Text-to-OverpassQL: A Natural Language Interface for Complex Geodata Querying of OpenStreetMap

Michael Staniek[*1] **Raphael Schumann**[*1] **Maike Züfle**[◇1,2] **Stefan Riezler**[1,3]
[1]Computational Linguistics, Heidelberg University, Germany
[2]School of Informatics, University of Edinburgh, UK
[3]IWR, Heidelberg University, Germany
{staniek,rschuman,zuefle,riezler}@cl.uni-heidelberg.de

Abstract

We present Text-to-OverpassQL, a task designed to facilitate a natural language interface for querying geodata from OpenStreetMap (OSM). The Overpass Query Language (OverpassQL) allows users to formulate complex database queries and is widely adopted in the OSM ecosystem. Generating Overpass queries from natural language input serves multiple use-cases. It enables novice users to utilize OverpassQL without prior knowledge, assists experienced users with crafting advanced queries, and enables tool-augmented large language models to access information stored in the OSM database. In order to assess the performance of current sequence generation models on this task, we propose OverpassNL[1], a dataset of 8,352 queries with corresponding natural language inputs. We further introduce task specific evaluation metrics and ground the evaluation of the Text-to-OverpassQL task by executing the queries against the OSM database. We establish strong baselines by finetuning sequence-to-sequence models and adapting large language models with in-context examples. The detailed evaluation reveals strengths and weaknesses of the considered learning strategies, laying the foundations for further research into the Text-to-OverpassQL task.

1 Introduction

The OpenStreetMap (OSM) database stores vast amounts of structured knowledge about our world. Users mainly access it via applications that render a visual map of inquired areas. A more advanced and systematic way to examine the stored information is to query the underlying geodata using the Overpass Query Language (OverpassQL). OverpassQL is a feature-rich query language that is widely adopted in the OSM ecosystem, by contributors, analysts, and applications. In order to make this empowering query language accessible via natural language, we propose the Text-to-OverpassQL task. As is shown in Figure 1, the objective is to take a complex data request in natural language and translate it to OverpassQL in order to execute it against the OpenStreetMap database. Several groups of users can benefit from such a natural language interface. Inexperienced users are spared from learning the OverpassQL syntax. Expert users can use it to draft Overpass queries and then manually refine them, saving time and mental load in comparison to writing the full query from scratch.

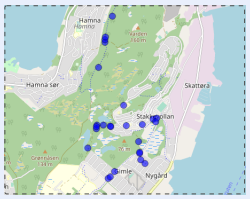
Natural Language Input:

Bike lanes 500 meters around the top of a hill or mountain in Troms.

Overpass Query Language:

```
{{geocodeArea:"Troms"}}->.searchArea;
(
  node["natural"="peak"](area.searchArea);
)->.peaks;
way["highway"="cycleway"](area.searchArea)(around.peaks:500);
out center;
```

Query Execution Results:



```
{"elements": [
  {"type":    "way",
   "id":      4013565,
   "center":  {"lat": 69.6900047,
               "lon": 18.9971581},
   "tags":    {"highway": "cycleway",
               "surface": "asphalt",
               "segregated": "no"},
  },
...
```

Figure 1: Natural language input and the corresponding Overpass query. The query is executed against the OpenStreetMap database and returns the requested elements in a structured response. The Overpass query language is highly expressive and allows to formulate complex queries to extract information from OpenStreetMap. Blue tokens in the query are syntax keywords, orange tokens are variable names and bold tokens define semantic properties of the requested elements. The green token in curly brackets geolocates an area called "Troms".

[*]equal contribution [◇]work done while at Computational Linguistics, Heidelberg University

[1]https://github.com/raphael-sch/OverpassNL

Another use-case would be to incorporate the interface as an API tool (Schick et al., 2023) for large language models (LLMs).

In this work, we introduce the three components that enable the Text-to-OverpassQL task. First, we present OverpassNL, a dataset of 8.5k natural language inputs and corresponding Overpass queries. The queries were collected from an OSM community website where they were written by OSM users to fulfill legitimate information needs. We then hired and trained students to write natural language descriptions of the queries. Second, we introduce a systematic evaluation protocol that assesses the prediction quality of a candidate system. To this end, we propose a task-specific metric that takes the similarity of the system output to Overpass queries on the levels of surface string, semantics, and syntax into account. Moreover, we ground the evaluation by executing the generated query against the OSM database and compare the returned elements with those returned by the gold query. Third, we explore several models and learning strategies to establish a base performance for the problem of generating Overpass queries from natural language. We finetuned sequence-to-sequence models and found that explicitly pretraining on code is not helpful for the task. We further explored in-context learning strategies for black-box LLMs and found that GPT-4 (OpenAI, 2023) with few-shot examples, retrieved by sentence similarity, yields the best results, outperforming the finetuned sequence-to-sequence models.

The proposed Text-to-OverpassQL task is a novel semantic parsing problem that is well motivated in real-world applications. While it shares characteristics with the Text-to-SQL task and its accompanying datasets, there are several key differences. The Text-to-OverpassQL task is grounded in a database that is genuinely in use and of global scale. The database is not divided into sub-databases or tables, and each Overpass query can retrieve any of the billions of stored elements in OSM. The desired elements have to be queried by a geographical specification and additional semantic tags. The tags are composed of key-value pairs that follow established community guidelines and conventions, but can also be open-vocabulary. Overall, the proposed task builds on a decades-long effort to structure and store the geographical world around us in a way to make it computationally accessible. With this work, we offer all components to benchmark future semantic parsing systems on a challenging real-world task.

Our main contributions are as follows: (i) We present OverpassNL, a dataset of 8.5k natural language inputs paired with real-world Overpass queries. (ii) We define task-specific evaluation metrics that take the OverpassQL syntax into account and are grounded in database execution. (iii) We train and evaluate several state-of-the-art sequence generation models to establish base performance and to identify specific properties of the proposed Text-to-OverpassQL task.

2 Background

2.1 OpenStreetMap

OpenStreetMap (OSM) is a free and open geographic database that has been created and is maintained by a global community of voluntary contributors. The ever growing community has over 10M registered members who have collectively contributed to the creation of the existing 9B elements in the database (OpenStreetMap Wiki, 2022). Elements are either nodes, ways, or relations. Nodes are annotated with geospatial coordinates. Ways are composed of multiple nodes and represent roads, building outlines, or area boundaries. Relations describe the relationships of elements, e.g., forming a municipal or major highway. Elements can be tagged with key-value pairs that assign semantic meaning and meta information. The OSM database is widely used in geodata analysis, scientific research, route planning applications, humanitarian aid projects, or augmented reality games. It also serves as a data source for geospatial services of companies like Facebook, Amazon or Apple (OpenStreetMap Foundation, 2019).

2.2 Overpass Query Language

The Overpass Query Language (OverpassQL) is a "procedural, imperative programming language written with a C style syntax" (OpenStreetMap Wiki, 2023). It is used to query the OpenStreetMap database for geographic data and features. OverpassQL allows for detailed queries that are capable of extracting elements based on specific criteria, such as certain types of buildings, streets, or natural features within a defined area. Users can specify the types of elements they are interested in, and filter them by their associated key-value pairs.

Query Syntax We briefly explain the OverpassQL syntax based on the query depicted in Figure 1 and refer to the official language guide for more information[2]. The keyword *geocodeArea* in the first line triggers a geolocation service[3] to find an area named "*Troms*". The retrieved area is then assigned to a variable named *searchArea*. The third line queries for nodes that are tagged with the *natural=peak* key-value pair. The search is limited to nodes that are geographically within the previously defined *searchArea*. The nodes that fullfill these criteria are stored into the *peaks* variable. Line 5 queries for ways within the same area that are tagged with *highway=cycleway* and are within a radius of 500 meters around a node stored in *peaks*. Finally, the query requests a return of the specified ways.

3 Related Work

Natural Language Interfaces for Geodata One of the first attempts to build a natural language interface for geographical data is GEOQUERY (Zelle and Mooney, 1996; Kate et al., 2005b). It is a system based on Prolog, later adapted to SQL, and is tailored for a small database of U.S. geographical facts. Following works proposed methods to map the text input to the structured query language (Zettlemoyer and Collins, 2005; Kate et al., 2005a). A more recent attempt is NLmaps (Haas and Riezler, 2016; Lawrence and Riezler, 2016) which aimed to build a natural language interface for OpenStreetMap. For querying the database, they designed a machine readable language (MRL). The MRL is an abstraction of the Overpass Query Language, but it supports only a limited number of its features. To facilitate building more potent neural sequence-to-sequence parsers, they later released NLmaps v2 (Lawrence and Riezler, 2018) with augmented text and query pairs. Our work aims to support the Overpass Query Language without simplifications or abstractions, allowing to fully leverage the effort of the OpenStreetMap community that developed a query language that is optimally suited for the large-scale geospatial information in the OSM database.

Text-to-SQL The Text-to-SQL task (Tang and Mooney, 2001; Iyer et al., 2017; Li and Jagadish, 2014; Zhong et al., 2017) is closely related to the proposed Text-to-OverpassQL task and aims to provide a natural language interface to relational databases. While most of the works in this area focus on a specific domain and database, the Spider (Yu et al., 2018) dataset provides text and queries for multiple databases spanning different domains. They emphasize the hurdles of collecting real databases with complex schemas and sufficient data records, and circumvent this problem by mainly sourcing the databases from educational material and populate them with synthetic data. In contrast, the OpenStreetMap database is of global scale and is used in real-world applications. We highlight more differences between the datasets and underlying databases in Section 4.4. The Text-to-SQL task is commonly treated as a sequence-to-sequence problem (Lin et al., 2020; Yu et al., 2021). Some methods explicitly encode the database schema (Zhang et al., 2019), while Scholak et al. (2021) show that finetuning a pretrained T5 model (Raffel et al., 2019) matches the performance of more specialized systems. They further introduced PICARD, a constraint decoding method that is SQL specific and enforces syntactic correctness. With the advent of large language models and in-context learning, more recent work focuses on prompt engineering the task (Sun et al., 2023; Chen et al., 2023; Pourreza and Rafiei, 2023).

4 OverpassNL Dataset

In order to facilitate the Text-to-OverpassQL task, we constructed a parallel dataset of natural language inputs and Overpass queries. This was done by collecting queries written and shared by users of Overpass Turbo[4], a web tool that allows to develop and execute Overpass queries within a graphical interface. We presented the queries to trained annotators who were tasked with writing natural language descriptions for them.

Initiating the dataset creation process with Overpass queries authored by real users and developers has several advantages: Firstly, the queries were created to satisfy legitimate information needs and, as such, cover a wide range of OverpassQL features. Also, the geographical coverage is high, as is shown in Figure 3 by the location distribution of elements returned by executing the queries in our dataset. Another advantage is that it is easier to

[2] https://wiki.openstreetmap.org/wiki/Overpass_API/Language_Guide

[3] https://nominatim.org/

[4] The shared queries constitute an unrestricted collection that is publicly available on https://overpass-turbo.eu/. The website is maintained by Martin Raifer, who helped us to acquire queries shared between 2014-2022.

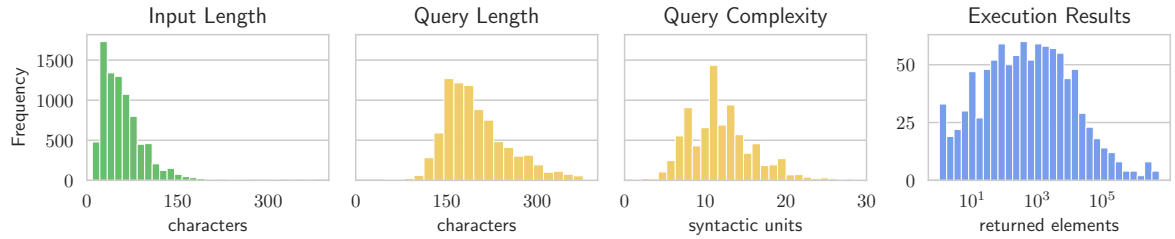


Figure 2: Dataset Statistics. Number of elements returned when executing the queries in the development set against OpenStreetMap. Each query returns at least one element and often several orders of magnitude more.

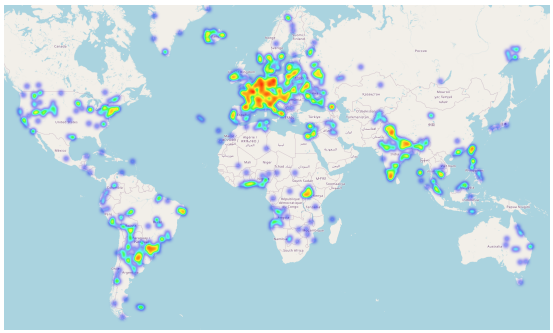


Figure 3: Location distribution of results returned by Overpass queries in our dataset. The queries cover locations on all continents. Europe is a traditional hotspot of the OpenStreetMap community and also has the best mapping coverage.

teach annotators how to interpret Overpass queries than how to write them from scratch.

4.1 Query Annotation

For the annotation task, we recruited university students with proficient English skills and experience in database query languages like SQL. They had to complete a tutorial about OverpassQL and were subsequently tested on their knowledge. The test consisted of multiple choice questions and assignments to write text descriptions of pre-selected Overpass queries. Only the 15 students that passed this test were selected to participate in our annotation task. We built a graphical interface that showed them an Overpass query, the raw execution results, and the results rendered on a map. The task of the annotators was to write a natural language description that best represents the query. To ensure quality, we encouraged the annotators to use the Overpass documentation, continuously conducted spot tests on the submitted inputs, and required the annotators to validate the inputs written by other annotators. A screenshot of the annotation interface is shown in Appendix G. We paid 100€ per 250 annotations, resulting in a wage of around 20€/hour.

4.2 Dataset Statistics

In total we obtained 8,352 queries annotated with natural language inputs. We split these into 6,352 instances for training, 1,000 for development, and 1,000 test instances. We constructed the splits such that there are no (near) duplicates on the input side or query side between training and evaluation instances. Figure 2 gives some statistics illustrating important dataset properties. There are a total of 11,259 distinct words in the natural language inputs, the mean input length is 59.7 characters, the mean query length is 199.8 characters, and each query has an average of 11.9 syntactic units. A syntactic unit is a subtree in the XML representation of an Overpass query. The rightmost plot in Figure 2 shows the number of elements returned by executing the development set queries.

4.3 Complexity & Coverage

There are a total of 41 major syntax features in the Overpass Query Language. 31 of these 41 features occur in at least 20 queries of our dataset, resulting in a feature coverage rate of 76%. See Appendix C for a detailed list of syntax features. Our queries utilize 1,046 unique keys to specify tagged elements. These keys cover 91% of all key usage in OpenStreetMap (see Appendix A). The coverage of corresponding values is harder to estimate because of open-class keys like *name*, *source* or *operator*. There are also keys that have recommended sets of values, e.g., the key *leisure* is commonly paired with *swimming_pool*, *skatepark* or *pitch*. In total, we count 3,879 unique values and 4,880 unique key-value pairs in our dataset queries.

4.4 Comparison to Other Datasets

Because we are the first to present a dataset for the Text-to-OverpassQL task, we compare it to datasets of related tasks (see Table 1). GEOQUERY (Zelle and Mooney, 1996) is a small-scale

Dataset	Query Language	Number of Instances	Query Templates	Named Identifiers		Extractable Elements		Results
				total	per database	total	per database	per query
GeoQuery	SQL	880	234	31 & 7	31 & 7	937	51	5
NLmaps	MRL	2,380	379	107	107	3.4B	3.4B	-
NLmaps v2	MRL	28,609	360	347	347	3.4B	3.4B	-
WikiSQL	SQL	81,654	-	168k	6.38	460k	11	1
Spider	SQL	10,181	5,693	4,669 & 876	58 & 5	1.6M	9.6k	30
OverpassNL	OverpassQL	8,352	3,890	1,046	1,046	9B	9B	10k

Table 1: Comparison of Text-to-Query datasets. Templates are normalized queries, i.e. removing named identifiers, variable names and digits. Named identifiers are tag keys in OpenStreetMap related datasets and table & column names in SQL related datasets. The Spider dataset includes 166 unconnected databases and the task is to generate a query for a specific database that is known a priori. Thus, the average number of relevant named identifiers and extractable elements is much smaller per query than for the whole dataset.

dataset with 880 instances that allow to query 937 different geographical facts about the United States. NLmaps (Haas and Riezler, 2016) and NLmap v2 (Lawrence and Riezler, 2018) provide queries for OpenStreetMap paired with natural language inputs, however, the queries are written in their own restricted query language called MRL. In contrast, we generate queries in the well-established OverpassQL language that has more features and is widely used by the OpenStreetMap community. Additionally, the NLmaps datasets only include up to 347 distinct keys in key-value pairs, limiting the semantic expressiveness of generated queries. Furthermore, NLmaps was built for an older version of the OSM database comprising one third of the size of the current version used in our work. WikiSQL (Zhong et al., 2017) converts single tables from Wikipedia articles to SQL databases and annotates queries with natural language inputs for them. While the dataset is of large scale, it contains only simple SQL queries and unconnected tables. The Spider dataset (Yu et al., 2018) includes 166 distinct databases of different domains and was collected for the Text-to-SQL task. Each natural language input in the dataset is intended for a specific database that is known a priori. This significantly reduces the number of relevant table names and column names per query, and simplifies the task by allowing to append known table and column names together with the database schema to the input. This stands in stark contrast to our Text-to-OverpassQL task where each query can utilize any key-value pair and retrieve elements from the entire OSM database, making it harder to predict the correct named identifiers for the desired elements. Also, the number of returned elements per query is orders of magnitude larger than in SQL related datasets. This makes the grounded evaluation harder by minimizing the likelihood of false positive matches in execution accuracy.

5 Task & Evaluation

The Text-to-OverpassQL task requires to generate an Overpass query $\mathbf{q}$, given a natural language input $\mathbf{x}$. A model for this task aims to accurately translate the request, formulated in natural language, into code, written in the Overpass Query Language, that returns the correct elements when executed against the OpenStreetMap database. In order to evaluate such a system, we propose to use different evaluation metrics. These include metrics that compare the generated query with the reference query, and metrics that compare the results returned by executing the queries against the OSM database. In order to make the execution results reproducible, we release the evaluation script and a Docker container with the exact snapshot of the OSM database we used. We further describe the metrics in detail.

5.1 Overpass Query Similarity Evaluation

We propose a language-specific metric called Overpass Query Similarity (OQS) to quantify the compatibility of a generated query and the reference query. The metric is composed of three parts. First, we employ character F-score (chrF) which measures the overlap of character n-grams between two strings (Popović, 2015). Because chrF operates on the character-level, it is well suited for Overpass queries which consist of words and special characters alike. Next, we calculate the overlap of keys and values between the generated query $\mathbf{q}_G$ and reference query $\mathbf{q}_R$. We define the Key Value

Similarity (KVS) as follows:

$$\mathrm{KVS}(\mathbf{q}_G, \mathbf{q}_R) = \frac{|\mathrm{KV}(\mathbf{q}_G) \cap \mathrm{KV}(\mathbf{q}_R)|}{max(|\mathrm{KV}(\mathbf{q}_G)|, |\mathrm{KV}(\mathbf{q}_R)|)}, \quad (1)$$

where the operator $\mathrm{KV}(\cdot)$ returns all key-value pairs as well as all individual keys and all individual values. This metric captures the semantic relatedness of two queries. Complementary, the third part of the OQS metric compares queries on the syntactic level. We compute the Tree Similarity metric (TreeS) by comparing the XML tree representation of the queries. We remove all key-value pairs and variable names from the trees and recursively compute the number of matching subtrees of the generated and reference query. Analogous to Equation 1, we normalize the number of matching subtrees by the maximum number of subtrees in either tree. Finally, the proposed Overpass Query Similarity metric is the mean of chrF, KVS and TreeS. The metric captures similarity of system outputs and reference queries on the levels of surface string, semantics, and syntax.

5.2 Grounded Evaluation

The nature of the Text-to-OverpassQL task allows us to perform a grounded evaluation of generated queries by executing them against the OpenStreetMap database. We quantify the correctness of the database execution by Execution Accuracy (EX) which measures the exact match of all elements returned by executing the generated query and the reference query. Each returned element has an identifier number that is unique within OSM. We use this identifier number to determine exact matching of results. The plot on the right in Figure 2 shows that there are up to 10^7 elements returned by a query. The matching by unique identifier and large number of returned elements make EX an inherently hard metric to satisfy.

Because OpenStreetMap is a community driven database that has grown over decades with changing annotation guidelines, there can be ambiguities in the tags of elements. For example, filtering all bridges can mean *node["bridge"]* or *node["bridge"="yes"]*. Both are correct according to current annotation guidelines, but do return slightly different sets of nodes. Another example is filtering for radar towers:

node["man_made"="tower"]["tower:type"="radar"].

The first filter is redundant according to tagging guidelines, but can not be omitted in queries because the guidelines are not consistently followed throughout the whole database. To account for this, we also report Soft Execution Accuracy ($\mathrm{EX}_{\mathrm{SOFT}}$). It is computed as the overlap of returned elements, normalized by the maximum number of elements returned by either the generated or reference query. The metric ranges from 0, which means no overlap, to 1, which is equivalent to exact match of results. We report the metric as percent in the results tables.

6 Experiments

In the following, we present experiments that showcase the opportunities to train machine learning models on the OverpassNL dataset and establish base performance of commonly used techniques. We finetuned sequence-to-sequence models of different sizes and pretraining settings. Additionally, we adapted black-box large language models with different in-context learning strategies. All models are evaluated with the proposed similarity and grounded metrics, as well as exact string match (EM).

6.1 Finetuning

The task of generating Overpass queries from text is a sequence-to-sequence problem and can be addressed by a model with encoder-decoder architecture. The encoder processes the input text and the decoder autoregressively generates the Overpass query. In order to choose a suitable pretrained model, we focused on the T5 family of models because of their strong performance in a variety of sequence-to-sequence tasks (Raffel et al., 2019), in particular Text-to-SQL (Scholak et al., 2021). Besides the vanilla T5 model[5], the family also includes models specifically trained for code generation (CodeT5, Wang et al. (2021)) and models with byte-level tokenization (ByT5, Xue et al. (2022)). Because neither model has been pretrained on data covering OverpassQL syntax, we ran experiments to compare the finetuning of CodeT5 and ByT5 on the training portion of our dataset. To expand the training set, we additionally created instances from comments that query authors put in some lines with the intention to describe the line's purpose and functionality. We extracted the comments and used them as the natural language input for the respective query. This produced 6,000 additional training instances.

[5]The vanilla T5 model is not suitable for code or OverpassQL generation because it lacks tokens for '{' or '[' in its vocabulary (Wang et al., 2021).

Model	Setting	Overpass Query Similarity					Execution Accuracy	
		chrF	KVS	TreeS	**OQS**	**EM**	**EX**	**EX_{SOFT}**
Finetuning								
CodeT5-small		74.0 ±0.2	61.2 ±0.6	72.2 ±0.1	69.1 ±0.2	18.5 ±0.1	31.9 ±0.9	41.9 ±0.6
CodeT5-small	+comments	74.1 ±0.2	62.4 ±0.7	72.7 ±0.3	69.8 ±0.2	18.9 ±0.4	32.7 ±0.6	43.9 ±0.5
CodeT5-base		74.6 ±0.0	63.2 ±0.4	73.0 ±0.1	70.3 ±0.2	19.8 ±0.4	33.3 ±0.3	44.3 ±0.1
CodeT5-base	+comments	74.9 ±0.1	63.6 ±0.5	73.5 ±0.1	70.7 ±0.2	20.3 ±0.2	34.5 ±0.4	46.2 ±0.4
ByT5-small		74.8 ±0.2	64.4 ±0.2	73.1 ±0.2	70.8 ±0.2	20.4 ±0.1	35.4 ±0.3	46.8 ±0.3
ByT5-small	+comments	75.0 ±0.0	64.6 ±0.2	73.4 ±0.2	71.0 ±0.1	21.0 ±0.3	36.0 ±0.4	46.2 ±0.5
ByT5-base		**75.5** ±0.0	65.0 ±0.2	**73.8** ±0.2	71.4 ±0.0	21.9 ±0.4	36.2 ±0.0	46.7 ±0.4
ByT5-base	+comments	**75.5** ±0.1	**66.0** ±0.2	73.7 ±0.4	**71.7** ±0.1	**22.0** ±0.1	**36.7** ±0.6	**47.0** ±1.0
5-Shot In-Context Learning								
GPT-3	random	58.8	48.8	57.5	55.0	4.1	16.9	28.0
GPT-3	retrieval-BLEU	67.4	55.5	66.8	63.3	18.0	28.7	37.7
GPT-3	retrieval-sBERT	72.1	63.3	69.9	68.4	19.5	34.1	44.2
GPT-4	random	63.8	57.5	61.1	60.8	5.1	25.4	39.5
GPT-4	retrieval-BLEU	74.3	66.1	72.4	71.0	22.9	38.5	50.7
GPT-4	retrieval-sBERT	**75.7**	**69.9**	**74.0**	**73.2**	**23.4**	**40.4**	**53.0**

Table 2: Results on the **development set** of OverpassNL. The proposed Overpass Query Similarity (OQS) metric is the mean of Character F-score (chrF), Key-Value similarity (KVS) and XML-tree similarity (TreeS). EM denotes exact string match. Execution accuracy (EX) is the exact match of all results returned by executing the generated query and reference query against OpenStreetMap. Soft Execution Accuracy (EX_{SOFT}) is the normalized overlap of returned results. Results in **bold** are best for the respective learning setup. Finetuning experiments are repeated three times with different random seeds and mean/standard deviation are reported.

The upper half of Table 2 shows the results for combinations of model type, model size, and training data, evaluated on the development set. In general, the *base* variant of the models is better than the *small* variant. We did not gain further improvements by finetuning even larger variants of the models. The results also show that the ByT5 models are better suited for our task than the CodeT5 models. Although the instances derived from developer comments are of low quality, they contribute to consistent improvements in execution accuracy for all models. The best model for our task is *ByT5-base* with 582M parameters, finetuned on the enhanced training set. We further refer to this model as *OverpassT5*.

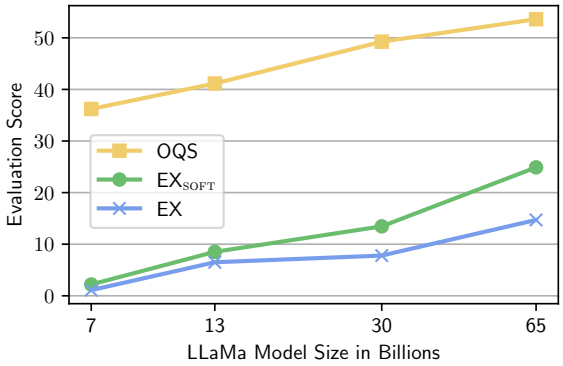


Figure 4: Development set results of LLaMa models with increasing number of parameters, prompted with five in-context examples.

6.2 In-Context Learning

We furthermore explore the use of in-context learning for the Text-to-OverpassQL task. We prompt large language models to generate an Overpass query for the given text input while providing five example pairs as context. The structure of the 5-shot prompt is shown in Appendix D. The example pairs are selected from the training set, either randomly or by input similarity (Liu et al., 2022). We compare BLEU (Papineni et al., 2002) and sentence-BERT embedding similarity (Reimers and Gurevych, 2019) as metrics to retrieve the most similar examples.

Figure 4 shows that the quality of queries generated by LLaMa (Touvron et al., 2023) increases with the model size. The lower half of Table 2 shows results for the even bigger GPT-3 (Brown et al., 2020) and GPT-4 (OpenAI, 2023) models. We see a similar trend of improved results with increasing model size. Furthermore, retrieving sim-

Model	Overpass Query Similarity						Execution Accuracy	
	chrF	KVS	TreeS	**OQS**	**EM**	**#Errors**	**EX**	**EX_{SOFT}**
Full Test Set (1,000 Instances)								
OverpassT5	**74.9** ±0.1	66.1 ±0.3	**72.7** ±0.2	71.2 ±0.2	**20.7** ±0.2	**23** ±5.6	33.9 ±0.1	46.3 ±0.3
GPT-4	73.6	**68.6**	72.0	**71.4**	**20.7**	34	**38.9**	**53.0**
Hard Partition (333 Instances)								
OverpassT5	62.8 ±0.4	57.7 ±0.4	56.6 ±0.3	59.1 ±0.3	8.8 ±0.3	15 ±4.5	18.7 ±0.1	29.7 ±0.5
GPT-4	61.4	59.6	56.3	59.1	9.0	25	22.2	35.3

Table 3: Results on the **test set** of our OverpassNL dataset. The proposed Overpass Query Similarity (OQS) metric is the mean of Character F-score (chrF), Key-Value Similarity (KVS) and XML-tree Similarity (TreeS). Exact match (EM) is query string match. #Errors are number of raised syntax errors when trying to execute the query. Execution accuracy (EX) is the exact match of all results returned by executing the model generated query and reference query against OpenStreetMap. Soft Execution Accuracy (EX_{SOFT}) is the normalized overlap of returned results. **Bold** results are best on the test set and underlined results are best on the hard partition.

ilar instances from the training set as in-context examples is consistently better than a random selection. We also see that retrieval by sentence-BERT embedding similarity is better than retrieval by BLEU score. Best results are obtained for GPT-4 with sBERT retrieval, which will simply be referred to as *GPT-4* in the following.

6.3 Comparison between Finetuning and In-Context Learning

In the previous sections, we selected the best models for finetuning and for in-context learning, based on development set performance. In order to compare the two models, we present results on the test set in Table 3 (top). While the surface metrics OQS and EM are nearly identical for both models, GPT-4 significantly outperforms OverpassT5 in the execution based metrics. It is also interesting that the queries generated by GPT-4 are more likely to raise syntax errors (#Errors), despite achieving higher execution accuracy. Inspecting the individual components of OQS reveals that GPT-4 is better at generating correct key-value pairs, indicating that they are more important for correct execution results than faithfulness to the syntax of the reference query. We conjecture that the reason for this is that GPT-4 has likely seen a larger amount of OSM key-value pairs during pretraining than the finetuned models that are limited to OSM knowledge acquired from our training set.

Table 3 (bottom) shows the results on the hard partition of the test set (defined in Section 7.1). All performance metrics decrease significantly, without affecting the relative improvement of GPT-4 over OverpassT5. Notably, the majority of syntax errors stem from this partition of the test set.

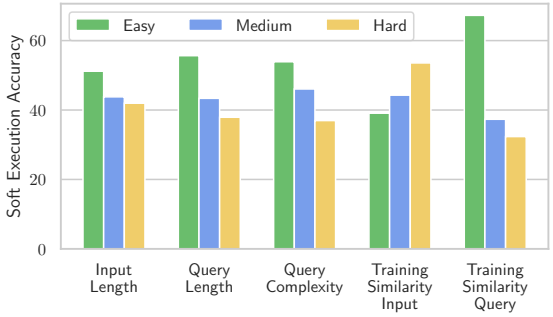


Figure 5: Instance difficulty on the development set using OverpassT5. Dividing the evaluation instances by highest similarity to any training query allows to measure performance on instances with different difficulties.

In sum, while GPT-4 is better at generating queries that return correct results, the OverpassT5 model is better at producing faithful OverpassQL syntax. However, GPT-4's advantage in execution accuracy comes at a computational cost. OverpassT5 is orders of magnitude smaller, resulting in faster inference speed and likely lower monetary cost per query. Also, GPT-4 is a 3rd-party API and does not allow for self-hosting, implying privacy concerns.

7 Analysis

7.1 Instance Difficulty

To better assess the performance of our proposed models, we aim to divide the evaluation instances into three difficulty partitions. Figure 5 displays EX_{SOFT} results for the easy, medium and hard partitions according to different difficulty criteria. A

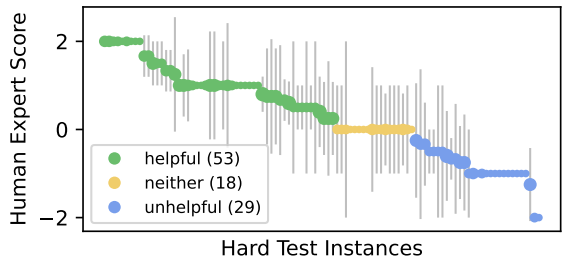


Figure 6: Human expert evaluation for 100 hard instances of the test set. Dot size indicates the number of human evaluators (up to 5 per instance). Grey bars depict the standard deviation.

Model	OQS	EM	#Errors	EX	EX_{SOFT}
GPT-4	**73.2**	**23.4**	24	40.4	53.0
Refine Syntax Errors Only					
no feedback	**73.2**	**23.4**	19	40.4	53.1
with feedback	**73.2**	**23.4**	**7**	40.7	53.7
Refine All Instances					
no feedback	73.1	21.9	31	39.6	52.9
with feedback	73.1	**23.4**	26	**41.4**	**54.5**

Table 4: Self-Refinement of hypotheses generated by GPT-4 on the **development set**. Feedback is either the error message if raised during execution or the returned results.

straightforward difficulty metric like the length of the input text has little influence on the accuracy. Using the length or complexity of the query (measured as the number of syntactic units) as difficulty metric leads to a clearer partition into easy, medium, and hard instances. Surprisingly, partitioning the instances based on their maximum input text similarity to any training instance leads to an undesired negative correlation of EX_{SOFT} and difficulty where the instances with the lowest similarity to training inputs achieve best EX_{SOFT} results. Finally, the clearest partition into instances of different difficulty is achieved by using the maximum similarity of a query to any query in the training set, where query similarity is measured by the OQS metric. A partitioning based on similarity to training queries can be seen as *out-of-distribution* testing since it measures the performance for instances that are less likely to be memorized from the training set. We thus use this criterion to select instances comprising the hard partition in the experiments in Section 6.

7.2 Human Expert Evaluation

One motivation of the Text-to-OverpassQL task is to facilitate a system that assists expert users with crafting queries. While the surface and execution metrics allow us to compare different models, it is difficult to estimate how helpful imperfect queries are to human developers. To this end, we conducted an evaluation of the OverpassT5 outputs by human experts. They are Overpass developers that were recruited by postings in Overpass communities and they participated voluntarily in our experiment. They were asked to rate the helpfulness of a query given the input text on a five-point scale from "very unhelpful" to "very helpful". There were seven experts casting a total of 228 votes across 100 instances of the hard partition. In Appendix F, we explain the survey design in more detail. The results in Figure 6 show that the majority of generated queries were rated helpful, and less than a third were deemed unhelpful. This shows that even for the hardest test instances, the OverpassT5 model generates queries that are mostly helpful for developers when crafting a new query.

7.3 Self-Refinement from Execution Feedback

Recently, studies have shown that LLMs are able to self-refine their own outputs (Madaan et al., 2023). The generated hypothesis is appended to the context and the LLM is prompted to generate an improved version. We conduct self-refine experiments for GPT-4 by appending the generated query and by additionally providing feedback from the query execution. If the query cannot be executed, we use the error message as feedback, otherwise we append a sample of the returned elements to the prompt. Table 4 shows results for self-refinement in two scenarios where either self-refinement is applied to all queries, or only to queries that raised a syntax error during execution. The results show that only refining syntax errors reduces the error count from 24 to 7 if an explicit error message is appended to the prompt, compared to a reduction to 19 errors if only the generated query is appended. However, this only leads to a slight increase in execution accuracy. On the other hand, refining all instances leads to an increase in errors, but also improves EX_{SOFT} by 1.5 points when providing explicit feedback. These experiments show that there is still room for improving query generation with clever prompting techniques. Examples of feedback types and their effect on the generated queries can be found in Appendix E.

8 Conclusion

We introduced a novel semantic parsing task, called Text-to-OverpassQL. The objective of this task is to generate Overpass queries from natural language inputs. We highlighted its relevance within the OpenStreetMap ecosystem and related it to similar tasks. We identified key differences to the Text-to-SQL that pose unique challenges. To facilitate research on the task, we proposed OverpassNL, a dataset of real-world Overpass queries and corresponding annotations with natural language inputs. We used this dataset to train several state-of-the-art models and establish a base performance of the task. In order to measure prediction performance, we proposed task-specific metrics that take the OverpassQL syntax into account and are grounded in database execution. We presented a detailed evaluation of results that reveals the strengths and weaknesses of the considered learning strategies. We hope that our works serves as a foundation for further research on the challenging task of semantic parsing of geographical information grounded in the large and widely used OpenStreetMap database.

Acknowledgements

The research reported in this paper was supported by a Google Focused Research Award on "Learning to Negotiate Answers in Multi-Pass Semantic Parsing". We also thank Martin Raifer for helping us to acquire queries shared by Overpass Turbo users.

References

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. 2023. Teaching large language models to self-debug.

Carolin Haas and Stefan Riezler. 2016. A corpus and semantic parser for multilingual natural language querying of openstreetmap.

Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, Jayant Krishnamurthy, and Luke Zettlemoyer. 2017. Learning a neural semantic parser from user feedback. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 963–973, Vancouver, Canada. Association for Computational Linguistics.

Rohit Kate, Yuk Wong, and Raymond Mooney. 2005a. Learning to transform natural to formal languages. volume 3, pages 1062–1068.

Rohit J. Kate, Yuk Wah Wong, and Raymond J. Mooney. 2005b. Learning to transform natural to formal languages. In *Proceedings of the 20th National Conference on Artificial Intelligence - Volume 3*, AAAI'05, page 1062–1068. AAAI Press.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations (ICLR)*, San Diego, California.

Carolin Lawrence and Stefan Riezler. 2016. Nlmaps: A natural language interface to query openstreetmap.

Carolin Lawrence and Stefan Riezler. 2018. Improving a neural semantic parser by counterfactual learning from human bandit feedback.

Fei Li and H. V. Jagadish. 2014. Constructing an interactive natural language interface for relational databases. *Proc. VLDB Endow.*, 8(1):73–84.

Xi Victoria Lin, Richard Socher, and Caiming Xiong. 2020. Bridging textual and tabular data for cross-domain text-to-sql semantic parsing. *CoRR*, abs/2012.12627.

Jiachang Liu, Dinghan Shen, Yizhe Zhang, Bill Dolan, Lawrence Carin, and Weizhu Chen. 2022. What makes good in-context examples for GPT-3? In *Proceedings of Deep Learning Inside Out (DeeLIO 2022): The 3rd Workshop on Knowledge Extraction and Integration for Deep Learning Architectures*, pages 100–114, Dublin, Ireland and Online. Association for Computational Linguistics.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: Iterative refinement with self-feedback.

OpenAI. 2023. Gpt-4 technical report. *ArXiv*, abs/2303.08774.

OpenStreetMap Foundation. 2019. Who uses openstreetmap? | openstreetmap. [Online; accessed 24-July-2023].

OpenStreetMap Wiki. 2022. Stats — openstreetmap wiki,. [Online; accessed 24-July-2023].

OpenStreetMap Wiki. 2023. Overpass api/overpass ql — openstreetmap wiki,. [Online; accessed 24-July-2023].

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

Maja Popović. 2015. chrF: character n-gram F-score for automatic MT evaluation. In *Proceedings of the Tenth Workshop on Statistical Machine Translation*, pages 392–395, Lisbon, Portugal. Association for Computational Linguistics.

Mohammadreza Pourreza and Davood Rafiei. 2023. Din-sql: Decomposed in-context learning of text-to-sql with self-correction.

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

Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. 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 3982–3992, Hong Kong, China. Association for Computational Linguistics.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools.

Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021. PICARD: Parsing incrementally for constrained auto-regressive decoding from language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 9895–9901, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Ruoxi Sun, Sercan O. Arik, Hootan Nakhost, Hanjun Dai, Rajarishi Sinha, Pengcheng Yin, and Tomas Pfister. 2023. Sql-palm: Improved large language model adaptation for text-to-sql.

Lappoon R. Tang and Raymond J. Mooney. 2001. Using multiple clause constructors in inductive logic programming for semantic parsing. In *Machine Learning: ECML 2001*, pages 466–477, Berlin, Heidelberg. Springer Berlin Heidelberg.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. Llama: Open and efficient foundation language models.

Yue Wang, Weishi Wang, Shafiq Joty, and Steven C.H. Hoi. 2021. Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021*.

Linting Xue, Aditya Barua, Noah Constant, Rami Al-Rfou, Sharan Narang, Mihir Kale, Adam Roberts, and Colin Raffel. 2022. ByT5: Towards a token-free future with pre-trained byte-to-byte models. *Transactions of the Association for Computational Linguistics*, 10:291–306.

Tao Yu, Chien-Sheng Wu, Xi Victoria Lin, bailin wang, Yi Chern Tan, Xinyi Yang, Dragomir Radev, richard socher, and Caiming Xiong. 2021. Gra{pp}a: Grammar-augmented pre-training for table semantic parsing. In *International Conference on Learning Representations*.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2018. Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Brussels, Belgium.

John M. Zelle and Raymond J. Mooney. 1996. Learning to parse database queries using inductive logic programming. In *AAAI/IAAI*, pages 1050–1055, Portland, OR. AAAI Press/MIT Press.

Luke S. Zettlemoyer and Michael Collins. 2005. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence*, UAI'05, page 658–666, Arlington, Virginia, USA. AUAI Press.

Rui Zhang, Tao Yu, Heyang Er, Sungrok Shim, Eric Xue, Xi Victoria Lin, Tianze Shi, Caiming Xiong, Richard Socher, and Dragomir Radev. 2019. Editing-based SQL query generation for cross-domain context-dependent questions. 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)*, Hong Kong, China.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2sql: Generating structured queries from natural language using reinforcement learning. *CoRR*, abs/1709.00103.

A Key Usage

There are 90k unique keys in OpenStreetMap of which only 6k are actively used[6]. We define active usage as 1k or more elements tagged with that key. Keys that are used less than 1k are often include typos, are outdated or leaked from proprietary tagging schemes. Figure 7 shows that the 1k unique keys in our dataset account for 91% of all key usage in OpenStreetMap. Key Usage is the total number of times any key is used to tag an element.

B Model Details

In the finetuning experiments, we use CodeT5[7] and ByT5[8] provided by huggingface. We finetune all weights (full finetuning) for 30 epochs using the Adam (Kingma and Ba, 2015) optimizer with weight decay of 0.1. The maximum learning rate is $4xs10^{-4}$ with warmup for 10% of the steps and a linear decay schedule. Training batch size is 16 and we decode with four search beams. For in-context learning, we use the official OpenAI API to access GPT-3 (*text-davinci-003*) and GPT-4 (*gpt-4-0314*).

C Syntax Features

We count the occurrences of major OverpassQL syntax features used in the queries of our dataset. We use the OverpassQL article[9] in the OpenStreetMap Wiki to categorize features and list their prevalence in our dataset in Figure 9. In total 31 of the 41 syntax features occur in at least 20 queries of our dataset. This represents a coverage of 76%. The "Evaluators" category is a subcategory of "Conditional Query Filter" and we do not count each evaluator type as major feature.

D Prompts

The structure of the 5-shot prompt is shown in Figure 10 and the prompt for self refinement is shown in Figure 11.

E Refine+Feedback

Examples of different feedback types (Empty Output, Error Message or Normal Output) and the effect they have on the generated query can be found in Figure 12.

[6] https://taginfo.openstreetmap.org/keys

[7] https://huggingface.co/Salesforce/codet5-base

[8] https://huggingface.co/google/byt5-base

[9] https://wiki.openstreetmap.org/w/index.php?title=Overpass_API/Overpass_QL&oldid=2568570

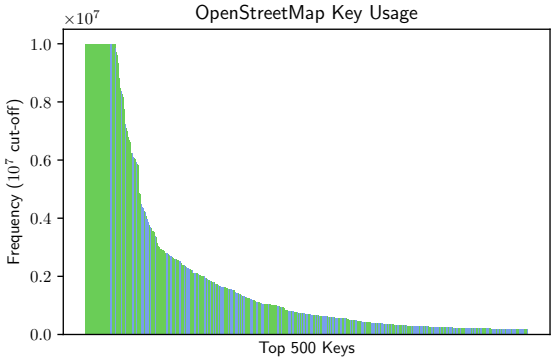


Figure 7: Top 500 keys by usage in OpenStreetMap. Green indicates that the key is used in our dataset and blue that it is not. The keys in our dataset cover 91% of keys tagged across the entire OSM database.

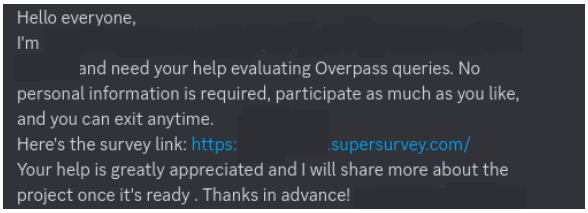


Figure 8: Post to announce our survey on the OpenStreetMap discord.

F Human Expert Survey

The link to the survey was posted on the Discord community called "Openstreetmap World" and the official OpenStreetMap Slack server. Both have a dedicated "Overpass" channel that we posted to. Figure 8 shows the survey announcement and Figure 13 shows the survey interface including the answer options and our definition of helpfulness.

G Annotation Interface

We show the annotation interface in Figure G. The annotators see the query on the top. The results of the query are visualized in the middle and the raw outputs are below. The text box on the bottom is used by the annotators to write the natural language input for the given query.

- Settings (5/6)
  - Timeout (5,985; 71.7%)
  - Element Limit (125; 1.5%)
  - Output Format (6,945; 83.2%)
  - Bounding Box (2,746; 32.9%)
  - Date (286; 3.4%)
  - Diff/adiff two dates (0; 0%)
- Block Statements (4/8)
  - Union (7,651; 91.6%)
  - Difference (231; 2.8%)
  - if (5; 0.1%)
  - for-each (0; 0%)
  - for (85; 1.0%)
  - complete (20; 0.2%)
  - retro (1; 0.0%)
  - compare (0; 0%)
- Standalone Statements (9/13)
  - out (8,186; 98.0%)
  - Item (1,115; 13.4%)
  - Recurse Up (30; 0.4%)
  - Recurse Up Relations (6; 0.1%)
  - Recurse Down (6,084; 72.8%)
  - Recurse Down Relations (54; 0.6%)
  - is_in (22; 0.3%)
  - timeline (0; 0%)
  - local (0; 0%)
  - convert (8; 0.1%)
  - make (115; 1.4%)
  - The Query Statement (8,313; 99.5%)
  - The Query Filter (120; 1.4%)
- Filters (12/14)
  - By Tag (7,860; 94.1%)
  - Bounding Box (2,548; 30.5%)
  - Recurse By nwr(402; 4.8%)
  - Recurse By Way Count (0; 0%)
  - By Input Set (8,206; 98.3%)
  - By Element Id (3,571; 42.8%)
  - Relative to Other Elements (389; 4.7%)
  - By Polygon (8; 0.1%)
  - Newer (286; 3.4%)
  - By Date of Change (345; 4.1%)
  - By User (611; 7.3%)
  - By Area (4,846; 58.0%)
  - Area Pivot (36; 0.4%)
  - Conditional Query Filter (236; 2.8%)

Figure 9: List of Overpass Query Language syntax features and the prevalence in our dataset.

The OverpassQL language allows one to formulate questions to the OpenStreetMap database.

Here are a few examples:

Input:
All historic castles in Germany.

Overpass Query:
```
[out:xml][timeout:500];area["name"="Deutschland"]["admin_level"]->.a;
(node["historic"="castle"](area.a);way["historic"="castle"](area.a);
relation["historic"="castle"](area.a););
```

Input:
Find every castle in Luxemburg,Neatherlands and Belgium.

Overpass Query:
```
[out:json][timeout:120];(({{geocodeArea:"Belgium"}}->.be;{{geocodeArea:"Luxembourg"}}->.lu;
{{geocodeArea:"Nederland"}}->.nl;)->.benelux;
(node["historic"="castle"]["name"](area.benelux);););out center;
```

Input:
Castles in current view.

Overpass Query:
```
[out:json][timeout:25];(node["historic"="castle"]({{bbox}});way["historic"="castle"]({{bbox}});
relation["historic"="castle"]({{bbox}}););out;>;out skel qt;
```

Input:
Castles in current view.

Overpass Query:
```
[out:json][timeout:25];(node["historic"="castle"]({{bbox}});
way["historic"="castle"]({{bbox}});relation["historic"="castle"]({{bbox}}););
out;>;out skel qt;
```

Input:
castles in Tuscany.

Overpass Query:
```
[out:json][timeout:250];{{geocodeArea:"Tuscany"}}->.searchArea;
(node["historic"="castle"](area.searchArea);way["historic"="castle"](area.searchArea);
relation["historic"="castle"](area.searchArea););out;>;out skel qt;
```

Input:
castle in Deutschland

Overpass Query:
<>

Figure 10: The **5-shot prompt** used to generate Overpass queries from natural language input. The in-context examples are selected from our training set by similarity to the current natural language input. We use sBERT to embed the natural language input and compute the cosine similarities. The current natural language input is in *cursive* and <> indicates the position where the LLM starts to generate next tokens.

The OverpassQL language allows one to formulate questions to the OpenStreetMap database. Your goal is, given an Input and a Hypothesis, to produce a improved version of the Hypothesis. If the Hypothesis is already good enough, do not try to improve it.

Here are a few examples:

Input:
atms in Germany

Hypothesis:
```
[out:json][timeout:25];area["name"="Germany"]->.a;(node["amenity"="atm"](area.a);
way["amenity"="atm"](area.a);relation["amenity"="atm"](area.a););
out;>;out skel qt;
```

Overpass Query:
```
[out:json][timeout:25];{{geocodeArea:"Deutschland"}}->.searchArea;
(node["amenity"="atm"](area.searchArea);way["amenity"="atm"](area.searchArea);
relation["amenity"="atm"](area.searchArea););out center;
```

Input:
ATMs and banks with ATMs in Berlin.

Hypothesis:
```
[out:json][timeout:25];{{geocodeArea:"Berlin"}}->.searchArea;
(node["amenity"="atm"](area.searchArea);node["amenity"="bank"]["atm=yes"](area.searchArea););
out;out;>;out skel qt;
```

Overpass Query:
```
[out:json][timeout:25];
area["name"="Berlin"]->.a;(node["amenity"="bank"]["atm"="yes"](area.a);
node["amenity"="atm"](area.a);way["amenity"="bank"]["atm"="yes"](area.a);>;
way["amenity"="atm"](area.a);>;);out;
```

... three more in-context examples ...

Here is an Input:
Banks or ATMS in Bürggen of Kreis Viersen.

Here is the Overpass Query Hypothesis produced by a model:

```
[out:json][timeout:25]; {{geocodeArea:"Bürggen"}}->.searchArea;
{{geocodeArea:"Kreis Viersen"}}->.searchArea2;
(node["amenity"="bank"](area.searchArea)(area.searchArea2);
node["amenity"="atm"](area.searchArea)(area.searchArea2);
way["amenity"="bank"](area.searchArea)(area.searchArea2);
relation["amenity"="bank"](area.searchArea)(area.searchArea2); ); out; >; out skel qt;
```

You will now get part of the Overpass result produced after using the generated Overpass Query Hypothesis. An error means that you should definitely improve on the Hypothesis. A normal result could mean that the Overpass Query is good enough, if the output fits to the asked query:

```
No Results found.
```

Improve on the Overpass Query or keep it if it is good enough:
<>

Figure 11: The **5-shot refine with feedback prompt** used to generate Overpass queries from natural language input. In this case, the execution of the hypothesis did not return any results.

Question: Cities and towns that are between 100 and 300 km away from Moscow

Hypothesis:

```
[out:json][timeout:25];(node["place"="city"](around:300000,{{geocodeCoords:"Moscow"}});
node["place"="town"](around:300000,{{geocodeCoords:"Moscow"}}); ); ->.all;
(.all; - node["place"~"city|town"](around:100000,{{geocodeCoords:"Moscow"}}););out center;
```

Feedback:

```
Error: line 1: parse error: Unknown type "->"
```

New Hypothesis:

```
[out:json][timeout:25]; ( node["place"="city"](around:300000,{{geocodeCoords:"Moscow"}});
node["place"="town"](around:300000,{{geocodeCoords:"Moscow"}}); ) ->.all;
( .all; - node["place"~"city|town"](around:100000,{{geocodeCoords:"Moscow"}}););out center;
```

Question: All subway stations in the current view

Hypothesis:

```
[out:json][timeout:25];(
node["railway"="subway_entrance"]({{bbox}});
way["railway"="subway_entrance"]({{bbox}});
relation["railway"="subway_entrance"]({{bbox}});
);out;>;out skel qt;
```

Feedback:

```
{'type': 'node', 'id': 486093610, 'lat': 31.3242632, 'lon': 121.5273796, 'tags': {'railway':
     'subway_entrance', 'ref': '3'}}
```

New Hypothesis:

```
[out:json][timeout:25];(
node["railway"="station"]["subway"="yes"]({{bbox}});
way["railway"="station"]["subway"="yes"]({{bbox}});
relation["railway"="station"]["subway"="yes"]({{bbox}});
);out;>;out skel qt;
```

Question: pharmacy named Apteka in Wrocław

Hypothesis:

```
[out:json][timeout:120];{{geocodeArea:"Wrocław"}}->.searchArea;(
node["amenity"="pharmacy"]["name"="Apteka"](area.searchArea);
way["amenity"="pharmacy"]["name"="Apteka"](area.searchArea);
relation["amenity"="pharmacy"]["name"="Apteka"](area.searchArea);
);out;>;out meta qt;
```

Feedback: No Results found.

New Hypothesis:

```
[out:json][timeout:120];{{geocodeArea:"Wrocław"}}->.searchArea;(
node["amenity"="pharmacy"]["name"~"Apteka", i](area.searchArea);
way["amenity"="pharmacy"]["name"~"Apteka", i](area.searchArea);
relation["amenity"="pharmacy"]["name"~"Apteka", i](area.searchArea);
);out;>;out meta qt;
```

Figure 12: Examples where the feedback from query execution led to an improved version of the query. **Top:** The execution of the query raises a syntax error. The symbol "->" is not allowed to occur immediately after a semicolon. This is fixed in the query generated after feedback. **Middle:** The execution of the query returns results that we append as feedback. However, the result contains the tag subway_entrance which does not align with the requested subway stations. The refined query then asks for stations instead of subway_entrances. **Bottom:** The execution of the hypothesis does not return any results. Given this information, the refined query uses a case-insensitive and partial matching search instead of a strict match.

Text Input:

restaurants in a radius of 1000 meters around New York City

Generated Overpass Query:

```
[out:json]
;
node["name"="New York City"];
(
  node
    ["amenity"="restaurant"]
    (around:1000);
  way
    ["amenity"="restaurant"]
    (around:1000);
  relation
    ["amenity"="restaurant"]
    (around:1000);
);
out;
>;
out skel;
```

↑ Please rate how helpful the generated Overpass query is based on the given text input. A query can be helpful even if it is not entirely correct. It should serve as a template that can be refined further with little work. A query is unhelpful if it requires substantial editing to fix it.

○ Very unhelpful	○ Somewhat unhelpful	○ Neither helpful nor unhelpful	○ Somewhat helpful	○ Very helpful

Figure 13: Interface of the expert survey. At the bottom are the answer options and our definition of helpfulness.

Query:

```
[out:json][timeout:25];
(
  way["leisure"="park"]({{bbox}});
  relation["leisure"="park"]({{bbox}});

  way["leisure"="playground"]({{bbox}});
  relation["leisure"="playground"]({{bbox}});

  way["landuse"="grass"]({{bbox}});
  relation["landuse"="grass"]({{bbox}});
);

out body;
>;
out skel qt;
```

Hint: Sometimes there will be no map because no results were returned.

Query Result:

```
  <nd ref="3492040623"/>
  <nd ref="11005241901"/>
  <nd ref="3492040624"/>
  <tag k="leisure" v="playground"/>
  <tag k="wheelchair" v="yes"/>
 </way>
 <way id="418603807">
  <nd ref="2709148007"/>
  <nd ref="4189883003"/>
  <nd ref="4189883002"/>
  <nd ref="4189883001"/>
  <nd ref="2709148007"/>
  <tag k="leisure" v="playground"/>
 </way>
```

Additional information on the tags used in this example can be found here:

https://wiki.openstreetmap.org/wiki/Tag:leisure%3Dplayground
https://wiki.openstreetmap.org/wiki/Key:landuse
https://wiki.openstreetmap.org/wiki/Tag:landuse%3Dgrass
https://wiki.openstreetmap.org/wiki/Key:leisure
https://wiki.openstreetmap.org/wiki/Tag:leisure%3Dpark
https://wiki.openstreetmap.org/wiki/Key:timeout:25

Please enter the most specific natural language description of the information need expressed in the Overpass query:

Where are parks, playgrounds or grass areas in the current view?

Submit

Figure 14: Annotation interface to collect the natural language descriptions for the queries in our dataset.