Structured Context and High-Coverage Grammar for Conversational Question Answering over Knowledge Graphs

Pierre Marion[*]
Sorbonne Université, CNRS
Laboratoire de Probabilités, Statistique et Modélisation, LPSM
F-75005, Paris, France
pierre.marion@sorbonne-universite.fr

Paweł Krzysztof Nowak and **Francesco Piccinno**
Google Research
{pawelnow,piccinno}@google.com

Abstract

We tackle the problem of weakly-supervised conversational Question Answering over large Knowledge Graphs using a neural semantic parsing approach. We introduce a new Logical Form (LF) grammar that can model a wide range of queries on the graph while remaining sufficiently simple to generate supervision data efficiently. Our Transformer-based model takes a JSON-like structure as input, allowing us to easily incorporate both Knowledge Graph and conversational contexts. This structured input is transformed to lists of embeddings and then fed to standard attention layers. We validate our approach, both in terms of grammar coverage and LF execution accuracy, on two publicly available datasets, CSQA and ConvQuestions, both grounded in Wikidata. On CSQA, our approach increases the coverage from 80% to 96.2%, and the LF execution accuracy from 70.6% to 75.6%, with respect to previous state-of-the-art results. On ConvQuestions, we achieve competitive results with respect to the state-of-the-art.

1 Introduction

Graphs are a common abstraction of real-world data. Large-scale knowledge bases can be represented as directed labeled graphs, where entities correspond to nodes and subject-predicate-object triplets are encoded by labeled edges. These so-called *Knowledge Graphs* (KGs) are used both in open knowledge projects (YAGO, Wikidata) and in the industry (Yahoo, Google, Microsoft, etc.). A prominent task on KGs is *factual conversational Question Answering* (Conversational KG-QA) and it has spurred interest recently, in particular due to the development of AI-driven personal assistants.

The Conversational KG-QA task involves difficulties of different nature: entity disambiguation, long tails of predicates (Saha et al., 2018), conversational nature of the interaction. The topology of the underlying graph is also problematic. Not only can KGs be huge (up to several billion entities), they also exhibit hub entities with a large number of neighbors.

A recent prominent approach has been to cast the problem as neural *semantic parsing* (Jia and Liang, 2016; Dong and Lapata, 2016, 2018; Shen et al., 2019). In this setting, a semantic parsing model learns to map a natural language question to a *logical form* (LF), *i.e.* a tree of operators over the KG. These operators belong to some grammar, either standard like SPARQL or ad-hoc. The logical form is then *evaluated* over the KG to produce the candidate answer. In the *weak supervision* training setup, the true logical form is not available, but only the answer utterance is (as well as annotated entities in some cases, see Section 5.4). Hence the training data is not given but it is instead generated, in the format of *(question, logical form)* pairs. We refer to this data as *silver data* or *silver LFs*, as opposed to unknown gold ground truth.

However, this approach has two main issues. First, the silver data generation step is a complex and often resource-intensive task. The standard procedure employs a Breadth-First Search (BFS) exploration (Guo et al., 2018; Shen et al., 2019), but this simple strategy is prone to failure, especially when naively implemented, for questions that are mapped to nested LFs. This reduces the *coverage*, i.e. the percentage of training questions associated

[*] Work done while an intern at Google Research.

to a Logical Form. Shen et al. (2020) proposes to add a neural component for picking the best operator, in order to reduce the computational cost of this task, however complicating the model. Cao et al. (2020) proposes a two-step semantic parser: the question is first paraphrased into a “canonical utterance”, which is then mapped to a LF. This approach simplifies the LF generation by separating it from the language understanding task.

Second, most of the semantic parsing models do not leverage much of the underlying KG structure to predict the LF, as in Dong and Lapata (2016); Guo et al. (2018). Yet, this contextual graph information is rich (Tong et al., 2019), and graph-based models leveraging this information yield promising results for KG-QA tasks (Vakulenko et al., 2019; Christmann et al., 2019). However these alternative approaches to semantic parsing, that rely on node classification, have their inherent limitations, as they handle less naturally certain queries (see Appendix C.3) and their output is less interpretable. This motivates the desire for semantic parsing models that can make use of the KG context.

Approach and contributions We design a new grammar, which can model a large range of queries on the KG, yet is simple enough for BFS to work well. We obtain a high coverage on two KG-QA datasets. On CSQA (Saha et al., 2018), we achieve a coverage of 96%, a 16% improvement over the baseline (Shen et al., 2020). On ConvQuestions (Christmann et al., 2019), a dataset with a large variety of queries, we reach a coverage of 86%.

To leverage the rich information contained in the underlying KG, we propose a semantic parsing model that uses the KG contextual data in addition to the utterances. Different options could be considered for the KG context, *e.g.* lists of relevant entities, annotated with metadata or pre-trained entity embeddings that are graph-aware (Zhang et al., 2020). The problem is that this information does not come as unstructured textual data, which is common for language models, but is structured.

To enable the use of context together with a strong language model, we propose the Object-Aware Transformer (OAT) model, which can take as input structured data in a JSON-like format. The model then transforms the structured input into embeddings, before feeding them into standard Transformer layers. With this approach, we improve the overall execution accuracy on CSQA by 5.0% compared to a strong baseline (Shen et al., 2019). On ConvQuestions, we improve the precision by 4.7% compared to Christmann et al. (2019).

2 Related work

Neural semantic parsing Our work falls within the neural semantic parsing approaches for Knowledge-Based QA (Dong and Lapata, 2016; Liang et al., 2017; Dong and Lapata, 2018; Guo et al., 2019b; Hwang et al., 2019). The more specific task of conversational KG-QA has been the focus of recent work. Guo et al. (2018) introduces D2A, a neural symbolic model with memory augmentation. This model has been extended by S2A+MAML (Guo et al., 2019a) with a meta-learning strategy to account for context, and by D2A+ES (Shen et al., 2020) with a neural component to improve BFS. Saha et al. (2019) proposes a Reinforcement Learning model to benefit from denser supervision signals. Finally, Shen et al. (2019) introduces MaSP, a multi-task model that performs both entity linking and semantic parsing, with the hope of reducing erroneous entity linking (see Appendix C.2 for a comparison with our setup). Recently, Plepi et al. (2021) extended the latter in CARTON. They first predict the LF using a Transformer architecture, then specify the KG items using pointer networks. This work was further extended in Kacupaj et al. (2021), which includes a graph attention network to exploit correlations between entities and predicates. Another related contemporaneous work is Thirukovalluru et al. (2021), where the decoder is informed with entity embeddings coming from KG random walks.

Learning on Knowledge Graphs Classical graph learning techniques can be applied to the specific case of KGs. In CONVEX (Christmann et al., 2019), at each turn, a subgraph is expanded by matching the utterance with neighboring entities. Then a candidate answer is found by a node classifier. Other methods include unsupervised message passing (Vakulenko et al., 2019). However, these approaches lack strong NLP components. Other directions include learning differentiable operators over a KG (Cohen et al., 2019), or applying Graph Neural Networks (Kipf and Welling, 2017; Hamilton et al., 2017) (GNNs) to the KG, which has been done for entity classification and link prediction tasks (Schlichtkrull et al., 2018). GNNs have also been used to model relationships between utterances and entities (Shaw et al., 2019).

Structured Input for neural models Our approach of using JSON-like input falls in the line of computing neural embeddings out of structured inputs. Tai et al. (2015) introduced Tree-LSTM for computing tree embeddings bottom-up. It has then been applied for many tasks, including computer program translation (Chen et al., 2018), semantic tree structure learning (such as JSON or XML) (Woof and Chen, 2020) and supervised KG-QA tasks (Tong et al., 2019; Zafar et al., 2019; Athreya et al., 2021). In the latter context, Tree-LSTM is used to model the syntactic structure of the question. Other related approaches include tree transformer (Harer et al., 2019) and tree attention (Ahmed et al., 2019). Syntactic structures were also modeled as graphs (Xu et al., 2018; Li et al., 2020). Specific positional embeddings can also be used to encode structures (Shiv and Quirk, 2019; Herzig et al., 2020).

3 A grammar for KG exploration

Several previous KG-QA works were based on the grammar from D2A (Guo et al., 2018). We also take inspiration from their grammar, but redesign it to model a wider range of queries. By defining more generic operators, we achieve this without increasing the number of operators nor the average depth of LFs. Section 3.4 presents a comparison.

3.1 Definitions

An *entity* (e.g. `Marie Curie`) is a node in the KG. Two entities can be related through a directed labeled edge called a *property* (e.g. `award received`). A property can also relate an entity to a *value*, which can be a date, a boolean, a quantity or a string. Entities and properties have several attributes, prominently a name and an integer ID. The `membership` property is treated separately; it relates a member entity (e.g. `Marie Curie`) to a *class* entity (e.g. `human being`).

The *objects* we will consider in the following are entities, properties, classes, and values. The grammar consists of a list of *operators* that take objects or sets of objects as arguments. A Logical Form is a binary expression tree of operators.

In several places, we perform *Named Entity Linking (NEL)*, i.e. mapping an utterance to a list of KG objects. Section 5.4 details how this is done.

Table 1 lists the operators we use, grouped in five categories. Most of them are straightforward, except meta-operators, which we explain next.

3.2 Meta-operators

Meta-operators are useful for questions such as: *Which musical instrument is played by the maximum number of persons?*. To answer this question, we first compute the set of all musical instruments in the KG. For each entity in this set, we then follow the property `played by`, producing a set of people who play that instrument. Finally, we compute the max cardinality of all these sets and return the associated instrument.

The corresponding LF is the following:

```
argmax(
  cardinality(
    follow_property(
      for_each(
        members(musical instrument),
      ),
      played by)))
```

`for_each` creates a parallel computation over each entity in its argument, which can be terminated by three operators (`arg`, `argmax` and `argmin`). We refer to Appendix B.1 for details.

3.3 Silver LF generation

To generate silver LFs, we explore the space of LFs with a BFS strategy, similarly to Guo et al. (2018); Shen et al. (2019). More precisely, to initialize the exploration, we perform NEL to find relevant entities, values and classes that appear in the question. LFs of depth 0 simply return an annotated object. Then, assume that LFs of depth less or equal to n have been generated and we want to generate those of depth $n+1$. We loop through all possible operators; for each operator, we choose each of its arguments among the already-generated LFs. This algorithm brings two challenges, as highlighted in Shen et al. (2020): computational cost and spurious LFs. We refer to Appendix B.2 for implementation details that mitigate these difficulties.

3.4 Comparison with D2A (Guo et al., 2018)

Section 5.5 shows that our grammar achieves higher coverage with a similar average LF depth. A more thorough quantitative comparison is delicate, as it would require reimplementing D2A within our framework, which is beyond the scope of this paper. On a qualitative basis, we use more elementary types: in addition to theirs, we introduce set of classes, strings and set of values (which can be strings, numerals or booleans). We use eight less operators than D2A; among our oper-

Category	Name	Signature	Description
Graph operators	`follow_property`	(SE, P) → SE	Returns the **entities** which are linked by property P **to** at least one element of SE.
	`follow_backward`	(SE, P) → SE	Returns the **entities** which are linked by property P **from** at least one element of SE.
	`get_value`	(SE, P) → SV	Returns the **values** which are linked by property P to at least one element of SE.
Numerical operators	`max`, `min`	SV → SV	Returns the max (resp. min) value from SV.
	`greater_than`, `equals`, `lesser_than`	(SV, V) → SV	Filters SV to keep values strictly greater than (resp. equal to, strictly lesser than) V.
	`cardinality`	SE → V	Returns the cardinality of SE.
Set operators	`is_in`	(a: SE, b: SE) → SV	Returns a boolean set: for each entity in a, the mask equals True if the entity is in b.
	`get_first`	SE → SE	Returns the first entity from SE.
	`union`, `intersect`, `difference`	(SE,SE) → SE	Returns the union (resp. intersection, difference) of input sets.
Class operators	`members`	SC → SE	Returns the members of classes in SC.
	`keep`	(SE,SC) → SE	Filters SE to keep the members of SC.
Meta-operators	`for_each`	SE → SE	Initializes a parallel computation over all entities in the input set.
	`arg`	SV → SE *or* SE → SE	Ends a parallel computation by returning all entities that gave a non-empty result.
	`argmax`, `argmin`	SV → SE	Ends a parallel computation by returning all entities that gave the max (resp. min) value.

Table 1: List of operators in our grammar. Their variables can be entities (E), classes (C), values (V), ordered sets of such elements (resp. SE, SC and SV), or properties (P).

ators, six are in common (`follow_property`, `follow_backward`, `cardinality`, `union`, `intersect`, `difference`), four are modified (`keep`, `is_in`, `argmax`, `argmin`), and the other ten are new. New intents that can be modeled include numerical reasoning (e.g. *What actor plays the younger child?*), temporal reasoning (e.g. *What is the number of seasons until 2018?*), ordinal reasoning (e.g. *What was the first episode date?*), textual form reasoning (e.g. *What was Elvis Presley given name?*). We refer to Appendix C.1 for more details and comparison methodology.

4 Model

4.1 Overview

The model, called Object-Aware Transformer (OAT), is a Transformer-based (Vaswani et al., 2017) auto-regressive neural semantic parser. As illustrated in Figure 1, the model has several steps.

The first step consists of retrieving relevant objects, annotated with metadata, that might appear in the resulting LF. This step is performed using NEL on the utterances and KG lookups to retrieve the graph context information. At this point, the input is composed of lists of *objects* with their *fields*. After embedding each field value in a vector space, we perform successive layers of input flattening and Transformer encoding. The *Flattener* layer is useful to transform the structured input into a list of embeddings. Then a decoder Transformer layer produces a linearized LF, i.e. a list of string tokens. Finally, we evaluate the LF over the KG to produce a candidate answer. In the next sections, we describe each step in details.

4.2 Structured Input computation

Hierarchical structure For each query, we construct the input as a JSON-like structure, consisting of lists of objects with their fields (represented in the left part of Figure 1). We chose this representation as it allows to incorporate general structured information into the model. A field can be a string, a KG integer ID, a numerical value, or a list thereof.

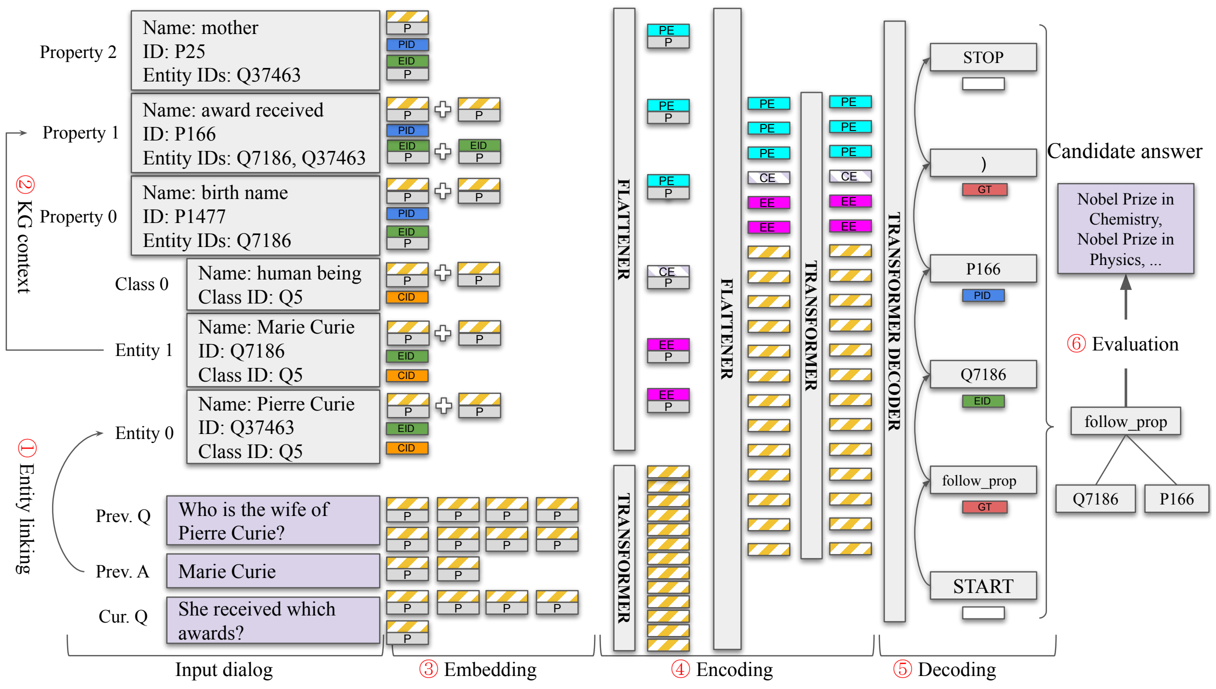

Figure 1: Architecture of the proposed model. The initial field embeddings are Positional (P), Property ID (PID), Entity ID (EID), and Class ID (CID). After the first Flattener layer, we obtain Property Embeddings (PE), Class Embeddings (CE), Entity Embeddings (EE). There are also Grammar Token (GT) embeddings in the output. Note that the entity IDs are actually randomized (not shown here).

To construct the input, we start from the last utterances in the dialog: the current query, previous query and previous answer. We first perform NEL to retrieve a list of entities $\mathcal{E}$ (and numerical values) matching the utterances. The KG is then queried to retrieve additional contextual information: the classes of the entities, and all outgoing and incoming properties from these entities $\mathcal{E}$. This gives a list of properties $\mathcal{P}$. For each property $p \in \mathcal{P}$, we fill several fields: its ID, its name, and an `Entity IDs` field, which corresponds to all entities $e \in \mathcal{E}$ such that at least one *graph operator* gives a non-empty result when applied to e and p. For instance, in Wikidata, the property `birth name` (P1477) is filled for Marie Curie but not for Pierre Curie, so the `Entity IDs` field of the `birth name` property only contains Marie Curie.

Let us introduce some formal notations, useful to explain the computation of the input's embeddings (Section 4.3). The input is a tree where the root corresponds to the whole input, and each leaf contains the primitive values. For a non-leaf node x, we denote by $c(x)$ its children. For instance, in Figure 1, the node `Property 2` has three children (leaves) whose values are `mother`, `P25` and `Q37463`. A node x has also a type, and $T_{\leadsto}(x)$ denotes the types of all nodes on the path from the root to x. For instance, for the `mother` node, $T_{\leadsto}(x)$ is equal to `(root, property, name)`. In our setup, the depth of the input is at most 2.

ID randomization Directly giving the entity ID to the model would mean training a categorical classifier with millions of possible outcomes, which would lead to poor generalization. To avoid this, we replace the integer ID with a random one, thereby forcing the model to learn to predict the correct entity from the list of entities in input by copying their randomized entity ID to the output.

For numerical values, we associate each value to an arbitrary random ID, that the model should learn to copy in the output. For properties and classes, since there are fewer possibilities in the graph (a few thousand), we do not randomize them.

4.3 Embedding

Preprocessing We apply BERT tokenization (Devlin et al., 2019) to textual inputs. A vocabulary $\mathcal{V}_t$ is generated for each of the non-textual input types t.

Token embedding The goal of this step is to associate an embedding to each field of each object in the input. We do so by using a learned embedding for each input type: BERT embeddings (Devlin et al., 2019) for textual inputs, and categorical embeddings for non-textual inputs. When the input is a list (textual tokens or `Entity IDs` field), we add to this embedding a positional embedding. To reduce the size of the model, list embeddings are averaged into a single embedding. Formally, the embedding step associates a matrix of embeddings $h(x) \in \mathbb{R}^{1\times d_h}$ to each leaf of the input tree.

4.4 Encoding layers

There are two types of encoding layers: Flattener layers and Transformer layers.

Flattener The goal of these layers is to compute the embeddings of tree nodes bottom-up. They are successively applied until we are able to compute the embedding of the root node, i.e. of the whole input. This operation can be seen as flattening the JSON-like structure, hence their name.

Say we want to compute the embedding of some parent node x. An affine projection is first applied to the embedding of each child, then the embedding of the parent node is computed by applying a reduction operation $\mathcal{R}$, which can be either a sum or a concatenation. The weights of the projections are shared between all nodes having the same types $T_{\leadsto}(x)$. For example, all class name nodes - with types `(root, class, name)` - share the same weights, but they do not share the weights of entity name nodes - with types `(root, entity, name)`. Hence the embedding of x is

$$h(x) = \mathcal{R}_{y\in c(x)}\left(\left\{W^{T_{\leadsto}(y)}_{project}h(y) + b^{T_{\leadsto}(y)}_{project}\right\}\right).$$

If the reduction is a sum, all children embeddings need to be matrices of the same dimension, and the dimension of the parent embedding is also the same. If the reduction is a concatenation, the dimension of the parent embedding is $\left(\sum_{y\in c(x)} d(y), d_h\right)$.

Transformer This layer is a classical multi-head Transfomer encoder layer (Vaswani et al., 2017), taking as input a matrix of embeddings of dimensions (n, d_h), performing self-attention between the input embeddings, and outputting another matrix of the same dimensions. Detailed setup can be found in Appendix B.4.

Architecture We apply a first Transformer layer only to the utterances, and in parallel a first Flattener layer with sum reduction to all other inputs. The latter computes one embedding for each object. We add a positional embedding to each object, to account for its position in the list of objects of the same type. Then we apply a second Flattener layer with concatenation to all outputs of the first layer. This creates a single matrix of embeddings containing the embeddings of all the objects and utterances. Finally, a second Transformer layer is applied to this matrix.

4.5 Decoding layers

The output is a list of tokens, which correspond to a prefix representation of the LF tree. Note that the model architecture is grammar agnostic, as this output structure is independent of the grammar and we do not use grammar-guided decoding. The tokens can belong to one of the non textual input types or be a grammar token. Remember that we computed a vocabulary $\mathcal{V}_t$ for all token types $t \in \mathcal{T}$. We augment the vocabulary with a STOP token.

The decoder predicts the output list of tokens iteratively. Assume that we are given the first j tokens $y_1, \ldots, y_j$. We then apply an auto-regressive Transformer decoder on the full sequence, and we condition on the last embedding h_j of the sequence to predict the next token $\hat{y}_{j+1}$. Several categorical classifiers are used to predict $\hat{y}_{j+1}$. We first decide whether we should stop decoding:

$$\hat{s}_{j+1} = \operatorname{argmax} p_{stop,j} \tag{1}$$

where $p_{stop,j} = \operatorname{softmax}(W_{stop}h_j)$ is a distribution over $\{0, 1\}$ given by a binary classifier. If $\hat{s}_{j+1} = 1$, the decoding is finished, and we set $\hat{y}_{j+1}$ to STOP; otherwise, we predict the type of the token:

$$\hat{t}_{j+1} = \operatorname{argmax} p_{type,j} \tag{2}$$

where $p_{type,j} = \operatorname{softmax}(W_{type}h_j)$ is a distribution over $\mathcal{T}$ given by a $|\mathcal{T}|$-class classifier.

Finally, depending on the predicted type, we predict the token itself

$$\hat{y}_{j+1} = \operatorname{argmax} p^{\hat{t}_{j+1}}_{token,j} \tag{3}$$

where $p^{\hat{t}_{j+1}}_{token,j} = \operatorname{softmax}(W^{\hat{t}_{j+1}}_{token}h_j)$ is a distribution over $\mathcal{V}_{\hat{t}_{j+1}}$ given by a $\left|\mathcal{V}_{\hat{t}_{j+1}}\right|$-class classifier.

Training We train by teacher forcing: for some training sample $(\mathbf{x}, [y_1, \cdots, y_M])$ and for each step j, the embedding h_j is computed using the expected output at previous steps: $h_j = h(y_j; \mathbf{x}, y_1, \cdots, y_{j-1})$. The loss is the cross-entropy between the expected output y_{j+1} and the distributions produced by the model. Precisely, let

$T : \bigcup_{t \in \mathcal{T}} \mathcal{V}_t \to \mathcal{T}$ be the mapping which projects tokens to their type. We denote by $p(x)$ the value of a categorical distribution p for the category x. Then, omitting the step subscripts j, the loss equals

$$l(p, y) = -\log(p_{stop}(0)) \\ - \left[\log(p_{type}(T(y))) + \log(p_{token}^{T(y)}(y))\right]$$

for all steps except the last, and $l(p, y) = -\log(p_{stop}(1))$ for the last step. p_{stop}, p_{type}, and p_{token} are computed as explained above. The total loss is obtained by averaging over all training samples and over all steps.

5 Experiments

Additional comments about the datasets, setups, and additional results can be found in the appendix.

5.1 Datasets

We use two weakly supervised conversational QA datasets to evaluate our method, Complex Sequential Question Answering (CSQA)[1] (Saha et al., 2018) and ConvQuestions[2] (Christmann et al., 2019), both grounded in Wikidata[3]. CSQA consists of about 1.6M turns in 200k conversations (152k/16k/28k splits), versus 56k turns in 11k conversations (7k/2k/2k splits) for ConvQuestions.

CSQA was created by asking crowd-workers to write turns following some predefined patterns, then turns were stitched together into conversations. The questions are organized in different categories, e.g. simple, logical or comparative questions.

For ConvQuestions, crowd-workers wrote a 5-turn dialog in a predefined domain (*e.g.* Books or Movies). The dialogs are more realistic than in CSQA, however at the cost of a smaller dataset.

As presented in Table 2, the datasets have different characteristics, which make them an interesting test bed to assess the generality of our approach.

5.2 CSQA Experimental Setup

Metrics To evaluate our grammar, we report the coverage, i.e. the percentage of training questions for which we found a candidate Logical Form.

To evaluate the QA capabilities, we use the same metrics as in Saha et al. (2018). F1 Score is used for questions whose answers are entities, while accuracy is used for questions whose answer is boolean or numerical. We don't report results for "Clarification" questions, as this question type can be accurately modeled with a simple classification task, as reported in Appendix A. Similarly the average metric "Overall" (as defined in Saha et al. 2018) is not reported in Table 3, as it depends on "Clarification", but can be found in the Appendix.

Baselines We compare our results with several baselines introduced in Section 2: D2A (Guo et al., 2018), D2A+ES (Shen et al., 2020), S2A+MAML (Guo et al., 2019a), and MaSP (Shen et al., 2019).

5.3 ConvQuestions Experimental Setup

Metrics We use the coverage as above, and the P@1 metric as defined in Christmann et al. (2019).

Baseline The only baseline to our knowledge is CONVEX (Christmann et al., 2019), which casts the problem to a node classification task. For comparison, we tried to make our setup as close as possible to theirs, and refer to Appendix C.3 for details.

Data augmentation Given the small size of the dataset, we merge it with two other data sources: CSQA, and 3.6M examples generated by random sampling. The latter are single-turn dialogs made from graph triplets, *e.g.* the triplet *(Marie Curie, instance of, human)* generates the dialog: *Q: Marie Curie instance of? A: Human*. More details are given in Appendix B.3. The ConvQuestions dataset is upsampled to match the other data sources sizes.

5.4 Named Entity Linking setup

We tried to use a similar setup as baselines for fair comparison. For CSQA, the previous gold answer is given to the model in an oracle-like mode, as in baselines. In addition, we use simple string matching between utterances and names of Wikidata entities to retrieve candidates that are given in input to the model. For ConvQuestions, we use the gold seed entity (as in the CONVEX baseline we compare with), and the Google Cloud NLP service. We refer to Appendices B.2 and B.4 for details.

Regarding CARTON (Plepi et al., 2021), their results are not directly comparable as their model uses gold entity annotations as input and hence is not affected by NEL errors. This different NEL setup does have a strong influence on the performance, as running our model on CSQA with a setup similar to CARTON improves our Total Average score by over 10%. We refer to Appendix D.2 for details. More generally, a more thorough study

[1]https://amritasaha1812.github.io/CSQA
[2]https://convex.mpi-inf.mpg.de
[3]https://www.wikidata.org

	CSQA	**ConvQuestions**
Average length of a dialog	8 turns	5 turns
Possible change of topic inside a conversation	Yes	No
Answer type	Entities, boolean, quantity	Entities (usually a single one), boolean, date, quantity, string
Entity annotations in the dataset	Yes, with coreference resolution	Only the *seed entity* (topic of the dialog) and the answer entities
Coreferences in questions	Yes, to the previous turn	Yes, to any preceding turn

Table 2: Some characteristics of the benchmark datasets.

Methods		**D2A**	**D2A+ES**	**S2A+MAML**	**MaSP**	**OAT (Ours)**
Question type	**# Examples**			**F1**		
Simple (Direct)	82k	91.41	83.00	**92.66**	85.18	82.69
Simple (Coreferenced)	55k	69.83	64.62	71.18	76.47	**79.23**
Simple (Ellipsis)	10k	81.98	83.94	82.21	83.73	**84.44**
Logical	22k	43.62	72.93	44.34	69.04	**81.57**
Quantitative	9k	50.25	63.95	50.30	73.75	**74.83**
Comparative	15k	44.20	55.05	48.13	68.90	**70.76**
Question type	**# Examples**			**Accuracy**		
Verification (Boolean)	27k	45.05	45.80	50.16	60.63	**66.39**
Quantitative (Count)	23k	40.94	41.35	46.43	43.39	**71.79**
Comparative (Count)	15k	17.78	20.93	18.91	22.26	**36.00**
Total Average	260k	64.47	64.75	66.54	70.56	**75.57**

Table 3: QA performance on CSQA. The metric is the F1 score for question types above the vertical separator, and accuracy for those under. The Total Average score is an average over all question types.

of the impact of the NEL step on the end-to-end performance would be an interesting direction of future work (see also Section 5.6).

5.5 Results

Our grammar reaches high coverage. With approximately the same numbers of operators as in baselines, we improve the CSQA coverage by 16%, as presented in Table 4. The improvement is particularly important for the most complex questions. We reach a coverage of 86.2% on ConvQuestions, whose questions are more varied than in CSQA.

Most queries can be expressed as relatively shallow LFs in our grammar, as illustrated by Table 5. This is especially interesting for the ConvQuestions dataset, composed of more realistic dialogs. On CSQA, the average depth of our LFs (2.9) is slightly slower than with D2A grammar (3.2).

We improve the QA performance over baseline on both datasets. For CSQA, our model outper-

Question type	**D2A**	**D2A+ES**	**Ours**
Comparative	28.6	45	**84.9**
Logical	48.2	92	**100.0**
Quantitative	58.1	62	**91.1**
Simple	94.4	96	**99.7**
Verification	77.9	85	**91.4**
Overall	74.3	80	**96.2**

Table 4: Coverage per question type for CSQA.

Depth	**1**	**2**	**3**	**4+**
CSQA (D2A)	0.0	47.0	30.9	22.0
CSQA (Ours)	5.5	67.9	7.4	19.2
ConvQuestions	53.9	43.7	2.4	0.0

Table 5: Silver LF depth distribution for both datasets.

Domain	1st turn	Follow-up	CONVEX
Books	68.1	**20.9**	19.8
Movies	54.2	**31.3**	25.9
Music	37.5	18.1	**19.0**
Soccer	43.8	**22.8**	18.8
TV	66.3	**31.8**	17.8
Overall	54.0	**25.0**	20.3

Table 6: ConvQuestions results by domain. The first two columns are our results. The baseline (Oracle+CONVEX) only reports follow-up turns.

forms baselines for all question types but Direct Simple questions, as shown in Table 3. Overall, our model improves the performance by 5%. For ConvQuestions, Table 6 shows that our model improves over the baseline for all domains but one, yielding an overall improvement of 4.7%. A precise evaluation of the impact of the various components of our KG-QA approach (grammar, entity linking, model inputs, model architecture, size of the training data, etc.) on the end-to-end performance was out of the scope of this paper, and is left for future work. Nevertheless, the fact that we are able to improve over baselines for two types of Simple questions and for Logical questions, for which the grammar does not matter so much, as these question types correspond to relatively shallow LFs, suggests that our proposed model architecture is effective.

5.6 Error analysis

CSQA By comparing the silver and the predicted LFs on 10k random errors, we could split the errors in two main categories: first, the LF general form could be off, meaning that the model did not pick up the user intent. Or the form of the LF could be right, but (at least) one of the tokens is wrong. Table 7 details the error statistics. The most frequent errors concern entity disambiguation. There are two types of errors: either the correct entity was not part of the model input, due to insufficient recall of the NEL system. Or the model picked the wrong entity from the input due to insufficient precision. It is known that the noise from NEL strongly affects model performance (Shen et al., 2019). We tried an oracle experiment with perfect recall NEL (see Appendix D.2), which corroborates this observation, in particular for Simple questions. As we focused on modeling complex questions, improving NEL was not our main focus, but would an interesting direction for future work, in particular via multi-task approaches (Shen et al., 2019).

ConvQuestions We manually analyzed 100 examples. Errors were mostly due to the LF general form, then to a wrong property token.

The model learns the grammar rules. In all inspected cases, the predicted LF is a valid LF according to the grammar, i.e. it could be evaluated successfully. This shows that grammar-guided decoding is not needed to achieve high performance.

Error category	Overall	Simple Dir.
LF general form	31.8	24.1
Entity ID token	36.2	38.2
insuff. recall	17.1	16.8
insuff. precision	19.6	21.6
Property ID token	4.2	2.9
Class ID token	24.7	37.6
Grammar token	11.6	2.6

Table 7: Distribution of errors in CSQA. The numbers are (non exclusive) percentages.We also report statistics for the Simple Direct type, as it is the largest.

6 Conclusion

For the problem of weakly-supervised conversational KG-QA, we proposed Object-Aware Transformer, a model capable of processing structured input in a JSON-like format. This allows to flexibly provide the model with structured KG contextual information. We also introduced a KG grammar with increased coverage, which can hence be used to model a wider range of queries. These two contributions are fairly independent: on the one hand, since the model predicts LFs as a list of tokens, it is grammar agnostic, and thus it could be used with another grammar. On the other hand, the grammar is not tied to the model, and can be used to generate training data for other model architectures. Experiments on two datasets validate our approach. We plan to extend our model with richer KG context, as we believe there is headroom for improvements.

Acknowledgments

We thank Yasemin Altun, Eloïse Berthier, Guillaume Dalle, Maxime Godin, Clément Mantoux, Massimo Nicosia, Slav Petrov, and anonymous reviewers, for their constructive feedback, and Daya Guo for providing results with the D2A grammar. P.M. was supported by a Corps des Mines stipend.

Ethical considerations

This work is not connected to any specific real-world application, and solely makes use of publicly available data (KG and QA datasets). The predominant ethical concern of the paper is the computing power associated with the experiments. To limit the energy impact of the project, we did not perform hyper-parameter tuning for model training. Using the formulas from Patterson et al. (2021), we estimate the GHG emissions associated with one run of model training to be approximately 29 - 42 kg of CO2e. For the silver LF generation, we iterated on a small subset of the datasets, then computed the LFs for the entire datasets as a one-off task.

References

M. Ahmed, M. R. Samee, and R. E. Mercer. 2019. Improving Tree-LSTM with Tree Attention. In *2019 IEEE 13th International Conference on Semantic Computing (ICSC)*, pages 247–254.

Ram G Athreya, Srividya K. Bansal, Axel-Cyrille Ngonga Ngomo, and Ricardo Usbeck. 2021. Template-based question answering using recursive neural networks. In *2021 IEEE 15th International Conference on Semantic Computing (ICSC)*, pages 195–198.

Ruisheng Cao, Su Zhu, Chenyu Yang, Chen Liu, Rao Ma, Yanbin Zhao, Lu Chen, and Kai Yu. 2020. Unsupervised dual paraphrasing for two-stage semantic parsing. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6806–6817, Online. Association for Computational Linguistics.

Xinyun Chen, Chang Liu, and Dawn Song. 2018. Tree-to-tree neural networks for program translation. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 2552–2562.

Philipp Christmann, Rishiraj Saha Roy, Abdalghani Abujabal, Jyotsna Singh, and Gerhard Weikum. 2019. Look before you hop: Conversational question answering over knowledge graphs using judicious context expansion. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management, CIKM 2019, Beijing, China, November 3-7, 2019*, pages 729–738. ACM.

William W. Cohen, Matthew Siegler, and Alex Hofer. 2019. Neural Query Language: A Knowledge Base Query Language for Tensorflow. *Computing Research Repository*, arXiv:1905.06209.

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

Li Dong and Mirella Lapata. 2016. Language to logical form with neural attention. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 33–43, Berlin, Germany. Association for Computational Linguistics.

Li Dong and Mirella Lapata. 2018. Coarse-to-fine decoding for neural semantic parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 731–742, Melbourne, Australia. Association for Computational Linguistics.

Daya Guo, Duyu Tang, Nan Duan, Ming Zhou, and Jian Yin. 2018. Dialog-to-action: Conversational question answering over a large-scale knowledge base. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 2946–2955.

Daya Guo, Duyu Tang, Nan Duan, Ming Zhou, and Jian Yin. 2019a. Coupling retrieval and meta-learning for context-dependent semantic parsing. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 855–866, Florence, Italy. Association for Computational Linguistics.

Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. 2019b. Towards complex text-to-SQL in cross-domain database with intermediate representation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4524–4535, Florence, Italy. Association for Computational Linguistics.

William L. Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 1024–1034.

Jacob Harer, Chris Reale, and Peter Chin. 2019. Tree-Transformer: A Transformer-Based Method for Correction of Tree-Structured Data. *Computing Research Repository*, arXiv:1908.00449.

Jonathan Herzig, Pawel Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Eisenschlos. 2020. TaPas: Weakly supervised table parsing via pre-training. In *Proceedings of the 58th Annual*

Meeting of the Association for Computational Linguistics, pages 4320–4333, Online. Association for Computational Linguistics.

Wonseok Hwang, Jinyeong Yim, Seunghyun Park, and Minjoon Seo. 2019. A Comprehensive Exploration on WikiSQL with Table-Aware Word Contextualization. *Computing Research Repository*, arXiv:1902.01069. Version 2. Presented at KR2ML Workshop at NeurIPS 2019.

Robin Jia and Percy Liang. 2016. Data recombination for neural semantic parsing. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12–22, Berlin, Germany. Association for Computational Linguistics.

Endri Kacupaj, Joan Plepi, Kuldeep Singh, Harsh Thakkar, Jens Lehmann, and Maria Maleshkova. 2021. Conversational question answering over knowledge graphs with transformer and graph attention networks. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 850–862, Online. Association for Computational Linguistics.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Thomas N. Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

Shucheng Li, Lingfei Wu, Shiwei Feng, Fangli Xu, Fengyuan Xu, and Sheng Zhong. 2020. Graph-to-tree neural networks for learning structured input-output translation with applications to semantic parsing and math word problem. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 2841–2852, Online. Association for Computational Linguistics.

Chen Liang, Jonathan Berant, Quoc Le, Kenneth D. Forbus, and Ni Lao. 2017. Neural symbolic machines: Learning semantic parsers on Freebase with weak supervision. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 23–33, Vancouver, Canada. Association for Computational Linguistics.

David Patterson, Joseph Gonzalez, Quoc Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David So, Maud Texier, and Jeff Dean. 2021. Carbon emissions and large neural network training. *Computing Research Repository*, arXiv:2104.10350.

Joan Plepi, Endri Kacupaj, Kuldeep Singh, Harsh Thakkar, and Jens Lehmann. 2021. Context transformer with stacked pointer networks for conversational question answering over knowledge graphs. In *The Semantic Web*, pages 356–371, Cham. Springer International Publishing.

Amrita Saha, Ghulam Ahmed Ansari, Abhishek Laddha, Karthik Sankaranarayanan, and Soumen Chakrabarti. 2019. Complex program induction for querying knowledge bases in the absence of gold programs. *Transactions of the Association for Computational Linguistics*, 7:185–200.

Amrita Saha, Vardaan Pahuja, Mitesh M. Khapra, Karthik Sankaranarayanan, and Sarath Chandar. 2018. Complex sequential question answering: Towards learning to converse over linked question answer pairs with a knowledge graph. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 705–713. AAAI Press.

Michael Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. 2018. Modeling Relational Data with Graph Convolutional Networks. In *The Semantic Web, Proceedings of the 15th International Conference, ESWC 2018*, pages 593–607, Heraklion, Crete, Greece. Springer, Cham.

Peter Shaw, Philip Massey, Angelica Chen, Francesco Piccinno, and Yasemin Altun. 2019. Generating logical forms from graph representations of text and entities. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 95–106, Florence, Italy. Association for Computational Linguistics.

Tao Shen, Xiubo Geng, Guodong Long, Jing Jiang, Chengqi Zhang, and Daxin Jiang. 2020. Effective search of logical forms for weakly supervised knowledge-based question answering. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, pages 2227–2233. ijcai.org.

Tao Shen, Xiubo Geng, Tao Qin, Daya Guo, Duyu Tang, Nan Duan, Guodong Long, and Daxin Jiang. 2019. Multi-task learning for conversational question answering over a large-scale knowledge base. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2442–2451, Hong Kong, China. Association for Computational Linguistics.

Vighnesh Leonardo Shiv and Chris Quirk. 2019. Novel positional encodings to enable tree-based transformers. In *Advances in Neural Information Processing*

Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada, pages 12058–12068.

Kai Sheng Tai, Richard Socher, and Christopher D. Manning. 2015. Improved semantic representations from tree-structured long short-term memory networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1556–1566, Beijing, China. Association for Computational Linguistics.

Raghuveer Thirukovalluru, Mukund Sridhar, Dung Thai, Shruti Chanumolu, Nicholas Monath, Sankaranarayanan Ananthakrishnan, and Andrew McCallum. 2021. Knowledge informed semantic parsing for conversational question answering. In *Proceedings of the 6th Workshop on Representation Learning for NLP (RepL4NLP-2021)*, pages 231–240, Online. Association for Computational Linguistics.

Peihao Tong, Qifan Zhang, and Junjie Yao. 2019. Leveraging Domain Context for Question Answering Over Knowledge Graph. *Data Science and Engineering*, 4(4):323–335.

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 *Proceedings of the 28th ACM International Conference on Information and Knowledge Management, CIKM 2019, Beijing, China, November 3-7, 2019*, pages 1431–1440. ACM.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008.

William Woof and Ke Chen. 2020. A Framework for End-to-End Learning on Semantic Tree-Structured Data. *Computing Research Repository*, arXiv:2002.05707.

Kun Xu, Lingfei Wu, Zhiguo Wang, Mo Yu, Liwei Chen, and Vadim Sheinin. 2018. Exploiting rich syntactic information for semantic parsing with graph-to-sequence model. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 918–924, Brussels, Belgium. Association for Computational Linguistics.

Hamid Zafar, Giulio Napolitano, and Jens Lehmann. 2019. Deep Query Ranking for Question Answering over Knowledge Bases. In *Machine Learning and Knowledge Discovery in Databases, Proceedings of the European Conference, ECML PKDD 2018*, Lecture Notes in Computer Science, pages 635–638, Dublin, Ireland. Springer International Publishing.

Jiawei Zhang, Haopeng Zhang, Congying Xia, and Li Sun. 2020. Graph-Bert: Only Attention is Needed for Learning Graph Representations. *Computing Research Repository*, arXiv:2001.05140. Version 2.

A Clarification Questions in CSQA

Take the following dialog as example:

T1	*Can you tell me which cities border Verderio Inferiore?* Cornate d'Adda, Bernareggio, Robbiate
T2	*And which cities flank that one?* Did you mean Robbiate?
T3	*No, I meant Cornate d'Adda.* Bottanuco, Busnago, Trezzo sull'Adda

The second turn is a "Clarification" question: the system asks the user for disambiguation. The disambiguation question usually takes the form "Did you mean", followed by an entity chosen among the previous turn answers. This choice appears to be entirely random. For this reason, we found that it would not be very interesting to try to predict this entity, as baselines propose. Hence we only ask the model to predict that the question is a Clarification (via a special `clarification` operator).

We report in Table 8 the scores for Clarification questions, as well as the "Overall" score, as defined in Saha et al. (2018). The results are not directly comparable as the baseline systems report an F1 score, while our approach uses accuracy.

B Detailed experimental setup

B.1 Meta-operators

Take the example given in the main paper: *"Which musical instrument is played by the maximum number of persons?"*. The corresponding LF is:

```
argmax(
  cardinality(
    follow_property(
      for_each(
        members(musical instrument)),
      played by)))
```

Assume that the KG contains exactly two musical instruments, piano and violin, i.e. `members(musical instrument)` equals `{piano, violin}`.

`for_each` creates a dictionary of entities. Each (key, value) pair corresponds to one entity in the argument of `for_each`, where the key is the entity itself and the value is a singleton set

Question type	# Examples	D2A	D2A+ES	S2A+MAML	MaSP	OAT (Ours)
Clarification	12k	18.31	36.66	19.12	80.79	**99.63**
Overall	206k	62.88	72.02	N/R	79.26	**81.49**

Table 8: QA performance on CSQA, including "Clarification" questions. The "Overall" metric is the average F1 scores of the following question types: "Simple (Direct)", "Simple (Coreferenced)", "Simple (Ellipsis)", "Logical", "Quantitative", "Comparative" and "Clarification".

containing the entity. Here `for_each({`*piano, violin*`})` gives the following dictionary:

```
{ piano: {piano}, violin: {violin} }
```

We then apply the same computation to each of the dictionary values, while keeping the keys untouched. In our example, we apply the expression

```
cardinality(
  follow_backward(., played by)
),
```

which gives the result

```
{piano: 20392, violin: 7918}.
```

Finally, an aggregation operator is computed over the values, and the result is a subset of the keys. `argmax` returns the set of keys associated with the maximum values, in our example `{piano}`. In other cases, we want to return all the keys associated to a non-empty value, `arg` allows to do so.

B.2 Silver LF generation

Wikidata version For CSQA, we used the preprocessed version of Wikidata made available by the authors, which contains 21.2M triplets over 12.8M entities and 567 distinct properties. For ConvQuestions, we used a more recent version of Wikidata, containing 1.1B triplets over 91.8M entities and 7869 distinct properties.

Named Entity Linking For ConvQuestions, we use gold entity annotations and Google Cloud NLP entity linking service. For CSQA, we use gold entity annotations.

To resolve the coreferences, in ConvQuestions, we use entity annotations from previous utterances during the silver LF generation step. In CSQA, since coreferences are already resolved by the gold annotations, we just use annotations from the current utterance.

Simplifying the BFS We observed that reaching a depth of 4+ is needed for some queries (see Table 5 of the main paper), but is impractical by exhaustive BFS, as the size of the space of LFs grows very quickly with their depth. To improve the efficiency, we used the following ideas:

- **Stopping criteria to abort the exploration:** timeout $t_{\max}$ and maximum depth $d_{\max}$.
- **Type checking:** by leveraging the operators' signatures (presented in Table 1 of the main paper), we only construct legal LFs.
- **Putting constraints on the form of the LF:** we manually forbid certain combinations of operators, e.g. `follow_backward` after `follow_property`.
- **Restriction of the list of operators:** for ConvQuestions, we use the graph operators, the numerical operators, `is_in`, and `get_first`. The removal of some set operators and of meta-operators strongly reduces the complexity of the BFS. For CSQA, all operators are needed, but we add more constraints to keep the BFS simple enough.

We choose $d_{\max} = 3$ for ConvQuestions and $d_{\max} = 7$ for CSQA, and $t_{\max} = 1200$ seconds.

All LFs found by BFS are evaluated over the KG, which gives candidate answers. We keep the LFs whose candidate answers have the highest F1 score w.r.t. the gold answer. The minimal F1 score for keeping a LF is 0.3.

Scores for LF ranking The BFS often returns several LFs (with the top F1 score, as explained above), among which some are spurious: they do not correspond to the semantic meaning of the question, but their evaluation over the KG yields the correct result by chance. As we keep only one for training, we need a way to rank the candidate LFs. We use the following heuristic scores to do so:

- **Complexity:** the score is $1 - {}^{(d-1)}\!/_{(d_{\max}-1)}$ where d is the depth of the LF and $d_{\max}$ is defined above.
- **Property lexical matching:** for each property appearing in the LF, we compute the Jaccard index of the words appearing in its name and of the words of the question.

- **Annotation coverage:** among the entities retrieved by NEL, we compute the percentage of entities which appear in the LF.

As these three scores are between 0 and 1, we average them and keep the LF with the highest total score. We found that this simple method is a good way to reduce spurious LFs, which are often either too complex or not matching lexically the question.

B.3 Random examples generation

To generate the random examples for data augmentation for ConvQuestions training, we first sample uniformly 80k entities from the graph. Then, for each entity, we generate a conversation for each triplet that links it to other entities. The question text is made by stitching the entity name and the property name. For instance, the triplet `(Marie Curie, native language, Polish)` generates the dialog: *Q: Marie Curie native language? A: Polish.* We also generate variants where the property name is replaced by aliases, which are alternative names in Wikidata, for example *mother tongue* for `native language`. When the question or answer has more than 256 characters, we eliminate it.

B.4 Modeling

Wikidata version As in B.2.

Named Entity Linking NEL is performed again, this time to create the structured context in input to the model. Due to the randomization step described in Section 4.2, missing entities in the input cannot be retrieved by the model, so we want to have a high NEL recall. The trade-off is that the NEL precision is low: we have many spurious entities in the input, which the model has to learn to ignore.

For CSQA, we use simple string matching between the utterances and the names of Wikidata entities. Note that in our model, as well as in all CSQA baselines (in particular Guo et al., 2018; Shen et al., 2019), the previous gold answer is given as input to the model in an oracle-like setup.

For ConvQuestions, we use the gold seed entity and the Google Cloud NLP entity linking service.

To resolve coreferences, we use entities from the dialog history: all preceding turns for ConvQuestions and only the previous turn for CSQA.

Implementation details We tokenize the input using the BERT-uncased tokenizer. All embeddings in the model have dimension 768. The two transformer encoders share the same configuration: 2 layers each, with output dimension 768, intermediate dimension 2048, 12 attention heads, and dropout probability set to 0.1. The model has 260M parameters. The transformer implementation is based on publicly available BERT code. We initialize the word embedding from a BERT checkpoint, but do *not* load the transformer layer weights, instead training them from scratch. We train for 600k steps with batch size 128, using the ADAM optimizer (Kingma and Ba, 2015) with learning rate 3×10^{-5}. Training takes around 14 hours on 16 TPU v3 with 32 cores.

C Comparison with baselines

C.1 Comparison with D2A grammar

The D2A (Guo et al., 2018) grammar is the main baseline in previous KG-QA works. We compare with the grammar implemented in their open-sourced code[4], which is a bit different from the published one. Numbers for D2A in Tables 4 and 5 were computed thanks to the results of the BFS gracefully provided by the authors.

Table 9 presents some intents which we are able to model in our grammar and are not straightforward to model with D2A grammar. First, *textual form reasoning* corresponds to questions about string attributes of entities, which are not included in the D2A grammar. Second, to handle *numerical and temporal reasoning*, computations based on numerical values are needed, which is not possible with the D2A grammar. Finally, the D2A grammar does not model the order of relations in the graph and the selection of class members, which we start to tackle with respectively the `get_first` and `members` operators.

C.2 Comparison with MaSP architecture

MaSP (Shen et al., 2019) also follows the semantic parsing approach, where the LF is encoded as a sequence of operators and graph items IDs. Regarding the model input, they only use the utterances, whereas we add additional KG context structured as a JSON tree. The training method differs: MaSP uses multi-task learning to learn jointly entity linking and semantic parsing, whereas we chain both, and trust the model to pick the good entity. Our approach is simpler in this regard, but we pay it by

[4]`https://github.com/guoday/Dialog-to-Action/blob/bb2cbb9de474c0633bac6d01c10eca24c79b951f/BFS/parser.py`

Intent	Question example	Missing in D2A
Textual form reasoning	What was Elvis Presley given name?	string type
Numerical reasoning	What actor plays the younger child?	`get_value`, `for_each`, `argmin`
Numerical reasoning	How old is the younger child?	`min`
Selection of the members of a class	Which television programs have been dubbed by at least 20 people ?	`members`
Temporal reasoning	What is the number of seasons until 2018?	`for_each`, `get_value`, `lesser_than`, `arg`
Ordinal reasoning	What was the first episode date?	`get_first`

Table 9: Examples of questions that are difficult to model with the D2A grammar. Examples are mostly chosen from ConvQuestions, as their questions look more realistic than CSQA.

having a slightly lower performance on Simple Direct questions (see Table 3). Finally, we do not use beam search for the decoding, contrarily to them.

C.3 Comparison with node classification

An alternative to the semantic parsing approach is to train classifiers to predict entities as nodes of the KG. A precise comparison of both approaches is out of the scope of this paper. Nevertheless, we think that the semantic parsing approach is better suited to our purpose of modeling complex questions. For instance, complex intents involving numerical comparisons can be expressed naturally by a LF, but would be difficult to perform using solely node classifiers. Examples include the numerical and temporal reasoning in Table 9. Additional examples include *The series consists of which amount of books?* (ConvQuestions) or *Which television programs have been dubbed by at least 20 people ?, How many episodes is it longer than the second longest season out of the three?* (CSQA).

CONVEX (Christmann et al., 2019) is an example of such an approach. It is an unsupervised graph exploration method: at each turn, a subgraph is expanded by matching the utterance with neighboring entities. Then a candidate answer is found in the subgraph by a node classifier. On our side, we propose a semantic parsing approach that makes use of entities annotated by an external entity linking service. This is a similar setup to the CSQA baselines (Shen et al., 2019), which we re-purposed for ConvQuestions in order to assess the quality of our proposal on another dataset. In order to be closer to the CONVEX baseline, we changed our CSQA setup by applying the entity linker only to the questions' text and not to the answers' text. In addition, as we use the gold seed entity, we compare with the Oracle+CONVEX setup of Christmann et al. (2019), which also uses the gold seed entity (and the gold first turn answer entity). Finally, we make use of data augmentation to train our model on ConvQuestions, whereas the baseline does not.

Domain	Coverage
Books	88.6
Movies	87.6
Music	90.0
Soccer	78.6
TV	86.2
Overall	86.2

Table 10: ConvQuestions coverage per domain.

C.4 BERT or no BERT, that is the question

The baselines of Table 3 do not use BERT. MaSP authors provide a variant of their model using a fine-tuned BERT base architecture. The Total Average score of this variant is 72.60%, which is 2% above their vanilla variant and 3% under our model. Since we are only using the word embeddings (loaded from a publicly available BERT base checkpoint) and not loading the transformer layer weights, we decided to compare with the vanilla variant of MaSP. Finally, CARTON is using a pretrained BERT base model as a sentence encoder.

D Additional results

D.1 Coverage and performance results

Table 10 shows the coverage per domain for ConvQuestions. Regarding the evolution of the coverage over turns, it is stable for both datasets, hence we do not report this result.

Tables 11 and 12 show the evolution of the performance over turns for both datasets. For CSQA, the performance drops after the first two turns, then remains constant. For ConvQuestions, the performance decreases throughout the turns. There is a sharp decrease after the first turn, probably because it is simpler as there is no coreference or ellipsis. The different behavior between the datasets may be due to the realism of ConvQuestions.

Turns	**0**	**1**	**2**	**3**	**4**
Score	85	87	75	74	74

Turns	**5-6**	**7-8**	**9-10**	**11-12**	**13+**
Score	75	75	75	74	74

Table 11: Average performance over turns for CSQA. For brevity, we average over turn ranges after turn 5.

Turn	**0**	**1**	**2**	**3**	**4**
Av. P@1	54	35	20	29	15

Table 12: Performance over turns for ConvQuestions.

D.2 Oracle setup and CARTON comparison

CARTON (Plepi et al., 2021) gives the entities annotated in the dataset as part of the model input (entities appearing in the previous turn and in the current question), contrarily to the models in Table 3 which all use an entity linker. For a fair comparison, we tested our model in an oracle setup, where we also give the gold annotations as input. As shown in Table 13, the Total Average score of our model increases by 10% w.r.t. the baseline approach. The improvement is particularly important for the most simple question types (Simple and Logical Questions). In this setup, our performance is 8% higher than CARTON, and we obtain a better score for 7 out of 10 question types.

D.3 Further error analysis

An alternative approach for error analysis is to assess the performance of the decoding classifiers (see Section 4.5) in a teacher forcing setup, *i.e.* to assess how often they predict the next token correctly, given the true previous tokens. Table 14 reports the results on the eval split of both datasets.

Question type	**CARTON**	**Ours**
Simple (Direct)	85.92	**96.95**
Simple (Coreferenced)	87.09	**94.77**
Simple (Ellipsis)	85.07	**96.66**
Logical	80.80	**95.54**
Quantitative	**80.62**	76.44
Comparative	62.00	**76.66**
Verification (Boolean)	**77.82**	67.02
Quantitative (Count)	57.04	**75.89**
Comparative (Count)	**38.31**	35.10
Total Average	77.89	**85.85**

Table 13: QA performance on CSQA in oracle mode.

Metric	**CSQA**	**ConvQuestions**
Token type	99.88	94.77
Grammar token	98.86	82.12
Entity ID	92.47	50.79
Property ID	99.45	30.26
Class ID	94.64	N/A
Numerical value	99.91	N/A
Avg. token	97.70	61.53

Table 14: LF token accuracy metrics, on the eval splits. For ConvQuestions, Class ID, numerical value and their relative operators are not used (see B.2).

The results corroborate the analysis presented in Section 5.6. First, the model learns the grammar rules, as it nearly always predicts the good token type. For CSQA, the most frequent errors concern entity ID and class ID. For ConvQuestions, they concern primarily entities and properties.

D.4 Case study

Table 15 presents examples from ConvQuestions where we are able to predict the good LFs, although there exists very similar properties in the graph. The textual forms of the questions are not sufficient to infer the good property to use, implying that the model had to learn elements from the graph structure in order to answer correctly these questions. Nevertheless, Table 14 shows that there is still significant room for improvement in that direction.

Question	**Property**
When did Seinfeld first air?	start time (P580)
When did Camp Rock come out?	publication date (P577)
Who screen wrote it?	screenwriter (P58)
Who wrote it?	author (P50)
What country are they from?	country (P17)
Belleville of which country?	country (P17)
What country did the band Black Sabbath originally come from?	country of origin (P945)
What country is Son Heung-min from originally?	country for sport (P1532)

Table 15: Examples of ConvQuestions questions for which the model was able to pick up the good property, although there are very similar properties in the graph.