the original DBNQA, or on Sanitized-1 DBNQA, which was partitioned based on the canonical partitioning of LC-QuAD as described in Sect. 4.3. The original DBNQA was randomly partitioned five times with unique random seeds but identical proportions between the splits (80% – 10% – 10%), and the models were independently trained on each random partition. The performance results of original DBNQA are therefore shown as the mean and ± standard deviation of the models across the five random partitionings. These results, listed in Table 4, show that the performance of the trained models was very similar across the test and validation splits of original DBNQA, as well as the unsanitized validation split of Sanitized-1 DBNQA. However, the sanitized test split showed a significant reduction in performance across all the trained models, in terms of both BLEU and perplexity.

5.2 Varying volumes of training data

Is the performance of the trained models with respect to the sanitized test split dependent on the amount of training data? This would indicate whether there is some degree of generalization from instances based on templates that have been *seen* before during training, or if there is no discernible generalization at all. We addressed this question with our second experiment, which compared the effects of different amounts of training data on the trained models' performance with respect to the Sanitized-1 DBNQA validation and test splits. Although performance increase as a function of increased training data volume is the expected behavior, it has been shown to not always be the case in QA [26]. Thus, it is not *a priori* certain that the sanitized test split is similar enough to the sanitized training split that the expected behavior occurs. This experiment verifies whether or not what the models learn generalizes to the sanitized test split *proportionally* to the volume of training data. Thus, the experiment also elucidates whether the models are only learning to memorize the *seen* template patterns, or is learning to generalize to instances from *unseen* templates.

We trained with fractions of the training data used in our first experiment: 12.5%, 25%, 50%, and 100%. From the results shown in Fig. 3, there is a clear trend for the models NSpM+Att1 and NSpM+Att2, where increased amounts of training data yield improved model performance on both the sanitized test split and the unsanitized validation split, in terms of both BLEU and perplexity. The NSpM baseline model, on the other hand, does not benefit as much from increased training data, improving in terms of perplexity, but even deteriorating slightly in terms of BLEU. However, this also holds for the unsanitized validation split, and so reflects on the architecture's ability to improve from training data, not on the sanitized partitioning.

For all models, performance is reduced by the challenge of the sanitized test split, but where performance improves with increased training data, it does so even on the sanitized test split.

5.3 Varying size of seed partitions before santitation

Is it possible that different partitions of the seed set can affect the degree of generalization? To address this question, in our third experiment we investigated whether increasing the proportion of templates *seen* via the training instances would translate into improved performance on the sanitized test split. We divided the canonical test split of LC-QuAD in half, and added one half back into the seed training split, before doing the sanitizing procedure, yielding Sanitized-2 DBNQA, as described in Sect. 4.3.

As can been seen from results shown in Table 5, here as in our second experiment the performance of all models on the unsanitized validation split was generally better than the performance of models trained on the original DBNQA. Models trained on Sanitized-2 DBNQA performed similarly to models trained on Sanitized-1 DBNQA, with some variations on the order of the standard deviations seen for original DBNQA in Table 4. We note, however, that all models trained on Sanitized-2 DBNQA performed better in terms of perplexity with respect to the sanitized test split.

Table 4: Results of comparing original DBNQA and Sanitized-1 DBNQA to train and evaluate models.

Architecture	Original DBNQA				Sanitized-1 DBNQA			
	BLEU		Perplexity		BLEU		Perplexity	
	Valid.	Test	Valid.	Test	Valid.	Test	Valid.	Test
NSpM baseline	62.56 ± 0.10	62.52 ± 0.10	2.37 ± 0.01	2.37 ± 0.01	64.96	41.12	2.33	11.86
NSpM+Att1	79.27 ± 1.82	79.22 ± 1.77	1.58 ± 0.06	1.58 ± 0.06	85.08	54.39	1.60	7.73
NSpM+Att2	80.58 ± 0.95	80.53 ± 0.89	1.54 ± 0.03	1.54 ± 0.02	84.67	54.55	1.61	9.01

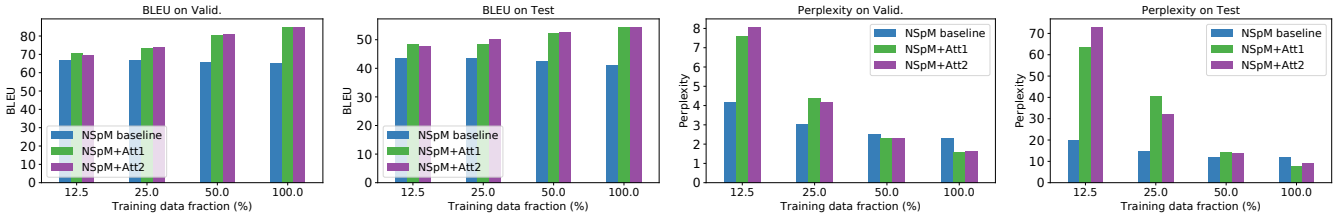

Figure 3: Results of comparing models trained on different fractions of the Sanitized-1 DBNQA training split.

Table 5: Results of evaluating models trained on Sanitized-2 DBNQA.

Architecture	BLEU		Perplexity	
	Valid.	Test	Valid.	Test
NSpM baseline	64.35	42.58	2.36	10.01
NSpM+Att1	82.87	53.09	1.59	7.33
NSpM+Att2	86.05	56.94	1.52	6.57

6 DISCUSSION AND CONCLUSION

As we have seen from the datasets presented in Sect. 3.1, there is a trend towards satisfying an important desideratum of machine learning generally, and deep learning in particular: (i) a large-scale training dataset of such quality and variety that it allows the model to observe and learn to predict patterns when presented new data from the same distribution. However, there is an important second desideratum: (ii) a test set that comprises data from the same distribution but which is novel enough to the model so that model performance is due to the model learning the underlying dynamics of the data, rather than memorizing a finite set of patterns.

In the present work, we have questioned whether DBNQA as used in previous work [46] satisfies (ii). Our hypothesis is that there is a leakage of information between the DBNQA training split on the one hand and the validation and test splits on the other, as used by Yin et al. [46]. We argue that Yin et al. [46] have sacrificed (ii) in favor of (i), while in this paper we considered the other extreme, where (i) is sacrificed in favor of (ii). For future work, we speculate, is there a middle ground that can be reliably found?

In our experiments, we first showed in Sect. 5.1 that there is indeed a large difference in performance on the test split of our sanitized DBNQA partitioning, compared to the validation split, which is randomly partitioned in a template-naive manner. From our second experiment in Sect. 5.2, we showed that for models that improve with increased volumes of training data, that improvement also generalizes to the sanitized test split. Finally, in our third experiment in Sect. 5.3, the models trained on Sanitized-2 DBNQA showed some tendency to improve performance on both validation and test split, indicating generalization from *seen* to *unseen* templates.

Our results raise a set of interesting questions around training models with synthetic data using fair conditions. These are questions raised by the present study that may be the subject for future work: How well do these findings generalize to other model families than those tested here? Of particular interest are the architectures of ConvS2S [18], Transformer [41], and BERT [13]. Can the distinction between memorization and generalized learning be precisely characterized? How can synthetically generated training data be structured to promote learning dynamics (e.g., of a formal syntax) rather a finite set of fixed patterns? For template-based synthetic data generation, what should be the relationship between training and test splits to fairly evaluate the performance of trained models?

In summary, we have shown that several NMT-based neural KGQA systems have reduced performance on instances generated from templates where the models saw no instances generated from those templates during training. At the same time, the performance on instances from such *unseen* templates did show improvement from increased training data, indicating that some models were able to generalize better with more training data.

We have shown that a significant part of performance in these models as reported by Yin et al. [46] may largely be attributed to the models learning to recognize the underlying patterns of specific templates from which were generated the instances seen during training. In contrast, the ideal NMT KGQA system would learn the underlying syntaxes of the source and target languages and handle unseen patterns according to implicit principles.

REFERENCES

[1] Abdalghani Abujabal, Mohamed Yahya, Mirek Riedewald, and Gerhard Weikum. 2017. Automated Template Generation for Question Answering over Knowledge Graphs. In *Proc. of WWW '17*. 1191–1200.

[2] Chris Alberti, Daniel Andor, Emily Pitler, Jacob Devlin, and Michael Collins. 2019. Synthetic QA Corpora Generation with Roundtrip Consistency. In *Proc. of ACL '19*. 6168–6173.

[3] M. Aubry and B. C. Russell. 2015. Understanding Deep Features with Computer-Generated Imagery. In *Proc. of ICCV '15*. 2875–2883.

[4] Leif Azzopardi, Maarten de Rijke, and Krisztian Balog. 2007. Building Simulated Queries for Known-item Topics: An Analysis Using Six European Languages. In *Proc. of SIGIR '07*.

[5] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural Machine Translation by Jointly Learning to Align and Translate. In *Proc. of ICLR '15*.

[6] Junwei Bao, Nan Duan, Zhao Yan, Ming Zhou, and Tiejun Zhao. 2016. Constraint-Based Question Answering with Knowledge Graph. In *Proc. of COLING '16*. 2503–2514.

[7] Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic Parsing on Freebase from Question-Answer Pairs. In *Proc. of EMNLP '13*. 1533–1544.

[8] Antoine Bordes, Nicolas Usunier, Sumit Chopra, and Jason Weston. 2015. Large-scale Simple Question Answering with Memory Networks. (2015). arXiv:1506.02075

[9] Peter F. Brown, Stephen A. Della Pietra, Vincent J. Della Pietra, Jennifer C. Lai, and Robert L. Mercer. 1992. An Estimate of an Upper Bound for the Entropy of English. *Computational Linguistics* 18, 1 (1992), 31–40.

[10] Qingqing Cai and Alexander Yates. 2013. Large-scale Semantic Parsing via Schema Matching and Lexicon Extension. In *Proc. of ACL '13*. 423–433.

[11] Nilesh Chakraborty, Denis Lukovnikov, Gaurav Maheshwari, Priyansh Trivedi, Jens Lehmann, and Asja Fischer. 2019. Introduction to Neural Network based Approaches for Question Answering over Knowledge Graphs. (2019). arXiv:1907.09361

[12] Tri Dao, Albert Gu, Alexander Ratner, Virginia Smith, Chris De Sa, and Christopher Re. 2019. A Kernel Theory of Modern Data Augmentation. In *Proc. of ICML '19 (PMLR)*, Vol. 97. 1528–1537.

[13] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proc. of NAACL '19*. 4171–4186.

[14] Dennis Diefenbach, Vanessa López, Kamal Deep Singh, and Pierre Maret. 2017. Core techniques of question answering systems over knowledge bases: a survey. *Knowledge and Information Systems* 55 (2017), 529–569.

[15] Heng Ding and Krisztian Balog. 2018. Generating Synthetic Data for Neural Keyword-to-Question Models. In *Proc. of ICTIR '18*. 51–58.

[16] Li Dong and Mirella Lapata. 2016. Language to Logical Form with Neural Attention. In *Proc. of ACL '16*. 33–43.

[17] Mohnish Dubey, Debayan Banerjee, Abdelrahman Abdelkawi, and Jens Lehmann. 2019. LC-QuAD 2.0: A Large Dataset for Complex Question Answering over Wikidata and DBpedia. In *Proc. of ISWC '19*, Chiara Ghidini, Olaf Hartig, Maria Maleshkova, Vojtěch Svátek, Isabel Cruz, Aidan Hogan, Jie Song, Maxime Lefrançois, and Fabien Gandon (Eds.). 69–78.

[18] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. 2017. Convolutional Sequence to Sequence Learning. In *Proc. of ICML '17*. 1243–1252.

[19] David Golub, Po-Sen Huang, Xiaodong He, and Li Deng. 2017. Two-Stage Synthesis Networks for Transfer Learning in Machine Comprehension. In *Proc. of EMNLP '17*. 835–844.

[20] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative Adversarial Nets. In *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger (Eds.). 2672–2680.

[21] Ann-Kathrin Hartmann, Edgard Marx, and Tommaso Soru. 2018. Generating a Large Dataset for Neural Question Answering over the DBpedia Knowledge Base. (2018).

[22] Robin Jia and Percy Liang. 2016. Data Recombination for Neural Semantic Parsing. In *Proc. of ACL '16*. 12–22.

[23] Diederik P. Kingma and Max Welling. 2014. Auto-Encoding Variational Bayes. In *Proc. of ICLR '14*.

[24] Unni Krishnan, Alistair Moffat, Justin Zobel, and Bodo Billerbeck. 2020. Generation of Synthetic Query Auto Completion Logs. In *Proc. of ECIR '20*, Joemon M. Jose, Emine Yilmaz, João Magalhães, Pablo Castells, Nicola Ferro, Mário J. Silva, and Flávio Martins (Eds.). 621–635.

[25] P. Liang. 2013. Lambda Dependency-Based Compositional Semantics. (2013). arXiv:1309.4408

[26] Trond Linjordet and Krisztian Balog. 2019. Impact of Training Dataset Size on Neural Answer Selection Models. In *Proc. of ECIR '19*. 828–835.

[27] Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective Approaches to Attention-based Neural Machine Translation. In *Proc. of EMNLP '15*. 1412–1421.

[28] Sergey I. Nikolenko. 2019. Synthetic Data for Deep Learning. arXiv:1909.11512

[29] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a Method for Automatic Evaluation of Machine Translation. In *Proc. of ACL '02*. 311–318.

[30] Siva Reddy, Mirella Lapata, and Mark Steedman. 2014. Large-scale Semantic Parsing without Question-Answer Pairs. (2014).

[31] Iulian Vlad Serban, Alberto García-Durán, Caglar Gulcehre, Sungjin Ahn, Sarath Chandar, Aaron Courville, and Yoshua Bengio. 2016. Generating Factoid Questions With Recurrent Neural Networks: The 30M Factoid Question-Answer Corpus. In *Proc. of ACL '16*. 588–598.

[32] Connor Shorten and Taghi M. Khoshgoftaar. 2019. A Survey on Image Data Augmentation for Deep Learning. *Journal of Big Data* 6, Article 60 (2019).

[33] Tommaso Soru, Edgard Marx, Diego Moussallem, Gustavo Publio, André Valdestilhas, Diego Esteves, and Ciro Baron Neto. 2017. SPARQL as a Foreign Language. In *Proc. of SEMANTiCS '17 - Posters and Demos*.

[34] Tommaso Soru, Edgard Marx, André Valdestilhas, Diego Esteves, Diego Moussallem, and Gustavo Publio. 2018. Neural Machine Translation for Query Construction and Composition. In *Proc. of ICML '18 - Workshop on Neural Abstract Machines & Program Induction (NAMPI v2)*.

[35] Yu Su, Huan Sun, Brian Sadler, Mudhakar Srivatsa, Izzeddin Gür, Zenghui Yan, and Xifeng Yan. 2016. On Generating Characteristic-rich Question Sets for QA Evaluation. In *Proc. of EMNLP '16*. 562–572.

[36] Alon Talmor and Jonathan Berant. 2018. The Web as a Knowledge-Base for Answering Complex Questions. In *Proc. of NAACL '18*. 641–651.

[37] Priyansh Trivedi, Gaurav Maheshwari, Mohnish Dubey, and Jens Lehmann. 2017. LC-QuAD: A Corpus for Complex Question Answering over Knowledge Graphs. In *Proc. of ISWC '17*, Claudia d'Amato, Miriam Fernandez, Valentina Tamma, Freddy Lecue, Philippe Cudré-Mauroux, Juan Sequeda, Christoph Lange, and Jeff Heflin (Eds.). 210–218.

[38] Christina Unger, Corina Forascu, Vanessa López, Axel-Cyrille Ngonga Ngomo, Elena Cabrio, Philipp Cimiano, and Sebastian Walter. 2014. Question Answering over Linked Data (QALD-5). In *Proc. of CLEF '14*.

[39] Ricardo Usbeck, Axel-Cyrille Ngonga Ngomo, Bastian Haarmann, Anastasia Krithara, Michael Röder, and Giulio Napolitano. 2017. 7th Open Challenge on Question Answering over Linked Data (QALD-7). In *Semantic Web Challenges (SemWebEval '17)*, Vol. 769.

[40] Svitlana Vakulenko, Javier David Fernandez Garcia, Axel Polleres, Maarten de Rijke, and Michael Cochez. 2019. Message Passing for Complex Question Answering over Knowledge Graphs. In *Proc. of CIKM '19*. 1431–1440.

[41] 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*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.). 5998–6008.

[42] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2016. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. (2016). arXiv:1609.08144

[43] Qian Yang, Zhouyuan Huo, Dinghan Shen, Yong Cheng, Wenlin Wang, Guoyin Wang, and Lawrence Carin. 2019. An End-to-End Generative Architecture for Paraphrase Generation. In *Proc. of EMNLP-IJCNLP '19*. 3132–3142.

[44] Scott Wen-tau Yih, Ming-Wei Chang, Xiaodong He, and Jianfeng Gao. 2015. Semantic Parsing via Staged Query Graph Generation: Question Answering with Knowledge Base. In *Proc. of AACL-IJCNLP '15*.

[45] Wen-tau Yih, Matthew Richardson, Chris Meek, Ming-Wei Chang, and Jina Suh. [n.d.]. The Value of Semantic Parse Labeling for Knowledge Base Question Answering. In *Proc. of ACL '16*.

[46] Xiaoyu Yin, Dagmar Gromann, and Sebastian Rudolph. 2019. Neural Machine Translating from Natural Language to SPARQL. (2019).

[47] Shiyue Zhang and Mohit Bansal. 2019. Addressing Semantic Drift in Question Generation for Semi-Supervised Question Answering. In *Proc. of EMNLP-IJCNLP '19*. 2495–2509.