OverpassNL: A Community-Generated Dataset and Real-World Semantic Parser for OpenStreetMap

Anonymous ACL submission

Abstract

We present OverpassNL, a complex dataset that pairs queries to the OpenStreetMap (OSM) database with natural language questions. It is based on nearly 10,000 queries issued by OSM users and developers in the Overpass query language. The Overpass queries were translated into suitable natural language forms by 15 trained computational linguistics students. The resulting dataset can be used as training data for real-world semantic parsing. The complexity of OverpassNL stems from both the nature of real-world queries and the expansive underlying OSM database. While existing semantic parsing datasets such as Spider (Yu et al., 2018) use formulaic synthetic queries and achieve complexity by combining multiple simple underlying databases, there is no natural split into database schemata in OSM (Yu et al., 2018) nor does Overpass provide a clear structure for slot-filling (Yao et al., 2019). The complexity of the task is shown by the mere 21% execution accuracy achieved by a generic neural semantic parser. We enhance the model by using different types of additional information and by training data augmentation, thereby increasing the performance to 36% execution accuracy.

1 Introduction

Semantic Parsing allows the mapping of natural language queries into a corresponding structural form. This form can be a structured query language like SQL (Yu et al., 2018), a programming language like Bash (Lin et al., 2018), If-Then recipes (Quirk et al., 2015), or a NoSQL language like Overpass – a query language for the real-world, large-scale, and widely-used OpenStreetMap (OSM) database of geographic information. Most existing semantic parsing datasets face the constraint that they only use a single database with a small number of tables, thus limiting the number of possible queries. For example, GeoQuery (Zelle and Mooney, 1996; Iyer et al., 2017) contains eight tables, Restaurant (Tang and Mooney, 2000; Popescu et al., 2003) three tables, and IMDB (Yaghmazadeh et al., 2017) sixteen tables. Recent works such as Zhong et al. (2017) or Yu et al. (2018) try to alleviate this problem by combining multiple different databases found on the internet. The combined databases comprise different tasks, thus the meta-database consists of smaller, independent databases. Taking the respective database schemata into account during training and testing allows for a drastic reduction of the complexity of semantic parsing (Zhang et al., 2019). Such simplifications by schema information cannot be exploited for real-world semantic parsing of the OSM database. OSM consists of element types (nodes, ways, and relations), each with associated database information such as current, history, current_tags, and history_tags. It thus comprises 39 tables that are combined into one large and complex database instead of a meta-database that consists of several different and unrelated databases.

Moreover, our dataset OverpassNL is generated from complex real-world queries of users and developers using the Overpass API. In contrast, existing datasets such as Spider start from artificial formulaic questions from which queries are generated by computer science students. We created OverpassNL by letting computational linguistic students create natural language counterparts for real-world Overpass queries. We acquired those queries using Overpass Turbo, an online visualization tool for Overpass that exploits the full expressivity of the Overpass language. All annotators received training and went through a test to ensure high quality annotations. The resulting dataset consists of

1 https://wiki.openstreetmap.org/wiki/Overpass_API
2 https://www.openstreetmap.org. For statistics on usage and database, see Table 5 and Figure 4 in the Appendix.
3 https://overpass-turbo.eu/
nearly 10,000 Overpass queries, each accompanied
by a natural language question.

We use OverpassNL to train a semantic parser
that allows access to the OSM database via natural
language questions. A state-of-the-art sequence-to-
sequence model trained on our dataset achieves
21% execution accuracy, showing that semantic
parsing of our dataset is indeed a challenging task.
Since we cannot take advantage of additional infor-
mation like the database schema, we increase the
performance of our model by 1) retrieving similar
elements from the training data as additional in-
puts, 2) clustering the data and augmenting them
with the resulting cluster information, 3) extracting
key-value pairs by fuzzy matching the natural
language questions to an already existing knowl-
edge source and 4) creating a synthetic dataset for
further data augmentation. The best combination
of these data enhancement techniques achieves an
significant increase in execution accuracy to 36%.

2 Related Work

Text-to-SQL parsing has been popular since the
1990s. Many different datasets have been created,
e.g., ATIS (Dahl et al., 1994; Iyer et al., 2017),
GeoQuery (Zelle and Mooney, 1996) and WikiSQL
(Zhong et al., 2017), each with their own shortcomings,
like only using a single database each. This problem
was addressed by Yu et al. (2018), who created a collection of databases pairing natural
language questions to SQL queries. This dataset
was later extended further into the contextual setting
by Yu et al. (2019b) and into the conversational
or interactive setting by Yu et al. (2019a).

Another semantic parsing task turns natural lan-
guage expressions into If-Then recipes (Quirk et al.,
2015) which connect actions (starting an alarm)
to triggers (specific time is reached). This task has
been extended into an interactive setting by Yao
et al. (2019). In the field of OpenStreetMap se-
matic parsing, preliminary work was already done
by Haas and Riezler (2016) who translated nat-
ural language queries into a self-designed Machine
Readable Language. We do not follow this ap-
proach since Overpass presents a more expressive
language that is used by the OSM community.

A crucial difference of our dataset to existing
semantic parsing data is that it consists of real-
world queries issued by users and developers trying
to satisfy a genuine information-seeking task by
executing a query against a large-scale database of
geographical information (see Section 3.2).

3 OverpassNL Dataset

3.1 Dataset Creation

We extracted all 150,000 queries that were logged
on the Overpass Turbo API with no pre-selection
procedures. We filtered out duplicates, which left
us with around 50,000 examples. A randomly se-
lected 10,000 of these were manually annotated.
The queries are therefore “standard” representative
user queries. We hired 15 computational linguistics
students for annotation of database queries with nat-
ural language questions. The annotators received a
tutorial, solved some training examples and com-
pleted a test to ensure they understood the task.
Then they were shown random examples of queries
and results using the annotation interface shown
in Figure 1. The task of the annotators was to cre-
ate natural language question corresponding to the
given Overpass query. This resulted in a dataset
of 9,609 paired question-parse pairs. We separated
those into train (7,109), dev (1,500) and test data
(1,000). An example of a query-question pair is as
follows:

question Ways with "name" tag containing values
"Power" or "power" edited by user with ID
2041564 in the Philippines
query [ out:json ]; ( (geocodeArea:
Philippines ) ) ->.searchArea: way[*
name="^.*-\^Power\^" (uid :
2041564) (area.searchArea);]; out
body;>out skel;

This approach to database creation has two main
advantages: First, teaching annotators to interpret

<table>
<thead>
<tr>
<th>Variable</th>
<th>ONL</th>
<th>Spider</th>
<th>WikiSQL</th>
<th>ATIS</th>
</tr>
</thead>
<tbody>
<tr>
<td>NL len.</td>
<td>9.35</td>
<td>12.09</td>
<td>12.27</td>
<td>10.47</td>
</tr>
<tr>
<td>Vocab. size</td>
<td>11,259</td>
<td>7,230</td>
<td>29,857</td>
<td>950</td>
</tr>
<tr>
<td>Query len.</td>
<td>203</td>
<td>116</td>
<td>57</td>
<td>1,020</td>
</tr>
<tr>
<td>String ops.</td>
<td>24%</td>
<td>2%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td># DBs</td>
<td>1</td>
<td>200</td>
<td>26,521</td>
<td>1</td>
</tr>
<tr>
<td># tables/DB</td>
<td>38</td>
<td>5.1</td>
<td>1</td>
<td>32</td>
</tr>
</tbody>
</table>

Table 1: Analysis of the complexity in OverpassNL
(ONL) versus other datasets. Natural language question
length (NL len.) and query length (Query len.) are
averaged. String ops. refers to string operations like
regular expressions.
existing queries into natural language is easier than training them to produce queries in the Overpass language. Using the existing Overpass queries is therefore a way to efficiently create a dataset of paired question-query tuples. Second, the original queries were entered by developers and users, thus the queries satisfy a real-world information need and exploit the full expressivity of Overpass instead of being based on the annotators’ limited knowledge of the Overpass language.

### 3.2 Complexity of Semantic Parsing Data

As the example in Section 3.1 shows, queries in OverpassNL often make use of regular expressions. In contrast, queries in the Spider dataset (Yu et al., 2018) consist only of simple string matching operations, such as strings starting, containing or ending with a specific (sub)string. ATIS (Dahl et al., 1994; Iyer et al., 2017) and WikiSQL (Zhong et al., 2017) queries do not even contain string operations, but only exact matches. Statistics comparing dataset complexity of OverpassNL to Spider, WikiSQL, and ATIS are given in Table 1. All of these properties show that OverpassNL offers a setting that has been lacking in research so far. We work with a new query language with its own challenges, such as regular expressions and the NoSQL-style that allows concise queries against a complicated database. Moreover, the underlying database consists of only one highly connected database, making it possible to issue many different queries, resulting in a high vocabulary size.

## 4 (Neural) Semantic Parsing

In addition to the dataset, we also present a first cut on semantic parsing, showcasing the complexity of the talk. We first employ a generic sequence-to-sequence neural network (Sutskever et al., 2014) the encoder-decoder variant from (Luong et al., 2015). We use Joey NMT (Kreutzer et al., 2019) as framework to build the baseline parser.

Given a dataset \( D = \{ (X_n, y_n) \}_{n=1}^N \) of natural language questions \( X \) and corresponding queries \( y \), standard supervised training is performed by minimizing the average cross-entropy loss:

\[
L = -\frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T_n} \log p(y_{n,t} \mid y_{n,<t}, X_n), \quad (1)
\]

where the sum is over all timesteps \( t = 1 \) to \( t = T_n \).
for sample \( n \).

The natural language question is fed into a bidirectional RNN (GRU) to generate the hidden states \( h \in \mathbb{R}^{[X] \times m} \), where \( [X] \) is the number of source inputs and \( m \) is the hidden state size. The decoder takes its previous hidden state \( s_{t-1} \) and calculates a context vector \( c_t \) with an attention mechanism (Bahdanau et al., 2015) such that \( c_t = \text{att}(s_{t-1}, h) \). This context vector is then used for prediction by passing it through another feed-forward and softmax layer to generate the output distribution. Meta-parameter settings used in our experiments can be found in Table 6 in the Appendix.

5 Evaluation Measures

We use a parseval-style (Black et al., 1991) evaluation metric that matches a generated query \( q_{\text{pred}} \) against a gold standard parse \( q_{\text{gold}} \) and counts how often the predicted key-value pairs \( \text{kv}(q_{\text{pred}}) \) match their counterparts \( \text{kv}(q_{\text{gold}}) \) in the gold standard parse. This is similar to the component matching done in Yu et al. (2018). Our parse_match metric is based on the Dice Coefficient (Dice, 1945) where the key-value pairs in predicted and gold parse is measured:

\[
\text{parse_match} = \frac{1}{|Q|} \sum_{q \in Q} \frac{|\text{kv}(q_{\text{pred}}) \cap \text{kv}(q_{\text{gold}})|}{\max(|\text{kv}(q_{\text{pred}})|, |\text{kv}(q_{\text{gold}})|)}.
\]  

(2)

Furthermore, we use a grounded evaluation metric that executes the queries against the OpenStreetMap database and computes an execution accuracy by matching the predicted results against the correct result. It is computed as follows:

\[
\text{exec_acc} = \frac{\sum_{q \in Q} \delta(\text{res}(q_{\text{gold}}), \text{res}(q_{\text{pred}}))}{|Q|},
\]  

(3)

where \( \delta(i, j) \) is the Kronecker delta and \( \text{res}(q) \) is a function that executes the query \( q \) and returns the results.

However, sometimes a hypothesis query executes, but produces only a part of the correct output. Therefore we use an additional metric that computes a part_exec average over partially correct query results:

\[
\text{part_exec} = \frac{1}{|Q|} \sum_{q \in Q} \frac{|\text{res}(q_{\text{gold}}) \cap \text{res}(q_{\text{pred}})|}{|\text{res}(q_{\text{gold}})|}.
\]  

(4)

6 Experiments

6.1 Experimental Setup

A state-of-the-art sequence-to-sequence model trained on the dataset achieves an execution accuracy of 21% when executing the predicted queries against the OSM database, showing that semantic parsing of the OverpassNL dataset is indeed a challenging task. We find that the difficulty stems from three sources: 1) The correct use of database keys and values, since a database schema cannot be provided; 2) The complex syntax of Overpass queries; 3) The limited size of the dataset. An example of a predicted and gold parse for a natural language question can be found in Table 7 in the Appendix. Our goal is to solve these problems by the following three approaches:

1. **db_info**: Adding additional information such as possible database keys and values to the model input (countering difficulty 1).

2. **query_templates**: Providing templates to help with the difficult syntax by retrieving similar examples from the training data or by clustering the data and providing the cluster ID (countering difficulty 2).

3. **data_augmentation**: Creating a synthetic silver training dataset by templating and substituting tokens in questions and queries (countering difficulty 3).

6.1.1 Database Information

The OSM database is accessible through Overpass using keys and values. However, it is hard for the model to find the correct key for a value that appears in a natural question because the keys are often very general and cannot simply be inferred from the value. To avoid this difficulty, we aim to find the corresponding keys through string matching in order to provide the keys and values along with the input question. As shown in the example below, the keys and values are simply appended to the input string with a [SEP] token separating the real natural language question and the additional information. Keys and values are marked with [K] and [V], respectively:

Charging stations around motorway A 8 in Germany. [SEP] [K] amenity [V] charging_station [K] highway [V] motorway [SEP]
Table 2: Statistics about Additional Information (keys and values) that was added to the data

<table>
<thead>
<tr>
<th>Data</th>
<th>#Examples with add. info</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>train</td>
<td>4117</td>
<td>58.76 %</td>
</tr>
<tr>
<td>dev</td>
<td>576</td>
<td>38.40 %</td>
</tr>
<tr>
<td>test</td>
<td>572</td>
<td>57.20 %</td>
</tr>
</tbody>
</table>

The approach db_info makes use of a nominatim table\(^5\). This table maps OSM entries to categories to be used as keys and values in Overpass. Similar to Lin et al. (2020), we apply a fuzzy string matching algorithm to obtain the additional information from the nominatim table. The exact algorithm is explained in Appendix A.6. Naturally, matches can only be found if the word in the natural language question (± two characters) appears in the nominatim table. This is not the case for all examples in the OverpassNL dataset: Overall, for the train and test set, keys and values could only be added in around 60 % of the cases. Exact numbers can be found in Table 2.

### 6.1.2 Query Templates

In approach retrieve, we follow Hashimoto et al. (2018) to retrieve for every natural language question \(x\) the most similar question-query pair \((x', y')\) from the training data, using BERTScore (Zhang et al., 2020) as similarity metric. These additional inputs are fed into different encoders with their own attention mechanisms. The output of the encoders are then concatenated in the order \(x, x', y'\) and fed into the decoder, turning this into a multi-source setup (Zoph and Knight, 2016). This gives the model access to a similar question-query pair through the additional encoded input.

In approach cluster, we provide the model with additional information about the type of query. This approach is inspired by previous approaches to use control tags as additional inputs (Sennrich et al., 2016a). We first embed the natural language questions with BERT (Devlin et al., 2019), cluster the data with the k-Means clustering algorithm (k=10), and then augment the data with a special tag indicating the corresponding cluster. For example, similar natural language questions like planetarium in current view and places of worship in current view will be assigned to the same cluster. Examples for clusters are given in Fig. 6 in the Appendix.

---

\(^5\)https://wiki.openstreetmap.org/wiki/Nominatim/Special_Phrases/EN

---

**gold question** Recycling in admin level 10 areas with the name Kupferdreh

**silver question** Restaurants in admin level 5 areas with the name La Vida

**gold query** area["name"="Kupferdreh"] [admin_level=10]->.a;
(node(area.a)) ["amenity"="recycling"];)

**silver query** area["name"="La Vida"] [admin_level=5]->.a;
(node(area.a)) ["amenity"="restaurant"];)

Figure 2: Example for a silver example creation. The underlined part in the gold example is replaced by random values from the training data to create the silver data.

### 6.1.3 Data Augmentation

Lastly, we conduct two further data augmentation strategies. In approach substitution, we generate silver data by jointly templating both natural language questions and queries, replacing tokens occurring in both question and query. Afterwards we insert random values from the training data into the template slots. This resulting data was then filtered by removing nonsense natural language questions according to their sentence probability predicted by GPT-2 (Radford et al., 2018). The sentence probability was normalized by sentence length and thresholded with a value of 0.0001. A full example can be seen in Figure 2. The final silver dataset contains 14,000 examples and is used on its own or combined with the other approaches.

In approach back-trans, we made use an approach inspired by backtranslation (Sennrich et al., 2016b). We use the existing question-query pairs to train a query-to-question model that was then used to generate natural language questions for queries that were not given manual annotations with questions. This process resulted in additional 19,000 data points that were added to the training data.

### 6.2 Experimental Results

As shown in the top part of Table 3, the baseline performance of our model achieves only 21% execution accuracy. Using approach retrieve to retrieve similar question-query pairs through the additional encoded input increases the model per-
Table 3: Accuracy in percent of different semantic parsing models: A baseline, enhanced by retrieving similar question-query pairs (+ retrieve), augmenting the data with special cluster tags (+ cluster) and adding more training data using automatic generated data (+ substitution or + back_trans). All results are significantly better than the baseline (p < 0.001).

<table>
<thead>
<tr>
<th>Approach</th>
<th>Model</th>
<th>exec_acc</th>
<th>part_exec</th>
<th>parse_match</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>baseline</td>
<td>0.21</td>
<td>0.43</td>
<td>0.22</td>
</tr>
<tr>
<td>Database Information</td>
<td>db_info</td>
<td>0.33</td>
<td><strong>0.62</strong></td>
<td>0.35</td>
</tr>
<tr>
<td>Query Templates</td>
<td>cluster</td>
<td>0.35</td>
<td>0.6</td>
<td>0.35</td>
</tr>
<tr>
<td></td>
<td>retrieve</td>
<td>0.33</td>
<td>0.57</td>
<td>0.3</td>
</tr>
<tr>
<td>Data Augmentation</td>
<td>back_trans</td>
<td>0.33</td>
<td><strong>0.62</strong></td>
<td>0.35</td>
</tr>
<tr>
<td></td>
<td>substitution</td>
<td>0.35</td>
<td>0.6</td>
<td>0.31</td>
</tr>
<tr>
<td>Combined</td>
<td>cluster</td>
<td>0.35</td>
<td>0.61</td>
<td>0.33</td>
</tr>
<tr>
<td>Combined</td>
<td>db_info</td>
<td><strong>0.36</strong></td>
<td><strong>0.62</strong></td>
<td><strong>0.37</strong></td>
</tr>
<tr>
<td>Combined</td>
<td>retrieve</td>
<td>0.35</td>
<td>0.61</td>
<td>0.33</td>
</tr>
<tr>
<td>Combined</td>
<td>substitution</td>
<td>0.34</td>
<td>0.61</td>
<td>0.32</td>
</tr>
</tbody>
</table>

Table 4: Analysis of error types that make queries not executable against the database. Parse errors are errors like missing closing brackets, static errors are wrong keywords and nominatim errors are errors that hinder nominatim to return area ids for locations.

<table>
<thead>
<tr>
<th>model</th>
<th>parse</th>
<th>static</th>
<th>nominatim</th>
</tr>
</thead>
<tbody>
<tr>
<td>baseline</td>
<td>0.05</td>
<td>0.02</td>
<td>0.11</td>
</tr>
<tr>
<td>+cluster</td>
<td>0.05</td>
<td>0.03</td>
<td>0.05</td>
</tr>
<tr>
<td>+retrieve</td>
<td>0.05</td>
<td>0.02</td>
<td>0.04</td>
</tr>
<tr>
<td>+db_info</td>
<td>0.07</td>
<td>0.02</td>
<td>0.05</td>
</tr>
<tr>
<td>+cluster+db_info</td>
<td>0.06</td>
<td>0.02</td>
<td>0.03</td>
</tr>
</tbody>
</table>

formance by 12 points to 33% execution accuracy. The db_info approach reaches the same performance. Allowing the model to easily generate similar queries by using approach cluster also leads to a better performance with 35% execution accuracy. Finally, using approach substitution to add the silver data to our training data, the model also achieves 35% execution accuracy.

The bottom part of Table 3 shows the results for the best combinations of approaches. Combining cluster with either retrieve or substitute achieves a score of 35% execution accuracy. Combining cluster and db_info yields the highest improvement, reaching 36% execution accuracy. Combining all three methods does not lead to further improvements. We conjecture that this result can be explained by a certain amount of redundancy in the information provided by retrieving similar instances or adding silver data, with the most accurate addition to the cluster information being provided by the explicit keys and values in the db_info approach. This combination also reaches the highest values according to the partial execution and parse match metrics.

In order to investigate the interaction of data properties and parsing performance, we took a closer look at the data characteristics of question length. Our hypothesis was that the dataset poses increased difficulties due to increased question length: The longer the question, the harder to find the correct query. In order to test this hypothesis, we use an LMEM-based significance test (Riezler and Hagmann, 2022) to investigate the interaction between the question length and the execution accuracy. With a p-value of < 0.01, question length makes a significant difference. This can be confirmed by fitting a line to the results split by question length, as can be seen in Figure 3. The negative gradient confirms our observation. An advantage of the best model (db_info + cluster) is that it seems to close the gap between the performance difference of long and short examples. As Figure 3 shows, the base model (left) performs worse the longer the natural language question gets. However, the best model seems to perform equally well independent of the question length. As the p-value shows, the line of best fit is not significantly different from a horizontal line, which would indicate no performance loss due to the question length. Further information on the distribution of the test data due to question length can be found in Figure 5 in the Appendix.
(a) Base Model: The line of best fit is significantly different from a horizontal line (p: 0.006).

(b) Base Model enhanced with cluster tags and db_info: The line of best fit is not significantly different from a horizontal line (p: 0.58).

Figure 3: Interaction of execution accuracy and question length in the base model and the best models. The questions were binned based on their sentence length. The average execution accuracy of each bin (blue dots) is measured on the y-axis. The line of best fit is illustrated in red.

7 Error Analysis

An error analysis (Table 4) shows that for the baseline parser, 11% of the queries do not yield a correct result due to nominatim errors. In these cases, the geolocation service provided by nominatim cannot find an id for a query string like ‘Nermany’ instead of ‘Germany’. For the best model that uses cluster and db_info, the nominatim error rate for the dataset is significantly lower at 3%. The nominatim error rate is the lowest even compared to models that use only one enhancement, the lowest being retrieve having an error rate of 4%.

An inspection of selected examples shows that the baseline model seems to have a problem with hallucination by inserting values in the hypotheses that appear often in the query but are different from the values given in the questions. Giving the model access to query templates via cluster or retrieve appears to make the model hallucinate less. In the following example, the baseline model inserts the correct uid only in one of the two places, whereas the improved model correctly predicts the correct uid in both places.

**Question:** Ways and nodes with the uid 9847941 newer than yesterday

**Baseline:** (way(uid:9847994)
  (newer:"{{date:1day}}");
  node(uid:9847941)
  (newer:"{{date:1day}}");); out;

**cluster:** (way(uid:9847941)
  (newer:"{{date:1day}}");
  node(uid:9847941)
  (newer:"{{date:1day}}");); out;

Cluster and retrieve also seems to reduce the generation of typos, as can be seen in the following example, where the baseline model produces the typo "miltary" instead of "military".

**Question:** Way with the attribute usage having a value military in Colorado

**Baseline:** geocodeArea:Colorado->.searchArea; ( way["usage"="miltary"] (area.searchArea)); out;

**cluster:** geocodeArea:Colorado->.searchArea; ( way["usage"="military"] (area.searchArea)); out;

Interestingly, even if cluster or retrieve approaches have never seen a certain value in the training data (like "furnace" in the following example), they seem to be able to copy better from the source than the baseline model.

**Question:** furnace shops in current view

6https://nominatim.openstreetmap.org/ui/search.html?q=Germany
8 Conclusion

We introduced OverpassNL, a new dataset for semantic parsing and interpretation of Overpass queries to the OpenStreetMap database. OverpassNL is a semantic parsing dataset that builds upon complex real-world user queries issued to a large-scale complex database. We illustrate the complexity of the dataset and the difficulty of the semantic parsing task, with the baseline model only reaching around 21% of execution accuracy. We then improved the model by incorporating more information, either by feeding similar examples into the model, by exploiting similarities in the natural language questions, and by enhancing our training data with silver data. Our best model then reaches an execution accuracy of 36%.

9 Future Work

An avenue of research we aim to pursue in the future is to use the PICARD (Scholak et al., 2021) algorithm which led to improvements on the Spider dataset by constraining the beam search to valid outputs. A reimplementation for the Overpass syntax could also yield improvements in our experiments.

Additionally we want to research the possibility of augmenting our models with even more knowledge sources, for example the contents of the OpenStreetMap wiki . Lastly, we are planning to establish an interactive setup where OSM users and developers can use a semantic parser trained on OverpassNL and provide feedback for interactive machine learning.

10 Limitations

A possible limitation of the presented work could be an inherent bias in the developer-generated data, for example, a gender bias, or simply a bias towards queries that appear complex on the surface, but ask for trivial contents. We hope that a future interactive scenario will encourage users and developers to take advantage of the natural language interface to query for interesting contents.

7https://wiki.openstreetmap.org/wiki/


A Appendix

A.1 Overpass Statistics

<table>
<thead>
<tr>
<th>Statistics</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of OSM users</td>
<td>8.3 million</td>
</tr>
<tr>
<td>number of nodes in OSM</td>
<td>7.4 billion</td>
</tr>
<tr>
<td>map changes per day in OSM</td>
<td>4.5 million</td>
</tr>
</tbody>
</table>


Figure 4: Accumulated registered users (linear scale) of OpenStreetMap (https://wiki.openstreetmap.org/wiki/Stats)
A.2 Dataset Properties

Figure 5: Distribution of the test data due to question length (in characters). The dotted line indicates the arithmetic mean.
A.3 Hyperparameter Settings

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>optimizer</td>
<td>adam</td>
</tr>
<tr>
<td>learning rate</td>
<td>0.0002</td>
</tr>
<tr>
<td>batch size</td>
<td>4</td>
</tr>
<tr>
<td>encoder rnn type</td>
<td>bidirectional GRU</td>
</tr>
<tr>
<td>attention</td>
<td>bahdanau</td>
</tr>
<tr>
<td>encoder embedding dim</td>
<td>620</td>
</tr>
<tr>
<td>encoder hidden dim</td>
<td>400</td>
</tr>
<tr>
<td>encoder layers</td>
<td>1</td>
</tr>
<tr>
<td>decoder rnn type</td>
<td>GRU</td>
</tr>
<tr>
<td>decoder embedding dim</td>
<td>620</td>
</tr>
<tr>
<td>decoder hidden dim</td>
<td>800</td>
</tr>
<tr>
<td>decoder layers</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 6: Hyperparameter settings of JoeyNMT sequence-to-sequence model used in our experiments.

A.4 Semantic Parsing Example

| SRC | Highways or routes with official name  
Rodovia Vespertino de Medeiros Bonorino in Brasil |
|-----|-----------------------------------------|
| PRED. | {{geocodearea:rs,brasil}}->.searcharea;  
(way["highway"~".*" ]["official_name"~  
"^rodovia estadual joão cândido$"])(area.searcharea); |
| GOLD | {{geocodearea:rs,brasil}}->.searcharea;  
(way["highway"~".*" ]["name"~  
"^rodovia vespertino de medeiros bonorino$"])(area.searcharea); |

Table 7: Semantic parsing example. SRC is the natural language question, PRED. the predicted query and GOLD the correct query.
A.5 Cluster Examples

• Cluster 0
  - Admin level 3 in Russia
  - Admin level 3 in Tanzania
  - Admin level 4 in Angola

• Cluster 2
  - places I can grill outside in current view
  - places of worship in current view
  - planetarium in current view

• Cluster 6
  - Boundary relations in Rio Grande do Sul, Brazil with IBGE order numbers matching the regular expression "^43[0-9]{8}$"
  - Milestones in mesorregião do oeste catarinense that have a description or a reference matching "^SC"

• Cluster 9
  - Nodes and ways that were changed between 2018-07-02T00:00:00Z and 2018-07-02T19:39:59Z by the user with the ID 8076784
  - Nodes and ways that were changed between 2019-07-10T00:00:00Z and 2019-07-10T23:59:59Z by the user with the ID 8710004
  - Nodes and ways that were edited between 2019-02-11T00:00:00Z and 2019-02-11T23:55:59Z by the user with the ID 7725447

Figure 6: Cluster examples that were used to improve the performance of our encoder-decoder model.

A.6 Fuzzy String Matching

For this algorithm, the natural language question and the whole word/phrase column from the nominatim table are converted into lower-cased character sequences and the longest subsequence match between the question and the column values is computed. The subsequence match is only considered valid if the word boundaries can be detected within ±2 characters of the match, thereby matches that are substrings of the words in the natural language question such as “way” in “motorway” are excluded. Additionally, if there is a preposition right after the word in the natural language question, it is checked whether the preposition appears in the nominatim table in the column “operator”.

14