Heidelberg University
Faculty of Modern Languages
Department of Computational Linguistics

M.A. Thesis

# An Online Learning System for Parsing and Answering Geographical Queries in Natural Language

Simon Will

4 May 2021

Berliner Straße 109a
69121 Heidelberg
simon.will@gorgor.de

Supervisor: Prof. Dr. Stefan Riezler
Second assessor: Prof. Dr. Katja Markert

# Contents

# List of Figures

# List of Tables

# Abstract

OpenStreetMap (OSM) stores a large amount of data useful for everyday tasks as well as for informational queries, but it is difficult to query without using specialized applications or being versed in special OSM query languages. The existing NLMaps v2 dataset can be used for building a question answering system that translates a natural language query into a machine-readable query used for extracting the answer from the OSM database, but the performance of parsers trained on that dataset has been very limited when tested on new queries.

In this thesis, we analyze NLMaps v2 and find several shortcomings. After fixing them, we extend the dataset by generating new, linguistically diverse queries with a probabilistic templating approach, which we then use to train a new parser that significantly outperforms parsers presented in previous work.

In order to make our parser accessible, we build a web interface for asking queries, which can also be used to correct wrong parses and which is capable of learning from the corrections in an online fashion. We use the new interface to hire users to issue queries and to correct the parses, thus creating the first large NLMaps dataset consisting of real user queries. This dataset is used in online learning simulation experiments in order to find the most effective approach to learn from new feedback.

# Abriss (German Abstract)

OpenStreetMap (OSM) speichert riesige Mengen an Daten, die nützlich sind, um alltägliche und weniger alltägliche Fragen zu beantworten. Doch für exakte Abfragen sind in der Regel entweder auf Einzelfragen spezialisierte Software oder Kenntnisse in speziellen OSM-Anfragesprachen notwendig. Allerdings kann mit dem bestehenden Korpus NLMaps v2 ein Parsing-System trainiert werden, das natürlichsprachliche Anfragen in eine maschinenlesbare Anfrage übersetzt, mithilfe derer dann die Antwort auf die Frage in der OSM-Datenbank gefunden werden kann. Die Zuverlässigkeit von auf diesem Korpus trainierten Parsing-Systemen war aber bisher sehr begrenzt.

In dieser Arbeit analysieren wir das Korpus NLMaps v2 und stellen verschiedene Mängel fest. Nach dem Beheben dieser Mängel erweitern wir das Korpus um neue, linguistisch diverse Anfragen, die mit einem probabilistischen Mustervorlagensystem erstellt werden. Das erweiterte Korpus nutzen wir dann, um ein neues Parsing-System zu trainieren, das die bisher in der Literatur vorgestellten Systeme deutlich übertrifft.

Unser Parsing-System machen wir in einem neuen Web-Interface zugänglich, das sowohl dazu verwendet werden kann, Fragen zu stellen, als auch dazu, falsche Antworten zu korrigieren. Das Web-Interface ist außerdem dazu in der Lage, das Parsing-System auf korrigierten Beispielen online weiter zu trainieren. Wir nutzen es in dieser Arbeit auch, um mithilfe von Studienteilnehmern das erste große NLMaps-Korpus zu erstellen, das aus Anfragen verschiedener Menschen besteht. Dieses Korpus wird in dieser Arbeit zudem dazu verwendet, in verschiedenen Online-Lern-Simulationen den besten Ansatz zu finden, von neuen Rückmeldungen zu lernen.

# 1. Introduction

The OpenStreetMap project's database stores a wealth of geographical information about the world ranging from map-typical features like borders, streets and buildings to detailed information about points of interest like shops, sights and recreational areas. However, extracting specific information – such as the answer for the simple natural language question *'Which Italian restaurants in Berlin are wheelchair-accessible?'* – requires either purpose-built tooling or knowledge of custom OpenStreetMap query languages. A natural language interface to the database would make the available information more accessible.

Important groundwork for such a natural language interface was laid by Haas and Riezler (2016a), who developed a simple machine-readable query language (MRL) for OpenStreetMap and also released the dataset *NLMaps*, which maps natural language (NL) queries to their corresponding counterpart in that query language. This dataset makes it possible to train a semantic parser that parses NL into MRL queries.

Despite various models producing good results on the NLMaps test set, their performance does not suffice for practical usage on new real world queries. Investigating that problem, this thesis reviews previous work on NLMaps and reveals shortcomings in the published dataset. In order to improve on it, the existing dataset is overhauled by eliminating some of the identified shortcomings and by extending it through the use of probabilistic templates to include more diverse queries on both the natural language and the machine-readable language side.

The improved NLMaps dataset is used to train an improved parsing model, which is exposed via a new web interface so that it can be used for both asking queries and correcting a parse if the model gets it wrong. In order to further improve the model, the system is able to directly learn from the corrected queries using an online learning technique.

The new web interface is employed in an annotation experiment with users from various backgrounds who use it to ask new queries and to correct errors. The data collected in this way is used to test the online learning setup and is also released as a new NLMaps dataset.

The web interface and all datasets published in this work are available at `https://nlmaps.gorgor.de/`. The following code repositories are associated with this thesis and are tagged with the Git tag `thesis` to mark the state they are in as of the thesis's publishing date:

- `https://gitlab.cl.uni-heidelberg.de/will/nlmapsweb/`: The web interface.
- `https://gitlab.cl.uni-heidelberg.de/will/joeynmt-server/`: The backend server handling parsing and training the parser.
- `https://gitlab.cl.uni-heidelberg.de/will/nlmaps-tools/`: A package containing various tools for working with NLMaps datasets, for generating the NLMaps v3 version and for interpreting an MRL query and retrieving an answer.
- `https://gitlab.cl.uni-heidelberg.de/will/nlmaps-ma/`: This thesis along with scripts for data analysis and plotting.
- `https://github.com/Simon-Will/joeynmt`: A slightly modified version of Joey NMT (Kreutzer, Bastings, et al. 2019).
- `https://github.com/Simon-Will/osm-python-tools`: A slightly modified version of OSMPythonTools (Mocnik 2017). A pull request to the upstream repository is still open.

# 2. Related Work

## 2.1. Question Answering by Semantic Parsing

With the advent of the computer age, there also arose interest in leveraging the digitally stored information for automatically answering natural language questions with first systems being developed as early as the 1960s (e.g. Green et al. 1961). The field of question answering (QA) can be genereally divided into *open-domain QA*, which concerns itself with answering arbitrary questions based on large quantities of natural language text or other unstructured information by employing techniques from information retrieval, and *knowledge-based QA*, which takes advantage of structured information in order to answer questions about specific information that is stored in that knowledge base (cf. Mollá and Vicedo 2007; Jurafsky and Martin 2021). The work in this thesis is an instance of the latter.

In the most common variant of knowledge-based QA, a natural language query is parsed into a machine-readable logical form for representing the meaning (*semantic parsing*), which can then be used to extract the answer from the knowledge base. Notable datasets for this task include the Air Travel Information System (ATIS) dataset (Hemphill et al. 1990), which maps 5280 questions about flights in the USA to a representation in SQL, and GeoQuery, which maps 877 questions about US geography to lambda-calculus based representations in Prolog (Zelle and Mooney 1996). While these two are small datasets specialized on one domain each, the crowd-sourced WikiSQL dataset introduced by Zhong et al. (2017) comprises 80 654 questions on numerous different databases, but all of their queries are very simple and include no advanced SQL operators.

With their dataset called Spider, Yu et al. (2018) introduced the first text-to-SQL dataset that features a large number of different tables as well as complex SQL queries. They present initial results on their dataset using sequence-to-sequence models like attention-based RNNs as well as more rigid slot-filling models based on SQLNet (Xu et al. 2017), which fill pre-defined places in the SQL query. The latter perform better in their experiments.

Since Hwang et al. (2019) showed that using BERT (Devlin et al. 2019) for encod-

ing query and database schema in text-to-SQL task is highly effective on WikiSQL, most recent systems use models based on BERT also on the Spider task (Shaw et al. 2020; Wang et al. 2020; Lin et al. 2020) with some even taking advantage of the content of the database (Wang et al. 2020; Lin et al. 2020) to further improve performance.

## 2.2. Online Learning and Domain Adaptation

### 2.2.1. Online Learning

Neural machine translation models $p_\theta$ are usually (Stahlberg 2020, p. 376) trained with the goal of minimizing the expected cross-entropy loss on the training set $(X, Y)_{\text{train}}$ consisting of a set of source sentences $X$ and a set of target sentences $Y$ by taking steps $\Delta\theta$ in the opposite direction of the gradient:

$$\theta^* = \arg\min_\theta \mathbb{E}_{(x,y)\sim(X,Y)_{\text{train}}} - \log p_\theta(y|x) \tag{2.1}$$

$$\Delta\theta \propto \nabla \mathbb{E}_{(x,y)\sim(X,Y)_{\text{train}}} - \log p_\theta(y|x) \tag{2.2}$$

$$\propto \mathbb{E}_{(x,y)\sim(X,Y)_{\text{train}}} -\nabla \log p_\theta(y|x) \tag{2.3}$$

This explicit method – also called *batch gradient descent* as an instance of *batch learning* (Goodfellow et al. 2016, p. 275; Murphy 2021, p. 100) – requires making a prediction for every instance of the training set, which makes it prohibitively computationally expensive. Instead, *stochastic gradient descent* (SGD) – as an instance of *online learning* (Goodfellow et al. 2016, p. 275; Murphy 2021, p. 100) – approximates the gradient by sampling a single instance $(x, y)$ from the training set and updating the parameters based on this instance:

$$(x, y) \sim (X, Y)_{\text{train}} \tag{2.4}$$

$$\Delta\theta \propto -\nabla \log p_\theta(y|x) \tag{2.5}$$

$$\tag{2.6}$$

With SGD, the model parameters can be updated much more often, but the variance of the gradient is also larger. In practice, minibatches are used as a compromise when training a model on an already-prepared dataset. However, in the case where the dataset becomes available only one instance at a time, the online learn-

ing setup is still useful. A prominent usecase is post-editing of machine translation outputs. Following the definition of Ortiz-Martínez (2016), online learning from post-editing can be operationalized like this:

1. An MT system receives a source sentence $x$.
2. The system makes a prediction $\hat{y}$ for the target sentence.
3. A user reviews $\hat{y}$, adjusts it and presents the system with the correct translation $y$.
4. The MT system is updated by learning from $y$.

This procedure is especially useful for adapting a system pre-trained on general data to another domain. In statistical machine translation (SMT), online learning research has first focused on adjusting the weights of the log-linear model (e.g. Liang et al. 2006; Arun and Koehn 2007; Watanabe et al. 2007). Later, Ortiz-Martínez et al. (2010) and Ortiz-Martínez (2016) employed online learning in the scenarios of post-editing and interactive machine translation (IMT; Casacuberta et al. 2009; Barrachina et al. 2009) for adjusting also the model features themselves.

For NMT, Turchi et al. (2017) simulated online learning with an attentional encoder-decoder network finding that simply updating by doing an SGD step with the post-edited instance is superior to a more complicated approach in which they added an additional training step after receiving each source sentence but *before* predicting the target sentence. They also found that doing five training iterations per post-edited instance is actually worse than doing only one iteration. Very interestingly, vanilla SGD outperforms Adagrad, Adadelta and Adam in their experiments.

At the same time, Peris, Cebrián, et al. (2017) performed very similar work, in which they compared gradient descent optimization algorithms with more complicated passive-aggressive update rules using subgradient methods. Gradient descent optimizers performed significantly better, although vanilla SGD was inferior to adaptive gradient update rules, out of which Adadelta and Adam performed best.

Peris and Casacuberta (2019) continued this work for post-editing and IMT scenarios and found Adadelta to be the best and also the most stable optimizer with regard to varying the learning rate. They demonstrate that online learning can be successfully used to enhance the performance not only of out-of-domain systems, but also of systems pre-trained with a limited amount of in-domain data.

While the work of Turchi et al. (2017), Peris, Cebrián, et al. (2017), and Peris and Casacuberta (2019) was done using simulations, Karimova et al. (2018) conducted an actual experiment with students of translation studies post-editing NMT outputs, in which they found that online adaptation was able to improve both BLEU score and key-stroke and mouse-action ratio (for KSMR cf. Barrachina et al. 2009).

### 2.2.2. Learning from Weak User Feedback

A related line of NMT research attempts to improve a pre-trained system with a weaker form of user feedback: Instead of a user post-editing a target sentence, they can give feedback in the form of scalar rewards about the quality of a translation or even specific parts of it, e.g. by rating it from 0 to 10. This has the advantage of requiring less user effort and skill, but it is more difficult for a model to correct its predicted translation due to the lack of a gold translation. This poses a particular challenge in tasks with a very large output space like machine translation.

Scalar feedback can be leveraged with reinforcement learning techniques, which was done by Kreutzer, Sokolov, et al. (2017), who simulated user feedback with sentence-level gGLEU (Wu et al. 2016) obtained from out-of-domain reference translations. In their comparison of domain adaptation via fully-supervised fine-tuning and learning from the gGLEU "rewards" using expected loss minimization (which is in essence the REINFORCE algorithm by Williams (1992)), they found that learning from weak feedback is a viable method of domain adaptation, even though the fully-supervised approach naturally yielded better results. They also showed that performance on the original data used for pre-training deteriorated in both scenarios, but more so with fully-supervised fine-tuning. Nguyen et al. (2017) employ the advantage actor-critic algorithm (Mnih et al. 2016) in a similar scenario with one difference of introducing noise and skew in their reward signals in order to simulate actual user feedback. They also do not perform domain adaptation, but *fine-tune* on a separate part of the training set.

In contrast to those systems, which simulate a sequence-level reward signal after predicting a complete translation, Lam et al. (2018) use an advantage actor-critic NMT system in an IMT-style setting where feedback is simulated for *partial* translations when the NMT model is uncertain about its prediction. Unsurprisingly, they observed that the more granular feedback yields improvements over the sentence-level feedback used in Nguyen et al. (2017).

Online methods are of limited use in production systems where the risk of deteriorating performance cannot be taken, especially in the light of possibly adversarial actors. Therefore, user feedback is usually logged for later learning, which is un-

problematic for post-edits since they can serve as self-contained training instances not much different from reference translations in specially-prepared training sets. However, weaker user feedback in the form of ratings is tied to the system's original prediction, which complicates learning from it with another system. Lawrence, Sokolov, et al. (2017) used a control-variate-based smoothing technique called *deterministic propensity matching* to leverage this kind of logged feedback for offline learning with an SMT system. Later, Lawrence and Riezler (2018) applied this approach also to improving an NMT-based parser on the NLMaps v2 dataset, which is discussed later in this thesis. They were able to learn from both sequence- and token-based feedback (both simulated and actual human), but the more granular token-based feedback proved superior.

Kreutzer, Berger, et al. (2020) conducted an offline learning NMT experiment with human annotators who corrected wrong translations in one scenario and only marked erroneous passages in another scenario. They found comparable improvements over the baseline in both scenarios, but the error markings took significantly less effort to collect.

For an overview of human feedback in reinforcement learning for NLP, see the work of Kreutzer, Riezler, et al. (2020).

### 2.2.3. Domain Adaptation in Semantic Parsing

Domain Adaptation is especially interesting for custom semantic parsing systems on new domains since creating custom datasets involves a lot of expensive annotation work resulting in small dataset sizes. The goal here is pre-training a model on a larger dataset that is similar to the custom one in order to reduce the needed size of the custom dataset or to simply improve performance. Kennardi et al. (2019) pre-trained an attention-based RNN on the ATIS dataset and then fine-tuned the pre-trained model on subsets of the GeoQuery dataset showing that especially for small subsets (i.e. small target domain datasets) pre-training on ATIS improved the performance.

Instead of pre-training a semantic parsing model on a larger semantic parsing dataset, Chen et al. (2020) employ a system that takes advantage of pre-trained language representations in the form of BART (Lewis et al. 2020), a BERT-inspired encoder-decoder setup trained by denoising artificially corrupted text. They show that the BART-based parser incorporating pre-trained language representations outperforms an LSTM-based (Hochreiter and Schmidhuber 1993) parser trained from scratch – an effect that is especially pronounced when only a subset of the data from the target domain is used.

## 2.3. OpenStreetMap Query Systems

### 2.3.1. OpenStreetMap and its Ecosystem

In a crowd-sourcing approach similar to Wikipedia's, the OpenStreetMap[1] (OSM) project aims to create a map of the world by letting users contribute missing data, ranging from low-granular objects like forests or streets to high-granular objects like benches or information boards and even including non-geographical information like opening hours of stores or the types of cuisine available in restaurants. The OpenStreetMap Foundation makes the map data available under the Open Data Commons Open Database License[2] effectively allowing the usage of the data for any project but requiring that any extensions of the data are shared under the same license again.

OpenStreetMap data is made up of three different elements: Nodes, ways (ordered lists of nodes) and relations (groups of elements). The elements' meaning is derived from the tags that are added to them. For instance, an Italian restaurant that has vegan options and is wheelchair-accessible may be tagged as a node with the following tags:

- `amenity=restaurant`
- `cuisine=italian`
- `diet:vegan=yes`
- `diet:vegetarian=yes`
- `wheelchair=yes`
- `opening_hours=Mo-Sa 11:30-22:00`
- `website=https://restaurant.example.com/`

The OpenStreetMap database can be queried in a number of ways, the most prevalent one of which is via Geocoders such as Nominatim[3]. They allow the database being queried by the name or address of an object in *forward geocoding* or by its geographic coordinates in *reverse geocoding*.

[1] OpenStreetmap Foundation (*OpenStreetMap*). `https://www.openstreetmap.org/about`.
[2] Open Data Commons (*Open Data Commons Open Database License (ODbL)*). `https://opendatacommons.org/licenses/odbl/`.
[3] Hoffmann et al. (*Nominatim*). `https://nominatim.org/`.

For querying by more than name or address, there are two specialized systems: Sophox[4] and the Overpass API [5], which can be used most conveniently via the Overpass Turbo [6] interface. The Overpass API allows queries using an XML-like language or – more prominently – its custom Overpass Query Language (Overpass QL). The question *'Which Italian restaurants in Berlin are wheelchair-accessible?'* could be expressed with the following Overpass Query

Overpass QL for wheelchair-accessible restaurants in Heidelberg

```
(area[name=Heidelberg];) -> .a;
nwr[amenity=restaurant][wheelchair=yes](area.a);
out geom;
```

### 2.3.2. NLMaps

The open license as well as the diverse and rich information available in the data make OSM a promising candidate for the foundation of an information retrieval system about geo-related questions. The first step in this direction was made by Haas and Riezler (2016a), when they released the first version of the NLMaps dataset a "corpus consisting of 2,380 questions about geographical facts that can be answered with the [OSM] database."[7] Each question is provided as a natural language (NL) query in English and in German and as its rendering in a custom machine-readable language (MRL). The dataset can be used to develop a parser for parsing an NL query into its corresponding MRL query, which can then be used to extract the answer to the question from the OSM database.

In two subsequent works, Lawrence[8] and Riezler (2016, 2018) expanded the English part[9] of the dataset to include more NL-MRL pairs and by extension also more word types and OSM tags. Table 2.1 shows key data about the size of the extended dataset. Table 2.2 shows the size of the dataset splits. In contrast to the first version, the NL-MRL pairs in this extended version were created with a templating approach, which made use of a table[10] mapping natural language expressions (such

[4] Astrakhan (*Sophox*). `https://wiki.openstreetmap.org/wiki/Sophox`. The official website at `https://sophox.org` is offline as of December 2020.
[5] Olbricht (*Overpass API*). `https://overpass-api.de/`.
[6] Raifer (*Overpass Turbo*). `https://overpass-turbo.eu/`.
[7] Haas and Riezler (2016b). `https://www.cl.uni-heidelberg.de/statnlpgroup/nlmaps/`.
[8] Carolin Haas changed her name to Carolin Lawrence in 2016.
[9] NLMaps v2 is not available in German.
[10] *Nominatim Special Phrases*. `https://wiki.openstreetmap.org/wiki/Nominatim/Special_Phrases/EN`. It is not known which version of the table was used for generating the NLMaps v2 dataset.

as *restaurant*) to OSM tags (such as `amenity=restaurant`).

| | NLMaps v1 | NLMaps v2 |
|---|---|---|
| Instances | 2380 | 28 609 |
| Tokens | 25 906 | 202 088 |
| Types | 1002 | 8710 |
| Avg. Tokens per NL | 10.88 | 7.06 |
| Distinct Tags | 477 | 6582 |

Table 2.1.: Numeric information about NLMaps v1 and NLMaps v2. The table is reproduced from Lawrence and Riezler (2018).

| Set split | NLMaps v2 |
|---|---|
| train | 16 172 |
| dev | 1843 |
| test | 10 594 |

Table 2.2.: Split sizes in the NLMaps v2 dataset.

In addition to the NL and the MRL queries, the dataset includes a linearized (LIN) version of the MRL query. This is a formally equivalent variant of the MRL query that avoids parentheses and commata by specifying each operator's arity instead. For further information on this, cf. Andreas et al. (2013) and Haas and Riezler (2016a). All parsing models discussed in this thesis parse the NL query into the LIN query, which can be converted into the MRL query for retrieving the result from the OSM database.

All of the question and query variants are also provided in a version where the locations and the points of interest are replaced by generic `_LOCATION` and `_POI` tokens, respectively. This is intended to simplify training a parser model which relies on an external Named Entity Recognition (NER) component for the named entities. Figure 2.1 shows an overview of the available variants.

### NLMaps Evaluation

There are two methods of evaluating a model's predictions on an NLMaps dataset: Comparing the predicted MRLs with the gold MRLs or comparing the results retrieved by interpreting the MRLs. The latter method defines precision, recall and $F_1$ score, where $F_1$ is the measure that is usually reported.

Unmasked MRL

```
query(
  around(
    center(
      area(keyval('name','München')),
      nwr(keyval('name','Super Cut'))
    ),
    search(
      nwr(keyval('amenity','post_box'))
    ),
    maxdist(DIST_INTOWN)
  ),
  qtype(latlong)
)
```

Masked MRL

```
query(
  around(
    center(
      area(keyval('name','_LOCATION')),
      nwr(keyval('name','_POI'))
    ),
    search(
      nwr(keyval('amenity','post_box'))
    ),
    maxdist(DIST_INTOWN)
  ),
  qtype(latlong)
)
```

Unmasked LIN

```
query@2
  around@3
    center@2
      area@1 keyval@2 name@0 München@s
      nwr@1 keyval@2 name@0 €SuperCut@s
    search@1
      nwr@1 keyval@2 amenity@0 post_box@s
    maxdist@1 DIST_INTOWN@0
  qtype@1 latlong@0
```

Masked LIN

```
query@2
  around@3
    center@2
      area@1 keyval@2 name@0 _LOCATION@s
      nwr@1 keyval@2 name@0 _POI@s
    search@1
      nwr@1 keyval@2 amenity@0 post_box@s
    maxdist@1 DIST_INTOWN@0
  qtype@1 latlong@0
```

Figure 2.1.: MRL and LIN queries for the NL query *'Where Post Boxes near Super Cut in München'* in their unmasked and masked form. The instance is taken from NLMaps v2.

$$\text{precision} = \frac{\text{number of correct answers}}{\text{number of MRLs that yield an answer}}$$

$$\text{recall} = \frac{\text{number of correct answers}}{\text{number of all NL-MRL pairs}}$$

$$F_1 = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

This thesis takes the stand that comparing the results of queries is not appropriate because of two major reasons: First, the result of two MRLs may be identical by chance, especially when asking whether something exists or how many of something there are. Second, such an evaluation is dependent on the versions of the OSM database and the version of the software interpreting the MRL making reported scores difficult to reproduce. Additionally, retrieving results takes up a large amount of time and computing resources, which renders this evaluation method unsuitable for validation during training.

Therefore, the models in this thesis are compared using exact match accuracy on MRLs. Theoretically, this method has the problem that MRLs can be semantically

equivalent but syntactically different by switching the order of OSM tags or values (e.g. `keyval('diet:vegetarian', or('yes','only'))` vs. `keyval('diet:vegetarian', or('only,'yes'))`). In practice however, we ensure that all MRLs used for training are in a canonical form by sorting tags and values alphabetically. This way, switched order does virtually not occur in the models of this thesis.

$$\text{accuracy} = \frac{\text{number of system MRLs perfectly matching the gold MRL}}{\text{number of all NL-MRL pairs}}$$

### Previous Results on NLMaps v2

Lawrence and Riezler (2018) trained a GRU-based encoder-decoder model (Cho et al. 2014) with Bahdanau attention (Bahdanau et al. 2015) on NLMaps v2, once without masking the named entities and once with masking them. The model trained on the masked version is accompanied by an NER model. Staniek (2020) trained a similar model for comparison with Lawrence and Riezler (2018), once as a token-based RNN and once as a character-based RNN, both without masking the named entities. The results are shown in Table 2.3. In essence, they show that it is easy enough for the character-based model to copy the named entities from the source to the target so that a separate NER model does not improve the results.

| Model | unmasked | masked + NER |
|---|---|---|
| Lawrence and Riezler (2018) (token) | 0.804 | 0.901 |
| Staniek (2020) (token) | 0.834 | — |
| Staniek (2020) (character) | **0.938** | — |

Table 2.3.: $F_1$ score after retrieving query results of models by Lawrence and Riezler (2018) and Staniek (2020) on NLMaps v2

Even though Staniek achieves an $F_1$ score of 93.8 % and an accuracy of 89.8 %[11], the task of parsing NL queries is not solved, at all. The high performance is the result of a number of shortcomings in the NLMaps v2 dataset, a part of which has already been discussed by Staniek and which are investigated in more detail in Chapter 3.

[11] The accuracy is not reported in the original work, but Staniek kindly made his model available.

# 3. NLMaps Data Improvement

After training a character-based encoder-decoder model as described by Staniek (2020) on the NLMaps v2 dataset, it quickly becomes apparent that the 89.8 % accuracy on the test split is not reflected in the model's performance on new queries. In particular, the model is not robust against unseen wordings of a query and it – more understandably – also fails for unseen OSM tags. Figure 3.1 shows typical errors the model makes on NL queries from NLMaps v4, which is introduced in Section 5.2.

Gold MRL

```
query(
  around(
    center(
      area(keyval('name','Westheim')),
      nwr(
        keyval('name','Martinskirche')
      )
    ),
    search(
      nwr(keyval('amenity','cafe'))
    ),
    maxdist(DIST_INTOWN)
  ),
  qtype(latlong)
)
```

System MRL

```
query(
  around(
    center(
      area(keyval('name','Westheim')),
      nwr(
        keyval('name','Martinskirche')
      )
    ),
    search(
      nwr(keyval('amenity','cafe'))
    ),
    maxdist(DIST_INTOWN)
  ),
  qtype(show:me)
)
```

(a) *'Show me the cafes near Martinskirche in Westheim'*

Gold MRL

```
query(
  around(
    center(
      nwr(keyval('name','Zenica'))
    ),
    search(
      nwr(keyval('natural','valley'))
    ),
    maxdist(DIST_OUTTOWN)
  ),
  qtype(least(topx(1)))
)
```

System MRL

```
query(
  around(
    center(
      area(keyval('name','Paris')),
      nwr(keyval('name','Zenica'))
    ),
    search(
      nwr(keyval('shop,'cenica'))
    ),
    maxdist(DIST_INTOWN)
  ),
  qtype(least(topx(1)))
)
```

(b) *'Is there any valley in the surroundings of Zenica?'*

Figure 3.1.: Selected typical errors of a model trained on NLMaps v2.

## 3.1. Analysis of NLMaps v2

Seven separate issues with the NLMaps v2 dataset can be identified that lead to a subpar performance on new queries not present in the training set or test set.

1. Extremely close resemblance between training set and test set
2. Inconsistencies in mapping from NL term to OSM tag
3. Inconsistencies in MRL syntax
4. Little linguistic variety on the NL side
5. Little variety with respect to location names
6. Unnatural wording of some queries
7. Usage of deprecated OSM tags

In the following subsections, these issues are analyzed in detail and solutions for how to improve on NLMaps v2 are proposed.

### 3.1.1. Train/Test Resemblance

As already noticed by Staniek (2020), the fact that NLMaps v2 was created by using fairly simple templates led to nearly identical NL queries occurring in the training and test set – only the named entities of the area and the named reference point (if any is present in the query) being different. E.g., the training set contains the query *'where book store in Heidelberg'*, while the test set contains the two queries *'where book store in Edinburgh'* and *'where book store in Paris'*.

By removing all queries from the development and test sets that appear identically in the training set when disregarding the named entities, Staniek (ibid.) reduced the size of the test set from 10 594 to 4156 queries. On this smaller test set, his model's accuracy fell from 93.8 % to 83.5 %.

It must be noted that even though the most glaring similarities between the training and the test set can be removed in this way, the underlying reason for the similarity remains: Both sets are generated by the same templates using the same table for mapping NL terms to OSM tags. As a consequence, only 11 of 534 tags (already excluding `name=*` tags) in the test set do not occur in the training set.[1]

[1] And 4 of those are proper names. The 11 tags are: `addr:street=Bergheimer Straße`, `brand=Vauxhall`, `cuisine=german`, `fireplace=yes`, `internet_access:fee=no`, `product=whisky`, `ref=A 4`, `ref=M90`, `school:de=Grundschule`, `shelter_type=weather_shelter`, `sports=tennis`.

While it is theoretically possible to split off a number of templates, terms and OSM tags for generating an independent test set, the templating engine will still remain the same and the templates may also be designed by the same template author. Therefore, a robust evaluation is impossible on a machine-generated test set and must be performed on human-written queries instead.

### 3.1.2. Inconsistencies in NL Term to Tag Mapping

There is a collaborative table (created for the Nominatim geocoder) on the OSM Wiki mapping NL terms to OSM tags,[2] whose structure is shown in a simplified way in the excerpt provided in Table 3.1. For generating an NL-MRL pair of NLMaps v2, Lawrence and Riezler (2018) selected a row of the table, put the NL term into an NL query template and used the OSM tag for building the corresponding MRL query. For terms which are mapped to only one OSM tag in the table, this approach works fine. However, there are terms like *forest* or *bar* which are mapped to two different OSM tags. This leads to the situation that in NLMaps v2 an NL query asking for a *pub* may have `amenity=pub` in the corresponding MRL and another query asking for a *pub* may have `amenity=bar` instead. This is of course unreasonable and also impossible to learn for a model.

| NL Term | OSM Tag |
|---|---|
| airport | `aeroway=aerodrome` |
| bar | `amenity=bar` |
| bar | `amenity=pub` |
| church | `amenity=place_of_worship` |
| church | `building=church` |
| church | `historic=church` |
| forest | `landuse=forest` |
| forest | `natural=wood` |
| pub | `amenity=bar` |
| pub | `amenity=pub` |
| wood | `landuse=forest` |
| wood | `natural=wood` |

Table 3.1.: Simplified excerpt from Nominatim special phrases table.

The solution for this requires some insight into the OSM tags in question. In cases like `landuse=forest` and `natural=wood`, the user issuing the query most

[2] *Nominatim Special Phrases.* `https://wiki.openstreetmap.org/wiki/Nominatim/Special_Phrases/EN`.

likely will not care about the difference,[3] so they should be merged into the union `or(landuse=forest, natural=wood)` in the MRL. In other cases, the user may care about the difference: The bar–pub distinction is fairly transparent[4] and a user asking for pubs should not be referred to bars or vice versa.

### 3.1.3. Inconsistencies in MRL Syntax

When querying for objects *around* some point of interest, it's possible to specify a name for that reference place with the `nwr` operator as well as the area which that reference place is located in with the `area` operator. A typical MRL is shown in Figure 3.2.

```
query(
  around(
    center(
      area(keyval('name','Liverpool')),
      nwr(keyval('name','Mollington Avenue'))
    ),
    search(nwr(keyval('amenity','bank'))),
    maxdist(DIST_INTOWN),
    topx(1)
  ),
  qtype(latlong)
)
```

Figure 3.2.: The MRL for *'closest Bank from Mollington Avenue in Liverpool'* has both the `nwr` and `area` operators in the `center` clause.

When however the reference place is given without specifying the area which it is located in, some MRLs have the reference place in the `nwr` operator while others have it in the `area` operator. Examples are shown in Figure 3.3. This is a meaningless difference and impossible for the model to learn consistently. The easiest way to resolve this is by just replacing the `area` operator with the `nwr` operator when the `center` clause has no `nwr` operator.

### 3.1.4. Little Linguistic Diversity

When looking through NLMaps v2, the reader will notice that most NL queries look very much alike, as demonstrated by the ten random samples in Figure 3.4.

[3] `landuse=forest` is mostly used for areas managed for forestry while `natural=wood` is used for wild forests. However, mappers have different opinions about the issue, as well. In practice, most data processors don't differentiate between the tags. Cf. `https://wiki.openstreetmap.org/wiki/Forest`.

[4] For the details, cf. `https://wiki.openstreetmap.org/wiki/Tag:amenity=bar`.

```
query(
  around(
    center(
      area(keyval('name','Heidelberg'))
    ),
    search(nwr(keyval('place','town'))),
    maxdist(DIST_OUTTOWN)
  ),
  qtype(count)
)
```

(a) *'how many towns around Heidelberg'*

```
query(
  around(
    center(
      nwr(keyval('name','Nantes'))
    ),
    search(nwr(keyval('amenity','waste_basket'))),
    maxdist(DIST_INTOWN)
  ),
  qtype(count)
)
```

(b) *'How many Rubbish Bins near Nantes'*

Figure 3.3.: Inconsistent use of `nwr` and `area` operator for reference place in two MRLs.

This betrays that there has been only a small number of rigid templates in use for generating the dataset, which is a problem because asking whether there is a museum in Nice is of course not only possible with the query *'Is there Museums in Nice'*.[5] The query may also be worded as *'Are there any museums in Nice?'* or *'Does Nice have a museum?'*, to give only two of many possible phrasings.

In order to quantify the NL queries' linguistic diversity, we are going to estimate the entropy rate of NLMaps v2 by viewing it as the result of a source emitting tokens $t$ with a probability $\mathrm{P}(T = t)$. Shannon (1948) defines the entropy of the random variable $T$ as:

$$H(T) = -\sum_{t} \mathrm{P}(T = t) \log_2 \mathrm{P}(T = t) \tag{3.1}$$

If a token's emission probability $\mathrm{P}(T = t)$ depends on the previously emitted

[5] The impact of grammatical errors in generated queries (like the wrong use of *'Is there'* with the plural *'Museums'*) on the accuracy on real world queries will not be investigated. We assume that occasional errors are alright or even beneficial, since slight grammatical (or orthographical) errors will also occur in the real world.

```
where theaters in Edinburgh
How many Doctor in Manchester
Is there Farm Shop in Lille
how many kindergarten in Edinburgh
Garden Centres near École maternelle La Bruyère in Lille
Is there Museums in Nice
Is there close by Public Building from Bramley Street in Bradford
Is there close by Fish Shop from Wohldorfer Schleuse in Hamburg in
    ↪ walking distance
Where Ferry Terminals near sapin noel in Nantes
How many Book Shop in Nice
```

Figure 3.4.: 10 random NLMaps v2 queries.

block $b$ of $n-1$ tokens as the manifestation of random variable $B$, the conditional entropy is defined by

$$H(T|B) = \sum_b \mathrm{P}(B=b) H(T|B=b) \tag{3.2}$$

$$= -\sum_b \mathrm{P}(B=b) \sum_t \mathrm{P}(T=t|B=b) \log_2 \mathrm{P}(T=t|B=b) \tag{3.3}$$

$$= -\sum_b \sum_t \mathrm{P}(B=b)\, \mathrm{P}(T=t|B=b) \log_2 \mathrm{P}(T=t|B=b) \tag{3.4}$$

$$= -\sum_b \sum_t \mathrm{P}(B=b, T=t) \log_2 \mathrm{P}(T=t|B=b) \tag{3.5}$$

where $\mathrm{P}(B=b, T=t) = \mathrm{P}(T=t|B=b)\,\mathrm{P}(B=b)$ is the probability of observing the $n$-gram $(b,t)$. The probabilities can be estimated by observing the trigrams ($n=3$) in NLMaps v2 and we arrive at conditional entropies of 2.11 and 1.37 bits per token for the unmasked and masked variants, respectively.

By using more templates and templates that are not as rigid as the ones used for NLMaps v2, a more diverse set of NL queries can be generated. We expect that the conditional entropy of such a dataset will be increased.

### 3.1.5. Little variety in location names

Suspiciously, the NL sample of NLMaps v2 in Figure 3.4 contains the areas *Edinburgh*, *Lille*, *Nice* in two queries each. In fact, a closer look at the statistics of areas in the dataset – built by extracting the values of `name=*` tags in the `area` operator in all NLMaps v2 MRLs – reveals a large imbalance, as demonstrated by the graph in Figure 3.5. The cities *Heidelberg*, *Edinburgh* and *Paris* appear over 3000 times each, 28 cities appear around 500 times each, 50 other areas appear between once and 65 times each – totalling only 81 different areas.

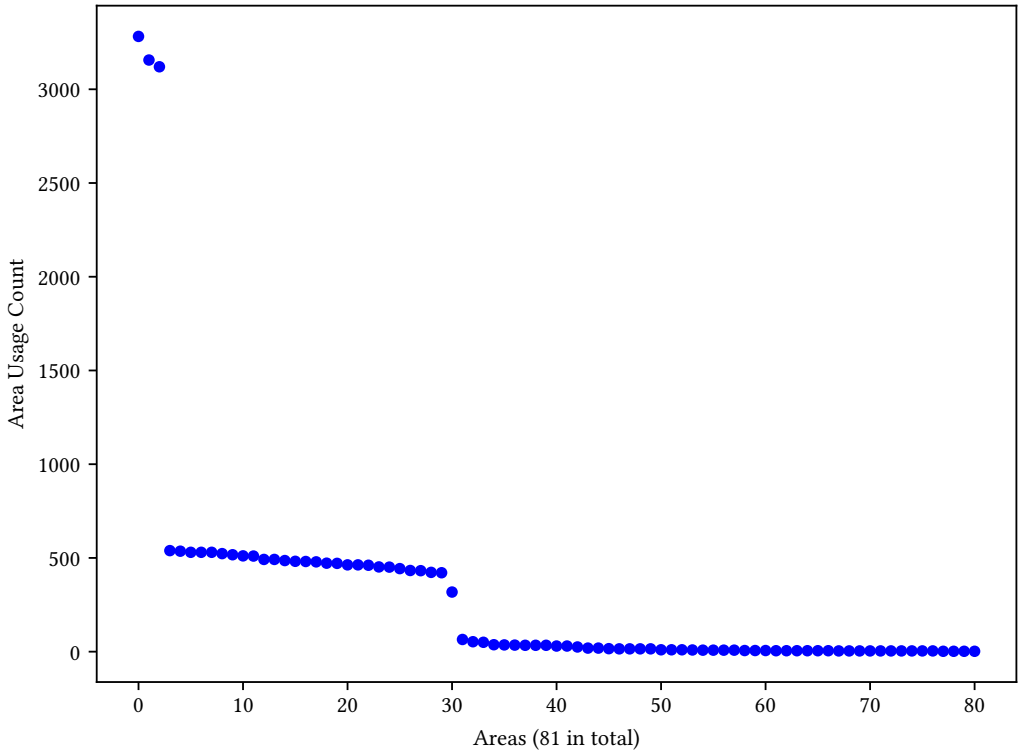



Figure 3.5.: Area names in NLMaps v2.

Some imbalance is also present in the values of `name=*` tags in the `nwr` operators, which are mostly names of points of interests. However, the imbalance is much less pronounced in this case and with 5969 different names there is a large variety of different names. This is illustrated in Figure 3.6.

This lack of variety is not a problem when training on the masked data and using an external NER system because the model will only ever see the placeholders for the location names. However, models directly trained on NLMaps v2 will learn a strong bias with respect to location names, as evidenced by the model hallucinating the area *Paris* out of thin air as seen in Figure 3.1b. An improved version of the dataset should provide a large variety of names, which should also stem from a variety of different countries and languages.

### 3.1.6. Unnatural Wording of Queries

It is some queries' purpose to extract the value associated with a certain key from the result set, which is reflected in the MRL by operations like `findkey('name')` or `findkey('opening_hours)` inside the `qtype` clause. This intention can be coded in the NL query through wordings like *'What are the names ...?'*, *'Name all the ...!'*, *'Tell me the opening hours ...!'* or *'When is ... open?'*.

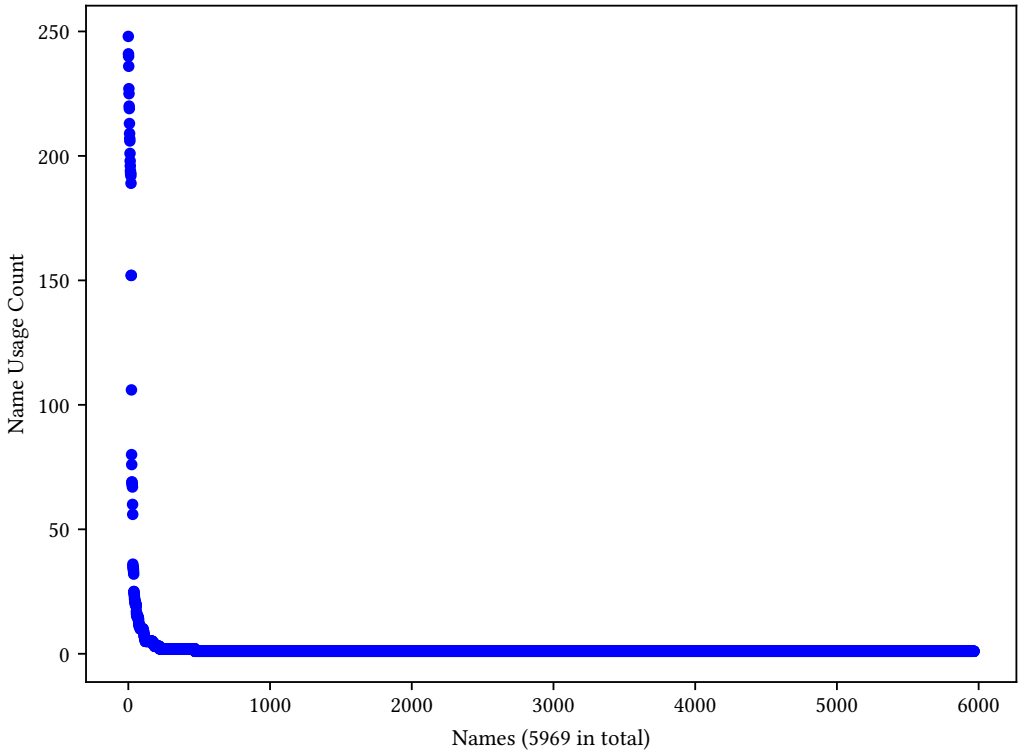



Figure 3.6.: Names in `nwr` operator in NLMaps v2.

This type of NL query is of course present in NLMaps v2, but there are also queries which are simply prefixed by an OSM key, which is then understood as an indication to extract that key via the the `findkey` operator. Five such queries are shown in Figure 3.7. Some of them (such as *'name Paris buy ice cream'*) can pass as crude wordings of legitimate queries.

```
wheelchair Jewelry Shops near München in walking distance
amenity closest Gas Station from Düsseldorf
bicycle Cycle Paths in Sheffield
source Sports Centres near Blakenhale Road in Birmingham
name Paris buy ice cream
```

Figure 3.7.: NL queries from NLMaps v2 where the OSM key that is to be extracted is just added as a prefix.

Others are misleading or at least ambiguous. E.g., *'wheelchair Jewelry Shops near München in walking distance'* is understood as being equivalent to *'Tell me if the Jewelry Shops near München are wheelchair-accessible.'*, whereas it could just as well mean *'Which Jewelry Shops near München are wheelchair-accessible?'*, which might even be the more likely query.

Some are actually nonsensical: Gas stations are selected via `amenity=fuel` in

the first place, so the `amenity` value will of course always be `fuel`; similarly for the bicycle path example. And some keys are rarely worth asking for because they are in general not interesting, such as the `source` key, which is used by OSM mappers to give the information source used for mapping an object (e.g. aerial photography or survey in person).

Paired with the lack of linguistic variety, this method of unnaturally prefixing an otherwise complete NL query with an OSM key is actually harmful when queries are encountered that start with unseen phrases. This is evidenced by the query *'Show me the cafes near Martinskirche in Westheim'* in Figure 3.1a, where the model attempts to extract the value of a nonsensical `show:me` key.

With the exception of somewhat reasonable cases like prefixing *'name'*, all of these queries should be deleted from the dataset in order to improve it.

### 3.1.7. Improper Usage of OSM Tags

So far we have only discussed intrinsic qualities of NLMaps v2 by examining NL and MRL queries and their consistency without paying heed to the usage of the covered tags in the OSM database. This is of course important because the tags only derive their meaning from their usage in the dataset. While most tags are used correctly in NLMaps v2 (e.g. `shop=clothes` used in MRLs for queries asking for *'clothing stores'*), there are also some tags that are not in current use in OSM, have never been in use or their use differs from what they are understood to mean in NLMaps v2. Most of these errors are inherited from the table discussed in Section 3.1.2. Some examples are given:

- `amenity=park`[6] is sometimes used in NLMaps v2, even though `leisure=park` is the proper tag for parks. As of April 2021, `amenity=park` is used only 30 times in OSM.
- `place=house` has probably never had any usage in OSM, even though it is used in NLMaps v2.
- Churches[7] are places of Christian worship and can be identified by the tag combination `amenity=place_of_worship + religion=christian`. However, NLMaps v2 uses `building=church` (and also `historic=church`), which should be used for buildings *built as* churches. They may be used for another

[6] `https://wiki.openstreetmap.org/wiki/Tag:amenity=park`.
[7] `https://wiki.openstreetmap.org/wiki/Church`.

purpose now[8], while some non-church buildings may be used for holding church services and are thus churches.[9]

Choosing the correct tag or tag combination is virtually irrelevant for training the machine learning model, but it will obviously be essential when the MRLs are actually used for an OSM lookup. Finer points of this multifaceted issue will be discussed at a later point in this thesis.

## 3.2. Improving on NLMaps v2

The most obvious way to produce a dataset that is closer to real world queries is to source it from actual users via an annotation project. This is what will be described later in this thesis. But in order to collect data efficiently, a model is helpful that answers the simple questions correctly already, so that annotators will spend less time constructing trivial MRLs. And even for more difficult queries, it's easier to adjust an MRL that is only slightly incorrect than one with several errors.

Therefore, it is reasonable to make an effort of improving on the existing NLMaps v2 dataset by fixing some of its shortcomings and by extending it with new queries generated by an improved templating approach. These steps are described in the following two sections.

### 3.2.1. Fixing NLMaps v2 Shortcomings

The fixing of shortcomings in the existing dataset concentrates on making it more consistent. For reproducibility, all of the fixes are made in a script,[10], which does the following:

- OSM tags in the MRLs are replaced by non-deprecated counterparts or by the union of all applicable tags, in some cases depending on the content of the NL query. Cf. Sections 3.1.2 and 3.1.7. Examples:
  - `amenity=park` ⟶ `leisure=park`
  - `landuse=forest` ⟶ `or(landuse=forest, natural=wood)`
  - NL asks for bars and MRL contains `amenity=pub` ⟶ `amenity=bar`

[8] The Hagia Sophia mosque in Istanbul may be the most famous example of this.
[9] The analogous situation of a non-religious building being used as a mosque is very common in Germany, for example.
[10] `https://gitlab.cl.uni-heidelberg.de/will/nlmaps-tools/-/blob/handed_in/nlmaps_tools/fix_nlmaps_v2.py`.

| Split | NLMaps v2 | Modified | Deleted | NLMaps v2.1 |
|---|---|---|---|---|
| Train | 16 172 | 1236 | 1059 | 15 113 |
| Dev | 1843 | 136 | 109 | 1734 |
| Test | 10 594 | 796 | 691 | 9903 |
| Total | 28 609 | 2168 | 1859 | 26 750 |

Table 3.2.: Numbers of deletions and modifications going from NLMaps v2 to NLMaps v2.1.

- The operator `area` in a `center` clause without an `nwr` operator is replaced by the `nwr` operator. Cf. Section 3.1.3.
- NL-MRL pairs where an OSM tag was used as the prefix of the NL query to indicate a matching `findkey` operator are removed with the exception of the keys `name`, `opening_hours` and `website`. Cf. Section 3.1.6.

By applying these changes to NLMaps v2, 2168 MRLs are modified and a further 1859 instances are deleted resulting in a modified dataset containing 26 750 NL-MRL pairs, which will be called NLMaps v2.1. More detailed numbers are given in Table 3.2. Note that the NL side of queries is never modified in the process.

### 3.2.2. Extension of NLMaps v2

In order to address also the other shortcomings, a new dataset is generated in a more sophisticated templating approach. The new approach differs by the one used for NLMaps v2 in the following ways:

- More templates are used.
- There is significant variation within each template.
- More area names are used.
- Area names are more evenly distributed.
- More OSM tags are used. For this, the information from the table used for NLMaps v2 is manually extended.
- Errors in the tag usage (as discussed in Sections 3.1.2 and 3.1.7) are of course avoided in the first place.

```
when
{{ choose(['can I', 'can we', 'to'], [0.3, 0.3, 0.4]) }}
{{ choose(['visit', 'go to'], [0.6, 0.4]) }}
{% if plural %}
  {{ choose(['the', 'all', 'all the', ''], [0.2, 0.2, 0.2, 0.4]) }}
{% else %}
  {{ choose(['a', 'some', 'any', '']) }}
{% endif %}
{{ thing_plural if plural else thing_singular }}
{% include 'meta/in_location.jinja2' %}
{{ optional('?') }}
```

Figure 3.8.: Simple template for one version of queries for opening hours.

The templates are designed to be probabilistic. I.e., instead of being rigid, they are essentially decision trees with various decisions being made to arrive at the final wording of an NL query. Figure 3.8 shows one of several templates used for generating queries that ask for opening hours. By choosing phrases according to the given probability distributions, it can produce queries like *'when to visit theatres in Bratislava?'* or *'when can I go to a cinema in Hannover'*. The resulting dataet is called NLMaps v3a.

The location names are collected via extracting all areas and named places from different regions in countries that use variations of the Latin alphabet. This is done to ensure that location names from various languages are included in the dataset.[11] Figure 3.9 shows a large variety in well-distributed area names. This is in contrast with the situation in NLMaps v2 shown in Figure 3.5.

In order to train models that are robust against typing errors and other small spelling deviations, some noise is added to the NL queries in NLMaps v3a by switching a character for another with a chance of 1 %. Names of locations are never touched, however. The resulting dataset with noise is called NLMaps v3b and a random sample of NL queries is shown in Figure 3.10.

Finally, we concatenate NLMaps v2 and NLMaps v3b and call the result NLMaps v3. Since we generate exactly as many instances for NLMaps v3b as are present in the corresponding split of NLMaps v2.1, NLMaps v3 is exactly twice as large as NLMaps v2.1.

The linguistic diversity of the resulting datasets is quantified by the entropy rate estimated by the conditional entropy on trigrams, as described in Section 3.1.4.

[11] When generating a query for things around some named place inside an area, both the place and the area are randomly selected independently from each other. This leads to queries like *'restaurants near Eiffel Tower in Rome'*, which do not make sense in practice because there is no Eiffel Tower in Rome, but that doesn't matter since the model is just supposed to learn to copy the names to the appropriate place in the MRL.

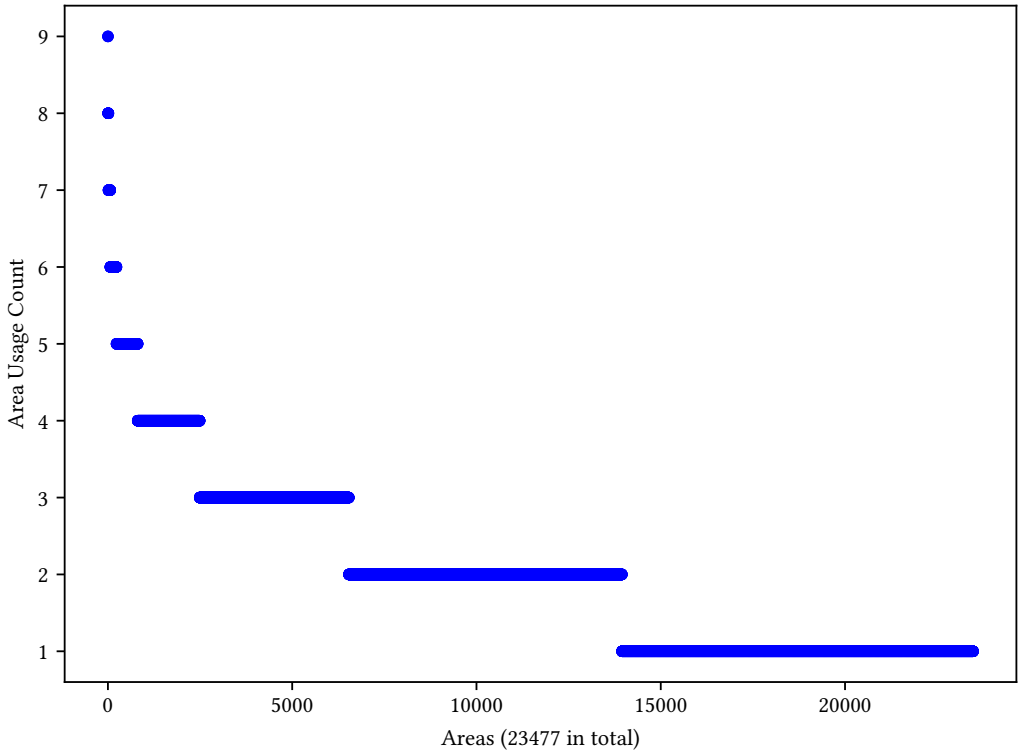



Figure 3.9.: Area values in NLMaps v3a.

Table 3.3 offers an overview over the datasets.

| Measure | v2 | v2.1 | v3a | v3b | v3 |
|---|---|---|---|---|---|
| Instances | 28 609 | 26 750 | 53 500 | 53 500 | 53 500 |
| Conditional Entropy | 2.11 | 2.08 | 2.85 | 2.93 | 2.73 |
| Avg. Tokens per NL | 6.98 | 7.02 | 10.72 | 10.69 | 8.85 |

Table 3.3.: Comparison of dataset statistics. Entropy rates are estimated by the conditional entropy trigrams.

```
what bathrooms in Záluží are around Zur Stöpe?
what preschools in Sadek are in walking distance hrom Gasthaus
    ↪ Tannengarten?
Do some veterinary surgerils exist east of studna in Montgeron (
    ↪ canton de Draveil)
Gire me any department store around ROBOT in Neue Vahr Südost
which setvice rosd in Bardzice is south of Le Ciré Jaune?
show me the opening times of all the monorail in the area of The
    ↪ KPH in Świecie
Which viewpoint is there in Miño de San Esteban?
In Grabowiec, what are the opening hours of all the boatyards less
    ↪  than 80 kilometres away from MakroMueble
Indicate the coordinaSes of all byways Tn přírodní památka
    ↪ Branžovy.
Are there parks east of Tigery?
```

Figure 3.10.: 10 random NLMaps v3b queries.

# 4. Web Interface

The model is exposed via a new web interface, which enables a user to enter an NL query. The NL query is then parsed into the MRL query, which is presented to the user. Additionally, the MRL is interpreted by a newly written MRL interpreter, which retrieves the requested information from OSM via queries to the Overpass API and Nominatim. The result is then presented on an interactive OpenLayers[1] map and also as a textual answer if applicable. (It doesn't make much sense to give coordinates as a textual answer, for example.)

If the user notices that the presented MRL is incorrect, they have the option to correct it. To facilitate correcting the MRL, the user is supported by automatic tag suggestions powered by TagFinder[2] (Gwerder 2014) and also fixed custom suggestions for tricky cases, both of which are based on keywords extracted from the NL query. Additionally, the MRL is corrected via a web form that abstracts away the details of the MRL syntax so that the user doesn't have to understand that and can also not make any simple mistakes like not closing parentheses.

Finally, the user can tell the system that an MRL query is correct when they are satisfied with it. The new NL-MRL pair is saved and is also directly used for improving the system by online training.

The screenshots in Figure 4.1 show the typical flow from the user's perspective when the query is successfully parsed and those in Figure 4.2 show the flow for a flawed MRL and its correction.

## 4.1. Architecture

Instead of being one large system, NLMaps Web is split into two parts: First, the web interface the user interacts with, which also includes user management, giving tag help, displaying answers, logging queries and a tutorial. Second, the parsing server that parses NL queries into MRL queries, trains and updates the model based on feedback received through the web interface and also stores that feedback. They

[1] Schaub et al. (*OpenLayers*). `https://openlayers.org/`.
[2] Gwerder (*OSM TagFinder*). `https://tagfinder.herokuapp.com/`.

are separated so that the machine that runs the web server is not required to also handle running or even training a neural network, which is a resource-intensive task that is usually parallelized on a GPU. Due to the separation, it is possible to use a small machine for the web interface that interacts with the parsing server running on a GPU cluster, which may only be accessible by SSH and thus be unreachable through a well-known HTTP port.

Both systems are implemented as HTTP Servers with the Python web framework Flask[3] and employ SQLite[4] as their database. The parsing server exposes a JSON-based HTTP API, which is used by the web interface. For handling the machine translation training and predicting, it wraps the PyTorch[5]-based sequence-to-sequence learning framework Joey NMT[6] (Kreutzer, Bastings, et al. 2019).

Figure 4.3 shows the basic querying flow through the architecture: The user enters their NL query into the web interface, which calls on the parsing server to parse it into an MRL. The MRL is used to retrieve the answer via the MRL interpretation package (cf. Section 4.2) and the web interface sends the MRL query along with the retrieved answer in its response to the user.

As shown in Figure 4.4, the web interface also extracts the keywords from the NL query (cf. Section 4.3) and suggests tags based on them. With this information, the user can correct the MRL if it is wrong and can send their feedback in the form of a correct NL-MRL query pair to the web interface. The feedback is then sent to the parsing server in order to initiate the training procedure and to update the model.

## 4.2. MRL Interpretation

In the course of their foundational work, Haas and Riezler (2016a) forked[7] the Overpass API and included functionality for reading an MRL query and executing the Overpass QL queries necessary to answer it. Unfortunately, there are three problems with this approach.

- Their fork has been unmaintained for years and thus does not profit from further development and bugfixes in the upstream Overpass API project. Merging the upstream bugfixes and other changes and maintaining the fork would mean a lot of ongoing work.

[3]Pallets Projects (*Flask*). `https://palletsprojects.com/p/flask/`.
[4]Hipp et al. (*SQLite*). `https://sqlite.org/`.
[5]Facebook Inc. (*PyTorch*). `https://pytorch.org`.
[6]Kreutzer, Bastings, et al. (*Joey NMT*). `https://github.com/joeynmt/joeynmt`.
[7]Lawrence (*Overpass NLmaps*). `https://github.com/carhaas/overpass-nlmaps`.

- Running an instance of the Overpass API entails having a complete copy of OSM data stored in the Overpass database and keeping it up to date, which also is a lot of work.
- The Overpass API is meant for precise queries to the OSM database and has no functionality for fuzzy matching of place names[8] or for ranking results by importance. These are typical cases where this becomes a problem:
  - The user asks *'Show Verpackungsmuseum in Heidelberg'* and the parser correctly analyzes that the user wants a place called *Verpackungsmuseum*. However, the Overpass API will not find such a place because that museum is in fact called *Deutsches Verpackungsmuseum*.
  - The user asks for objects in *Paris*, but there are several cities with that name in the world and the Overpass API has no importance ranking with which it could determine that the capital of France is most likely the city the user has in mind.

Instead of using the forked version for answering MRLs, we develop a Python module for that task whose manner of operation is explained here. In a first step, the area requested in the query is looked up in Nominatim, which supports fuzzier search than the Overpass API and also ranks the results by an importance score. Second, the named reference location is looked up if there is one in the query (e.g. *Eiffel Tower* in *'bars near Eiffel Tower in Paris'*). This is also done via Nominatim and the results are restricted to the area that was selected in the previous step. Finally, an Overpass QL query is generated where the previously retrieved area and reference location are selected by their OSM ID instead of their name. This query is then sent to any of the publicly available Overpass API instances for retrieving the result, which means that there is no need of running a separate instance of the API with all the maintenance work.

## 4.3. NL Query Keyword Extraction

In order to suggest tags for the user to use when correcting a faulty MRL query, the most relevant keywords are extracted from their NL query so that the keywords can be looked up in the TagFinder. Assuming that a relevant keyword is a term occurring in that query that does not occur in a lot of other queries, we turn to tfidf for ranking the terms.

[8] Regular expressions are supported, but they do not suffice for this task.

For a term $t$ in an NL query $d$, its tfidf score is calculated as

$$\text{tfidf}(t, d, D) = \text{tf}(t, d) \times \text{idf}(t, D) \tag{4.1}$$

where the collection of all NL queries in NLMaps v3 is used as the reference document set $D$. The term frequency $\text{tf}(t, d)$ is the raw count of term $t$ in the NL query $d$ and the inverse document frequency is calculated as

$$\text{idf}(t, D) = \ln \frac{N + 1}{\text{df}(t, D) + 1} + 1 \tag{4.2}$$

where $N = |D|$ is the number of NL queries in $D$ and $df(t, D) = |\{d \in D : t \in d\}|$ is the number of NL queries containing the term $t$. 1 is added in the numerator and in the denominator for smoothing over unseen terms and a further 1 is added in order not to completely disregard terms that occur in every NL query.

Any terms with a score $\text{tfidf}(t, d, D) > 0.3$ (the cutoff was determined manually) are looked up in TagFinder except if they occur in a separate list of stop words or if they are part of a location name, which is determined by using the MRL query predicted by the parser.

## 4.4. Online Learning

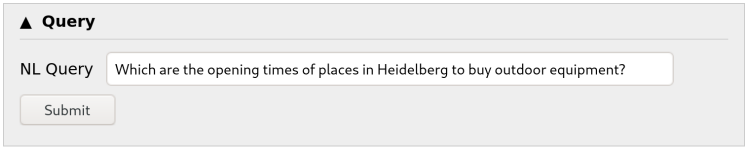
In interactive machine translation or other human-in-the-loop learning processes, it is often assumed that there is only one user and the interaction process is strictly sequential: The system makes a prediction and then waits for the human to make a post-edit or to give some other kind of feedback. As soon as it receives the feedback, it immediately updates the model parameters and is then ready to make another prediction. However, this process breaks down in a multi-user setting since a user can send feedback when the model is still in the process of updating its parameters on some other user's feedback. One could resolve this by providing one model per user that will only update based on their corresponding user's feedback, but this would mean that a user will not benefit from other users' feedback, which is undesirable. On top of that, maintaining a separate model for each user quickly becomes costly in practice.

Therefore, we use an asynchronous learning setup – formalized in Algorithm 1 –, where the MT server permanently runs a dedicated process for updating the model. It runs a loop that detects if any fresh feedback is present in the list $\mathcal{F}$. If this is the case, it goes through all instances in $\mathcal{F}$ creating batches that can also include instances from a fixed training set (e.g. the training set the model was

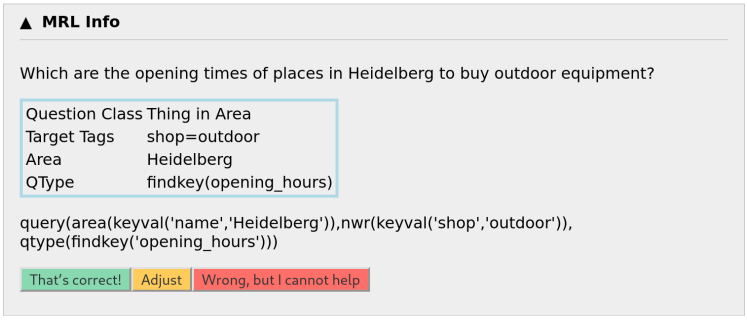
pre-trained on) and instances that are *memorized* from feedback that has already been processed in the past. On these batches, the process then makes gradient descent updates of the model parameters. While the process is running the update procedure, users can still use the system and give feedback, which is added to $\mathcal{F}$ and will be processed by the update process in the next iteration of the loop.

**Algorithm 1** Asynchronous Online Learning

**procedure** AsynchronousOnlineLearning($\theta$, $\mathcal{F}$, $\mathcal{M}$, $\mathcal{T}$, $n_F$, $n_M$, $n_T$, $I$)
  $\theta$: Model parameters
  $\mathcal{F}$: List of fresh MRL-NL feedback
  $\mathcal{M}$: Set of memorized older MRL-NL feedback
  $\mathcal{T}$: Set of MRL-NL pairs from pre-training
  $n_F$: Instances from F per batch
  $n_M$: Instances from M per batch
  $n_T$: Instances from T per batch
  $I$: Iterations per fresh feedback
  **loop**
    **if** $|\mathcal{F}| > 0$ **then**
      $\mathcal{F}' \leftarrow$ Copy $\mathcal{F}$
      $\mathcal{F} \leftarrow$ Empty list ▷ Users can add to $\mathcal{F}$ without affecting $\mathcal{F}'$
      **for** $i \leftarrow 1 \dots I$ **do**
        **for** offset $\leftarrow 0 \dots \left\lfloor \frac{|\mathcal{F}'|}{n_F} \right\rfloor$ **do**
                ▷ Go through $\mathcal{F}'$ in $n_F$-sized batches
          $b_F \leftarrow \mathcal{F}'[\text{offset}] \dots \mathcal{F}'[\text{offset} + n_F - 1]$
          $b_M \leftarrow$ Sample $n_M$ instances from $\mathcal{M}$
          $b_T \leftarrow$ Sample $n_T$ instances from $\mathcal{T}$
          $b \leftarrow$ Concatenate batches $b_F$, $b_M$, $b_T$
          Update parameters $\theta$ on batch $b$
        **end for**
      **end for**
      Add all instances in $\mathcal{F}'$ to $\mathcal{M}$
    **end if**
  **end loop**
**end procedure**

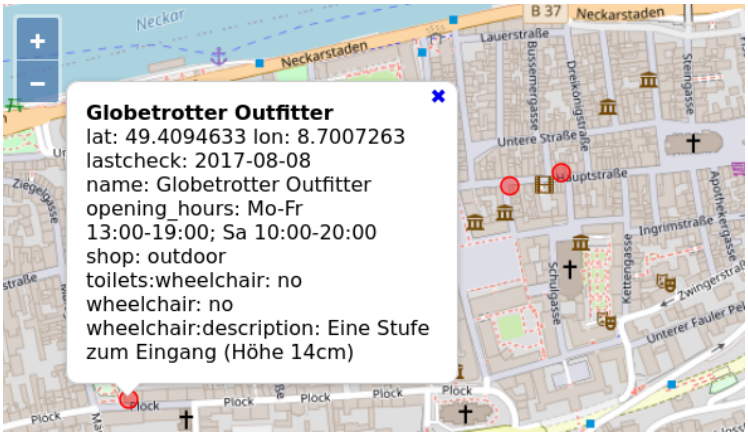



(a) User enters NL query.



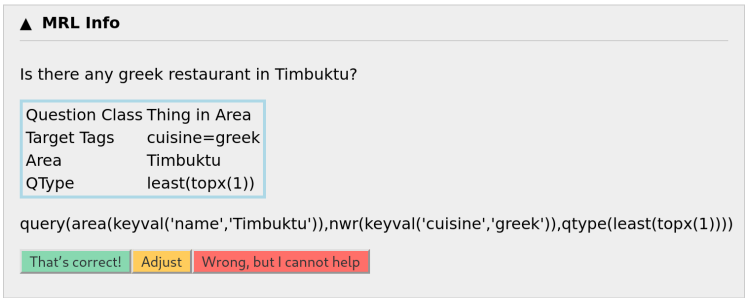

(b) Info about MRL query the parser produced.


(c) Answer of the query.



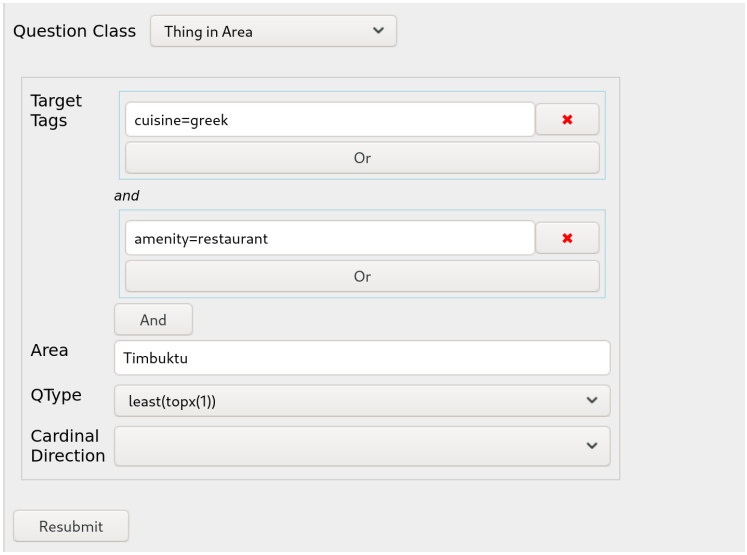

(d) Interactive map of the results.

Figure 4.1.: Successful query process with the NL query *'Which are the opening times of places in Heidelberg to buy outdoor equipment?'*.




(a) MRL Info missing the `amenity=restaurant` tag.


(b) Help for the user showing tags with similar spelling, custom suggestions for the keyword *greek* and TagFinder suggestions for *restaurant*.




(c) Form where the user added `amenity=restaurant`.

Figure 4.2.: MRL correction process after asking *'Is there any greek restaurant in Timbuktu?'*.

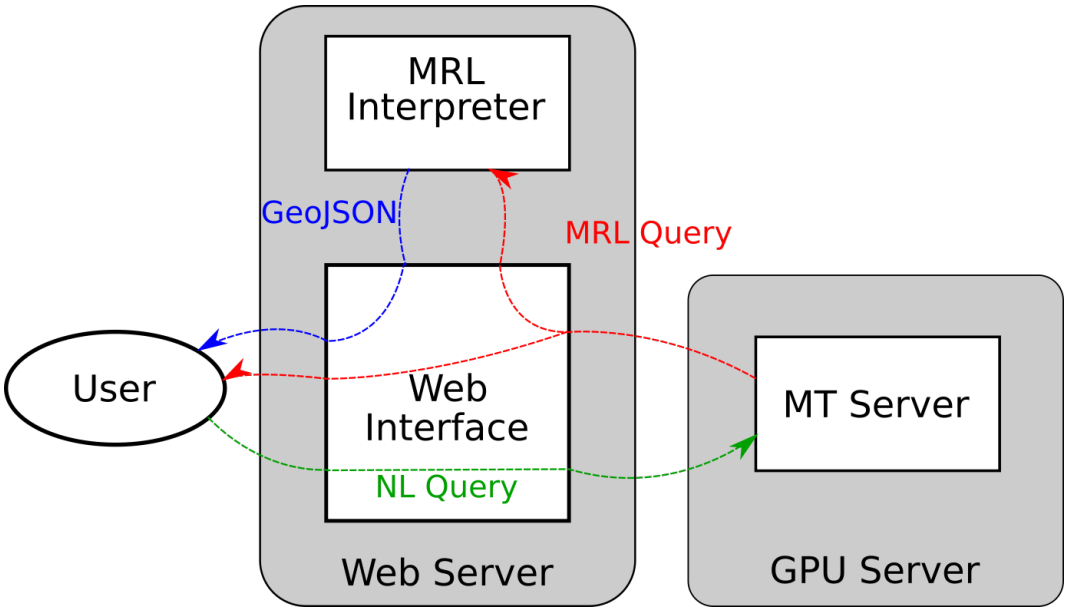



Figure 4.3.: System Architecture for Querying.

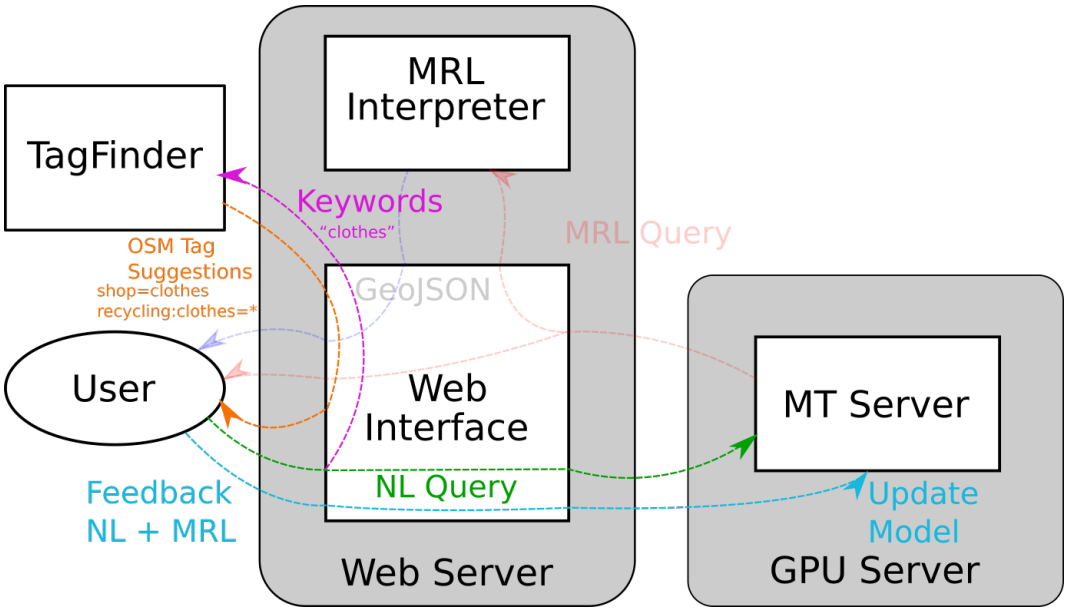



Figure 4.4.: System Architecture for Feedback and Learning.

# 5. Experiments

Our goal is simulating different strategies of online learning for the new web interface. Before we can do that, we first pre-train parsers in Section 5.1 on the existing datasets NLMaps v2 and the variations we introduced in NLMaps v3 and evaluate whether the dataset extensions yield any improvement in parser quality. In Section 5.2, we then hire annotators to use our web interface to ask NL queries and correct the MRL parse if it is incorrect, thus collecting a new dataset consisting of real user queries. Finally, in Section 5.3 we use the newly acquired dataset for evaluating various online learning setups.

For all our experiments, we use the same model architecture: A character-based one-layer bidirectional GRU encoder-decoder (Cho et al. 2014) model with attention(Bahdanau et al. 2015). The dimension of both source and target embeddings is 620, the encoder layer size is 500, the decoder layer size is 1000 and we don't use dropout. This model configuration is adopted from Staniek (2020).

## 5.1. Training on NLMaps v2 and NLMaps v3

While Staniek (ibid.) trained his model on the NLMaps v2 dataset for 100 epochs (of 16 172 instances each), we train our models in this section for a shorter time, which is still enough for sufficient convergence: The model on NLMaps v2.1 is trained for 60 epochs (of 15 113 instances each), while the models on variations of NLMaps v3 are trained for 30 epochs (of 30 226 instances each). All models are trained with the Adam optimizer ($\beta_1 = 0.9$ and $\beta_2 = 0.999$) and a learning rate of 0.0002.

Table 5.1 shows the results of testing the models on the different variations of NLMaps datasets. Recall: v2.1 is the result of fixing issues in the MRL queries of v2, but has identical NL queries; v3a is purely generated with probabilistic templates and v3b is its counterpart with added noise; $\text{v3}_{\text{no-noise}}$ is v2.1 + v3a and v3 is v2.1 + v3b. The datasets $\text{v4}_{\text{raw}}$ and v4 are introduced in Section 5.2 and are the user-supplied MRL-NL pairs with some corrections applied to them in v4. Even though the last two datasets were not yet available when the models were pre-trained, we still evaluate the pre-trained models on them in this section because we consider

the results on these user-supplied queries the most relevant.

| Train \ Test | v2 | v2.1 | v3a | v3b | $v3_{no\text{-}noise}$ | v3 | $v4_{raw}$ | v4 |
|---|---|---|---|---|---|---|---|---|
| Staniek (2020) | **0.898** | 0.844 | 0.039 | 0.033 | 0.441 | 0.439 | 0.050 | 0.052 |
| v2.1 | 0.783 | 0.913 | 0.034 | 0.029 | 0.474 | 0.471 | 0.070 | 0.069 |
| v3a | 0.224 | 0.251 | **0.987** | 0.789 | 0.618 | 0.519 | 0.217 | 0.223 |
| v3b | 0.372 | 0.424 | 0.976 | **0.884** | 0.700 | 0.656 | 0.226 | 0.233 |
| $v3_{no\text{-}noise}$ | 0.790 | **0.919** | 0.978 | 0.792 | **0.948** | 0.857 | **0.307** | **0.311** |
| v3 | 0.787 | 0.913 | 0.950 | 0.834 | 0.931 | **0.874** | 0.281 | 0.289 |

Table 5.1.: Performance of pre-trained parsers.

Unsurprisingly, the model by Staniek (2020), which was trained on NLMaps v2, performs best on the corresponding test set with an accuracy of 89.8 %. The model trained on v2.1, which contains the fixes of tag usage and inconsistencies described in Chapter 3, achieves a higher accuracy on its corresponding test set with 91.3 %. This is to be expected since the resolution of inconsistencies in MRL structure and tag usage makes a new part of the dataset accessible for confident predictions in the first place. These two models' performance drops dramatically on new datasets. An error analysis (see Figure 5.1) shows that not only does the model trained on v2.1 make an error in the `target_nwr` operator in over 70 % of queries from the NLMaps v4 test set, the recognized `area` is false in more than half the queries, as well.

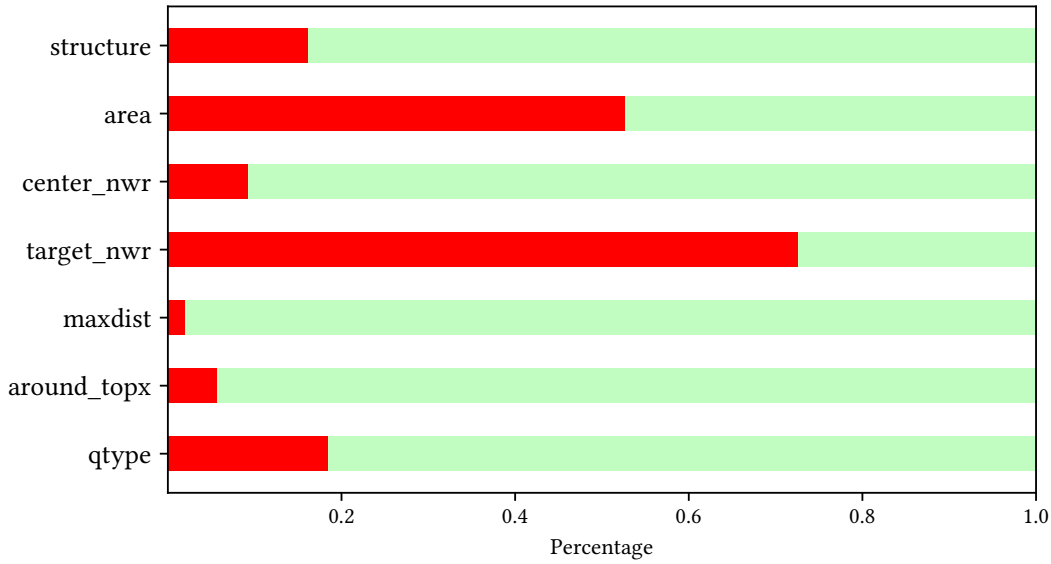



Figure 5.1.: Percentage of queries with an error in a specific operator. The model was pre-trained on NLMaps v2.1 and tested on NLMaps v4.

The models trained on the purely synthetic training sets of v3a and v3b achieve an extremely high accuracy of around 98 % on the test set of v3a. The model trained on the non-noisy v3a turns out to be less robust on the noisy v3b test set with a performance drop of almost 20 % whereas the accuracy of the model trained on v3b only drops by around 9 %. More interestingly, the v3b model significantly outperforms the v3a model on the v2.1 test set, as well, which suggests that the added noise serves as a means of regularization and avoids overfitting on the synthetic queries. When confronted with the real queries in NLMaps v4, both models' accuracy again drops starkly to around 23 % – with the noisy model still performing slightly better than its non-noisy counterpart. As shown in Figure 5.2, the improvement with respect to the model trained on v2.1 is mostly due to the greater variety of location names (cf. Section 3.1.5), which significantly reduces the number of errors in the `area` and `center_nwr` operators. The main source of errors remains choosing an incorrect set of OSM tags in the `target_nwr` operator.

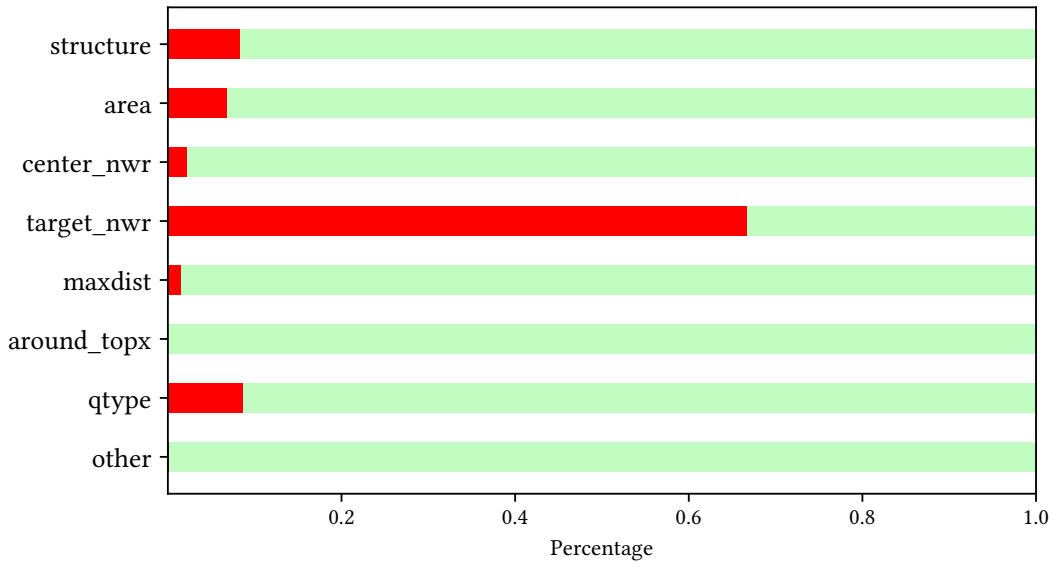



Figure 5.2.: Percentage of queries with an error in a specific operator. The model was pre-trained on NLMaps v3b and tested on NLMaps v4.

Combining v2.1 and the new synthetic data from v3a or v3b yields further improvement as evidenced by the performance of the models trained on $v3_{\text{no-noise}}$ and v3. They perform well on v2.1 as well as on the variations of v3, but the accuracy on the noisy versions is still noticeably smaller than on their non-noisy counterparts. While the comparison of the two models trained on v3a and v3b showed a clearly superior performance of the model trained on the noisy data in all variations of v2 and v3 except v3a, this is not observed when comparing the two models trained on $v3_{\text{no-noise}}$ and v3. Instead, they have a similar accuracy on all test sets with the

model trained on $v3_{\text{no-noise}}$ being slightly superior on the non-noisy datasets and the model trained on v3 being slightly superior on the noisy ones. Testing on the real user queries from v4 reveals that the model trained on the non-noisy data actually performs better in the end. However, since v4 was not available during the pre-training phase, at that point the two models seemed comparable and our assumption was that the model trained on v3b would prove more robust against typing errors.

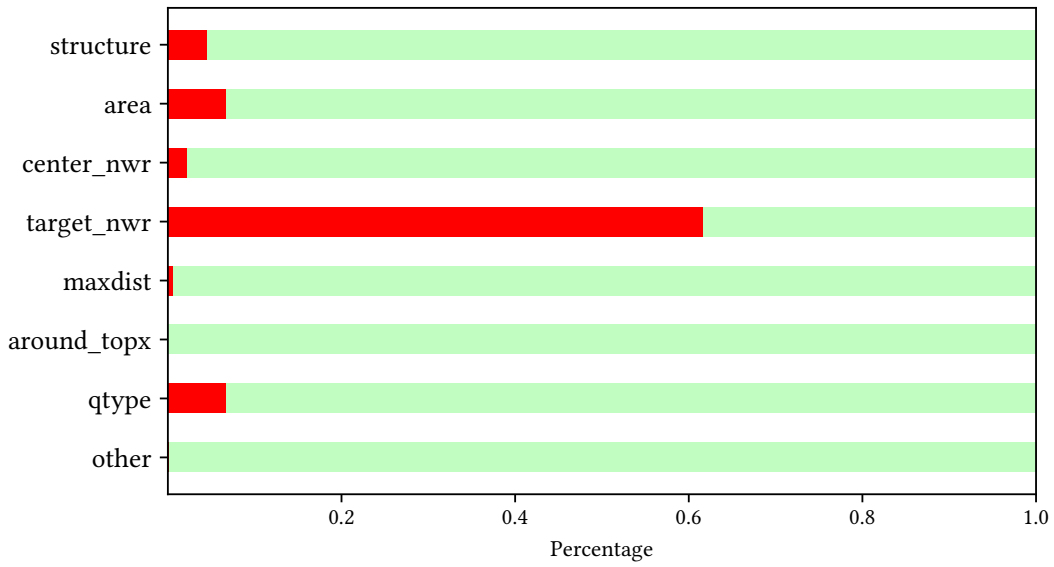



Figure 5.3.: Percentage of queries with an error in a specific operator. The model was pre-trained on NLMaps v3 and tested on NLMaps v4.

Figure 5.4 shows the models' learning curves on the development set of v3 during pre-training. While all models converge fairly quickly, it can be seen that the model trained on v2.1 converges somewhat faster than the other models. This might again be due to the simpler location names, which can be memorized by the model, while the other models have to learn to copy arbitrary location names – including some that are very long or contain rare characters.

## 5.2. Annotation for New Dataset

Training and evaluating useful NLMaps models without having a dataset of actual user queries is problematic. For this reason, an annotation experiment is conducted, during which several people use our web interface presented in Chapter 4 to issue new NL queries. They are tasked to confirm the model's predicted MRL query if they deem it correct or to correct it if they consider it false. Before beginning the task, the annotators complete a tutorial, which explains the basics of

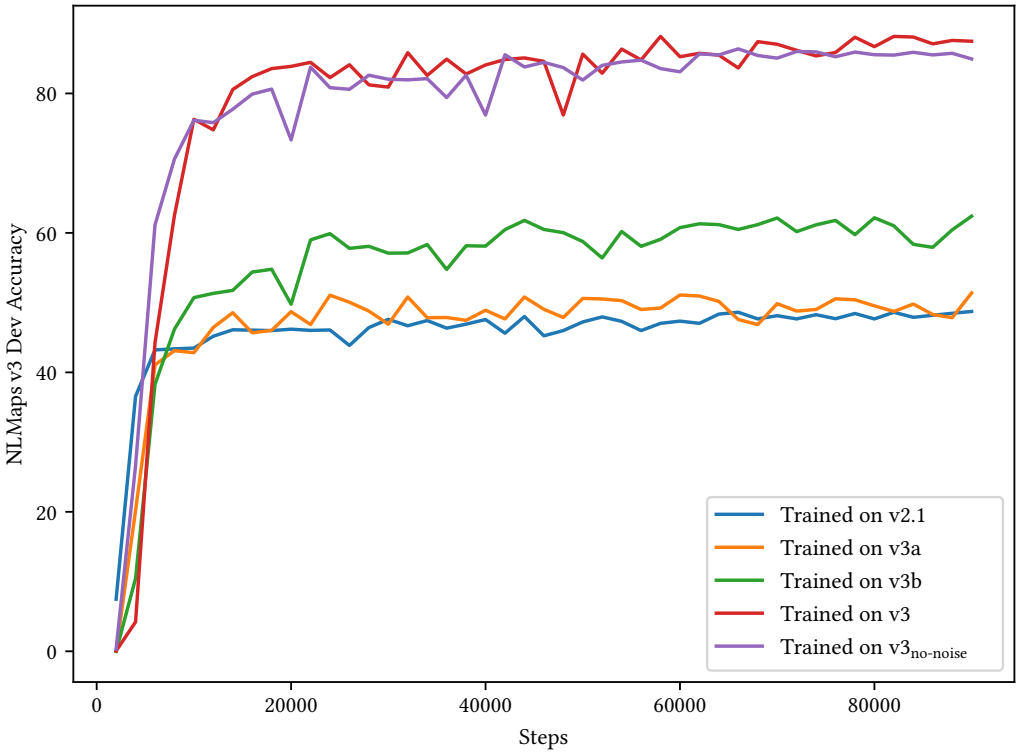



Figure 5.4.: Learning curves on the development set of NLMaps v3.

OSM, the capabilities of the MRL language and the usage of the web interface.

The call for participation in the annotation study was distributed through various OSM-typical channels including the *talk* mailing list[1] and the OSM subreddit[2], which yielded the highest response. Albeit slightly eurocentric, the participants stem from a reasonably diverse set of countries and have diverse native languages. In contrast, all of the acquired annotators report their gender as male, which is reflective of the overall OSM demographic since only between 2 % and 5 % of OSM contributors are estimated to be female (Budhathoki 2010; Stark 2010; Lechner 2011; Schmidt and Klettner 2013; Das et al. 2019). This is problematic as it has the potential to lead to a bias in the dataset. In future annotation projects, this effect should be mitigated by targeted advertising in female mapping communities such as GeoChicas.[3]

The annotation guidelines explain to the annotators that their queries should be linguistically diverse, cover a large part of OSM tags and cover more useful tags in more detail. To ensure that the annotators sufficiently cover the most important tags, we choose a set of tags (e.g. `amenity=cafe` and `wheelchair=yes`) and keys

[1] *OSM Mailing List* talk. `https://lists.openstreetmap.org/listinfo/talk`.
[2] *Subreddit r/openstreetmap*. `https://www.reddit.com/r/openstreetmap`.
[3] *GeoChicas*. `https://wiki.openstreetmap.org/wiki/GeoChicas`.

| #Annotations | Nationality | Native Language | OSM Experience | Gender |
|---|---|---|---|---|
| 442 | Poland | Polish | Extensive | Male |
| 414 | Germany | German | Extensive | Male |
| 405 | Turkey | Turkish | Medium | Male |
| 404 | UK | English | Medium | Male |
| 404 | Hungary | Hungarian | Extensive | Male |
| 400 | India | Hindi | Little | Male |
| 400 | Turkey | Turkish | Little | Male |
| 393 | Germany | German | Little | Male |
| 318 | Germany | German | Extensive | Male |
| 253 | Brazil | Portuguese | Little | Male |
| 253 | Nepal | Nepalese | Extensive | Male |
| 26 | Philippines | English, Filipino, Cebuano | Extensive | Male |
| 40 | *Various untracked people without login* | | | |

Table 5.2.: Information about annotators. Everything but the number of annotations is self-reported by the annotators.

(e.g. `shop` and `leisure`) for which each annotator must issue a minimum number of queries. See the full guidelines in Appendix B for details. The annotators can monitor their annotation progress via an overview page in the web interface, which is shown in Figure 5.5.

During the annotation, the parser model used by the web interface is initially the model from Section 5.1 that is pre-trained on NLMaps v3. Due to technical problems however, no systematic online learning happened throughout the annotation experiment because of various crashes. The result is a dataset where the annotators made their corrections based on predictions made by a model that had learned from some, but not all annotations.

In total, the annotators issued 4152 NL queries. For 92 of those, they could not give an MRL, mostly because the queries were not expressible by an MRL query (e.g. *'cities without hospitals in Poland'*), leaving 4060 NL-MRL pairs. Since Staniek (2020) correctly criticized that NLMaps v2 has queries in its development and test sets that appear identically in the training set if location names are masked (cf. Section 3.1.1), we identify such sets of identical queries in the remaining 4060 NL-MRL pairs and delete all but one in each set, which leaves 3773 NL-MRL pairs.

Furthermore, we observe that users tend to reuse the same areas many times. E.g., a user from Istanbul will ask lots of questions about Istanbul, which leads to the problem of lacking variety in location names that was already identified as a problem in Section 3.1.5. In order to avoid this problem in our new dataset, the web interface automatically replaces a location name (anything with OSM tag `name=*`) if it has occurred three or more times in that user's queries already. The

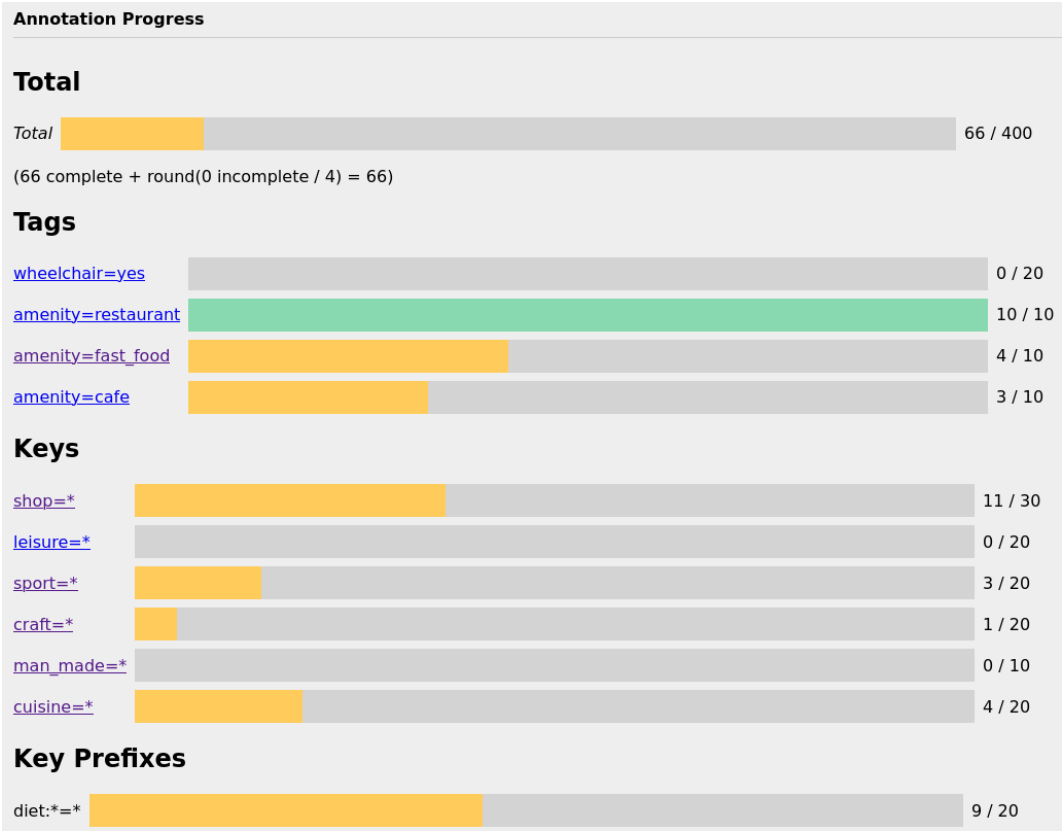



Figure 5.5.: Overview of an annotator's annotation progress.

original query is of course saved separately, as well. The location names used for replacement are selected from the same large list of areas and points of interest that is also used for generating the NLMaps v3 dataset. The resulting dataset of 3773 NL-MRL pairs with replacement of some location names is called NLMaps $v4_{raw}$.

Finally, we notice some errors in the MRLs the annotators created. E.g., for the NL query *'show me restaurants with limited wheelchair access in gmina Piaseczno'*, the annotated MRL query erroneously uses `qtype(findkey( 'wheelchair'))` (which informs about the wheelchair accessibility status of returned objects) instead of actually *selecting* for wheelchair status by using `keyval('wheelchair', 'limited')` in the `nwr` operator. Similar queries by the same annotator where they didn't make that mistake show that they understood the issue and may just have been unconcentrated in this case. Other examples include using `diet=chinese` instead of `cuisine=chinese` or `shop=fuel` instead of `amenity=fuel`, both of which can be attributed to a lack of concentration, which is not surprising in a repetitive annotation task.

Besides these obvious errors, there is a separate and more frequent issue with ambiguous NL queries. E.g., the NL query *'count the fireplaces in the Naturpark Schönbuch'* is annotated to select objects tagged as `keyval('leisure','firepit')`,

which ignores fireplaces that are mapped with the key `fireplace=yes` on picnic sites. Another annotator issues a similar NL query, but creates an MRL which selects objects tagged as `or(keyval('fireplace','yes'), keyval('leisure', 'firepit'))`. While the second query is more inclusive and overall more fitting for the NL query, there is a valid reason to exclude `fireplace=yes`: It can also be used on wilderness huts to indicate that they feature an *indoor* fireplace, which may not be what the user has in mind. This is one of numerous cases where it is not at all obvious what the correct MRL query for an NL query should be.

Since our goal is creating a dataset that is as useful as possible for training an NLMaps model which makes sensible and consistent predictions, we fix the obvious errors and we also adjust MRLs where there are diverging MRL interpretations for similar NL queries in the dataset. This is judged by the author Simon Will based on the MRLs created by all the annotators. As a result of this resolution, 377 MRLs of the 3773 instances are modified to create the dataset called NLMaps v4.

Both $v4_{raw}$ and v4 are split into training, development and test sets containing 2264, 754 and 755 instances respectively. Table 5.3 compares the new datasets with v2.1 and v3b and shows that the user queries are actually shorter and have a smaller entropy rate than the queries generated by our probabilistic templates.

| Measure | v2.1 | v3b | $v4_{raw}$/v4 |
|---|---|---|---|
| Instances | 28 609 | 53 500 | 3773 |
| Conditional Entropy | 2.11 | 2.93 | 2.68 |
| Avg. Tokens per NL | 6.98 | 10.69 | 8.37 |

Table 5.3.: Comparison of dataset statistics. Entropy rates are estimated by the conditional entropy trigrams.

Probably due to the influence of the annotation guidelines, the most common tags in NLMaps v4 are `wheelchair=yes`, `amenity=restaurant` and `amenity=cafe`, which are used in over 100 queries each. What makes the dataset especially challenging, is the long tail of rarely used tags: 310 tags are used only once and 489 are used at most three times. Figure 5.6 reveals two insights about how the tag distribution in NLMaps v4 compares to that of the previous datasets: The synthetic nature of NLMaps v3b and the larger part of NLMaps v2.1 makes it feasible to generate an arbitrary number for each tag, which is of course not possible when manually issuing queries. At the same time, the manual processs is more creative resulting in a larger number of different tags. Note that even though the number of instances in NLMaps v4 is only 7 % of that in NLMaps v3, it still contains 1.3 times as many different tags.

In conclusion, the annotation process was unfortunately only done by male participants and the question of how to handle diverging MRL queries for similar NL queries deserves further research. Nevertheless, the result is a challenging dataset with a large number of tags and more *real* user queries than any of the previous NLMaps datasets.

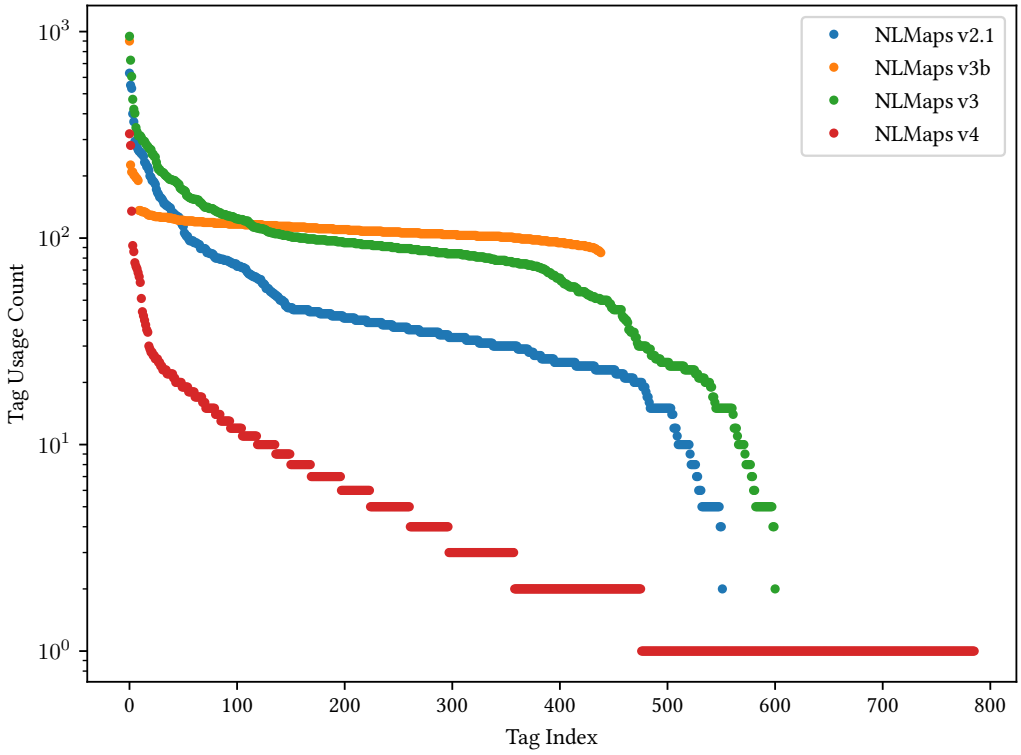



Figure 5.6.: Tag distribution in different NLMaps datasets.

## 5.3. Online Learning Simulation

In this section, we train various models on the new NLMaps v4 dataset and investigate which learning setup is best suited for online learning in the web interface. The model trained on the large NLMaps v3 dataset is used as a starting point for all models in this section.

First, we establish an upper boundary for the fine-tuning on v4 in a regular batch-based offline learning experiment. In the setup called v3 ⟶ v4 in Table 5.4 the model is fine-tuned on v4 for 20 epochs with a batch size of 10. While that model achieves significant accuracy gains on v4, its performance on the old dataset v3 drops starkly. Therefore, another model is trained for 20 epochs (of v4) with batches made up of 5 instances from v4 and 5 instances of v3. This model achieves

the highest accuracy on v4 in this thesis: 58.8 %. The learning curve shown in Figure 5.7 shows that the performance on v3 remains fairly stable throughout the training while the performance on the new v4 is improved. Figure 5.8 shows that errors are still predominantly caused by selecting the wrong tags to query for, even though the tag error rate is greatly reduced with respect to the pre-training model (compare Figure 5.3).

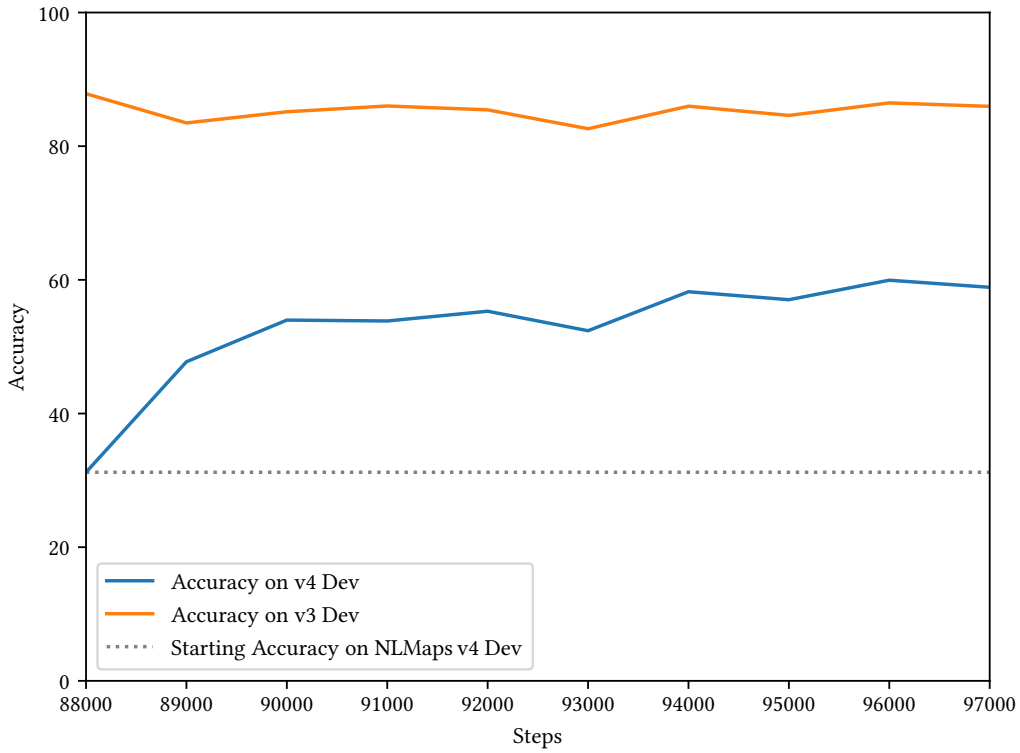



Figure 5.7.: Learning curve of model pre-trained on NLMaps v3 and fine-tuned on a mix of NLMaps v3 and NLMaps v4.

The next experiment is a simulation of online learning: The model trained on NLMaps v3 is fine-tuned on NLMaps v4 by making one pass through the v4 training set and performing one gradient descent step per instance. In the 1-0-0 variant of the experiment, the "minibatch" for calculating the gradient consists of only this instance. In the 1-0-5 variant, 5 instances sampled from NLMaps v3 are added to the minibatch. In the 1-0-5 variant, a further 4 instances sampled from the part of v4 seen at that point are added to the minibatch, which means that the minibatches consist of 5 instances from v3 and 5 instances from v4, just like in the previous offline experiment. Another variation of the experiment is iterating five times instead of once for each instance in v4, which is indicated by the "Iter: 5" in Table 5.4. Note that while the one instance is used in all five iterations in this case,

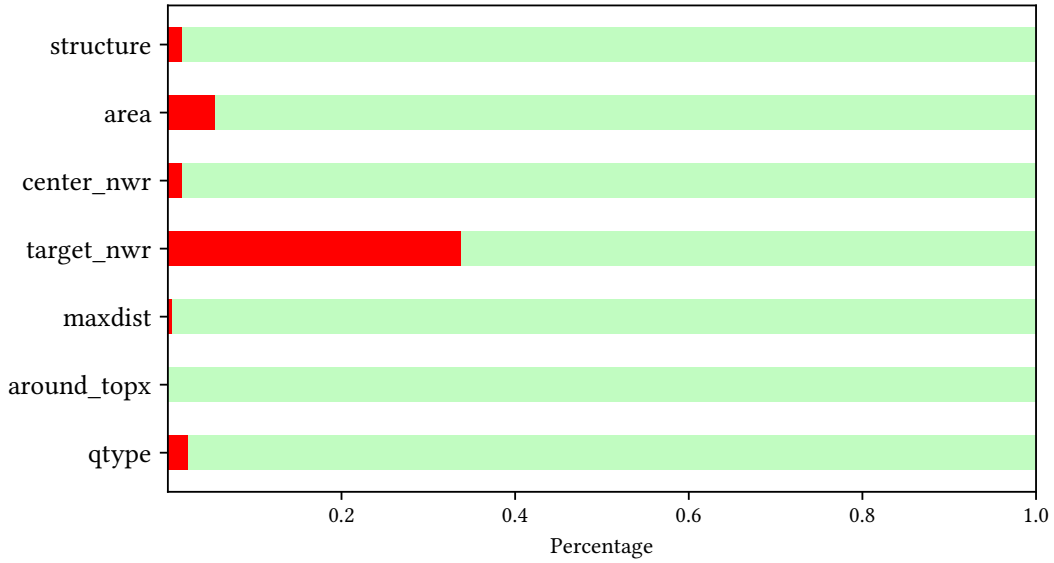



Figure 5.8.: Percentage of queries with an error in a specific operator. The model was pre-trained on v3, fine-tuned on v3 and v4 and tested on v4.

the other samples change with each minibatch. The exact meaning of these four parameters $n_F$, $n_M$, $n_T$ and $I$ is described more formally in Algorithm 1 at the end of Chapter 4.

Since previous work suggests that Adadelta performs as well as (Peris, Cebrián, et al. 2017) or even better than (Turchi et al. 2017; Peris and Casacuberta 2019) Adam for online learning on single instances, the first experiments are conducted using Adadelta with the learning rate set to 0.01 as done by Peris and Casacuberta (2019). The results show that the simple 1-0-0 variant increases the accuracy on v4 by over 10 %, but at the same time the accuracy on v3 degrades by 10 %. Adding instances sampled from v3 to the minibatch almost eradicates this problem, but also impedes adaptation as evidenced by an increase of accuracy on v4 by only 7 %. Adding further memorized v4 instances in the 1-4-5 variant evidently rectifies this problem. By performing 5 iterations per instance, the accuracy on v4 can even be increased by 15 % with respect to the pre-trained model.

Substituting Adam (again with a learning rate of 0.0002) for Adadelta yields little improvement in the 1-0-0 variant. In fact, there is an extreme decrease in accuracy to only 55.2 % on the original v3 data, which is illustrated well in the learning curve in Figure 5.10. Again, adding the v3 instances to the batch alleviates this problem, but not as effectively as with Adadelta. However, this stabilization helps the Adam 1-0-5 variant to reach an v4 accuracy of 45.0 %, significantly outperforming its Adadelta counterpart. The memory of previous v4 instances further increases the accuracy to 51.0 %. Performing 5 iterations per instance yields another accuracy

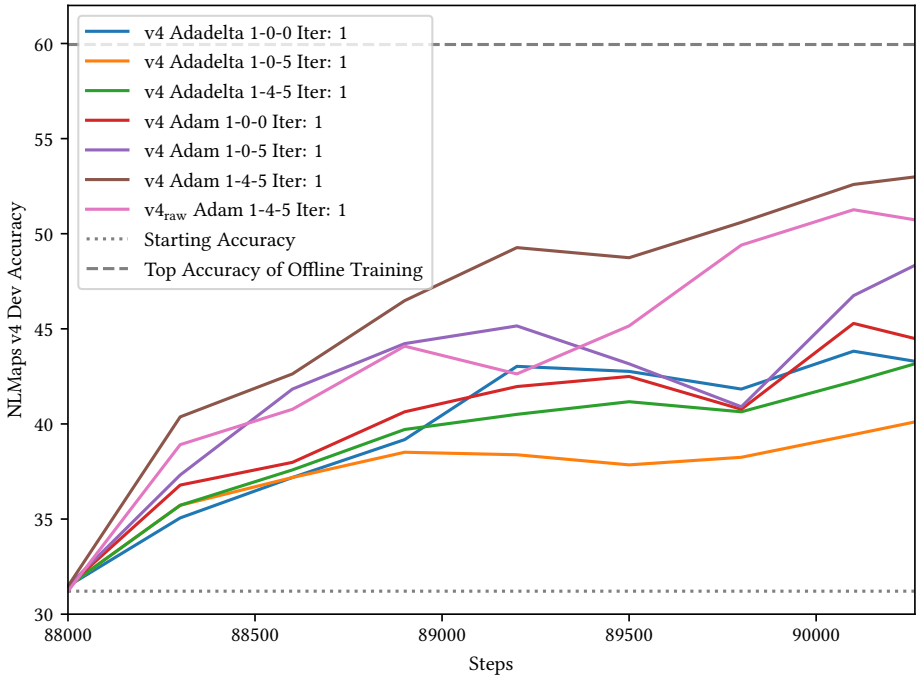



Figure 5.9.: Learning curve on development set of NLMaps v4 during online simulation with one iteration per instance.

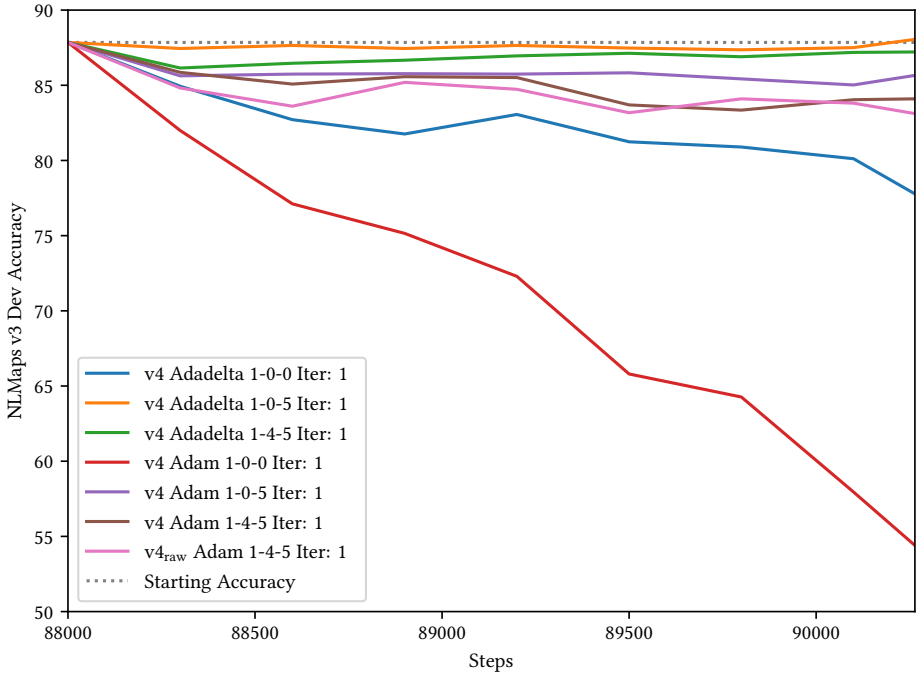



Figure 5.10.: Learning curve on development set of NLMaps v3 during online simulation with one iteration per instance.

| Train \ Test | v2.1 | $v3_{no\text{-}noise}$ | v3 | $v4_{raw}$ | v4 |
|---|---|---|---|---|---|
| v3 | 0.913 | **0.931** | **0.874** | 0.281 | 0.289 |
| v3 → v4 | 0.836 | 0.852 | 0.792 | 0.518 | 0.554 |
| v3 → v3/v4 | **0.914** | 0.923 | 0.865 | **0.548** | **0.588** |
| v3 → v4 Adadelta 1-0-0 Iter: 1 | 0.817 | 0.835 | 0.774 | 0.376 | 0.391 |
| v3 → v3/v4 Adadelta 1-0-5 Iter: 1 | 0.911 | 0.927 | 0.871 | 0.354 | 0.366 |
| v3 → v3/v4 Adadelta 1-4-5 Iter: 1 | 0.905 | 0.924 | 0.866 | 0.381 | 0.396 |
| v3 → v3/v4 Adadelta 1-4-5 Iter: 5 | 0.912 | 0.930 | 0.871 | 0.430 | 0.448 |
| v3 → v4 Adam 1-0-0 Iter: 1 | 0.565 | 0.601 | 0.552 | 0.384 | 0.397 |
| v3 → v3/v4 Adam 1-0-5 Iter: 1 | 0.886 | 0.907 | 0.847 | 0.430 | 0.450 |
| v3 → v3/v4 Adam 1-4-5 Iter: 1 | 0.887 | 0.901 | 0.841 | 0.486 | 0.510 |
| v3 → v3/v4 Adam 1-4-5 Iter: 5 | 0.859 | 0.827 | 0.774 | 0.499 | 0.530 |
| v3 → $v3/v4_{raw}$ Adam 1-4-5 Iter: 1 | 0.880 | 0.884 | 0.831 | 0.483 | 0.490 |

Table 5.4.: Performance of fine-tuned parsers.

increase of 2 %, but in the case of Adam this comes with a 7 % decrease on v3.

While all of the online learning setups are able to increase the accuracy on the new NLMaps v4 dataset, it is a challenge not to degrade the model on the original data. It proves beneficial to include instances from the original data as well as from memorized new instances in each update in order to make the largest gains in accuracy on the new data while preserving old performance as well as possible. Overall, Adadelta is the more conservative optimizer in this regard, which comes at a significantly worse ability to adapt to the new data. In contrast, Adam adapts better, but is very prone to overfitting on the new data when not provided with original data. It also turns out not to have any advantage over Adadelta when the minibatch consists only of the one new instance, which is in line with findings in previous work. Furthermore, using a single instance in five consecutive batches leads to overfitting even when original data is added. Most likely, this is due to Adam's first-order momentum which builds up over these batches.

Overall, the online learning is successful, but still not as effective as traditional offline learning from a mix of v3 and v4, which achieves an accuracy which is higher by 5.8 %. But note that the offline learning was run for 20 epochs while the online learning simulations made only one pass over the training data. Compare the number of steps in Figures 5.7 and 5.9 for this.

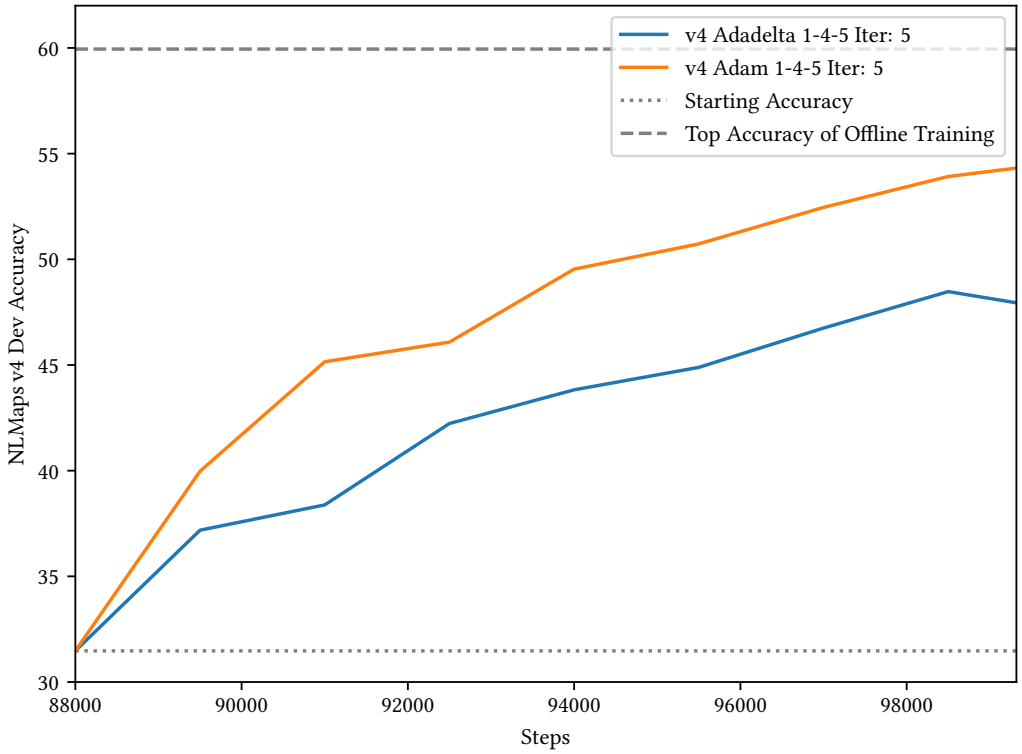



Figure 5.11.: Learning curve on development set of NLMaps v4 during online simulation with five iterations per instance.

## 5.4. Qualitative Longitudinal Analysis

In order to analyze what was learned by adapting on v4, we conduct a longitudinal analysis by observing the predictions of the model called "v3 → v3/v4 Adam 1-4-5 Iter: 1" in Table 5.4 made during the online learning pass *before* each update step. More specifically, we track the performance on twenty tags that were randomly sampled from all tags occurring at least twice in the v4 training set. Figure 5.12 plots the success of the predictions for each tag across the pass over v4 meaning that the instance index on the x axis is the number of instances that have been processed at that point. Note that the plotted tags will in general not be the only target tags in an MRL. E.g., `healthcare=optometrist` usually occurs in a union with `shop=optician` and `sport=cricket` often occurs in an intersection with either `leisure=stadium` or `leisure=pitch`.

The results of the longitudinal analysis are mixed: There are several tags like `shop=tattoo`, `sport=cricket`, `sport=ice_skating` and `cuisine=mexican` where the predictions feature a noticeable improvement over time, but there are also tags like `shop=butcher` where there is no clear improvement. This discrepancy can be understood by taking a closer look at the difference between queries involving

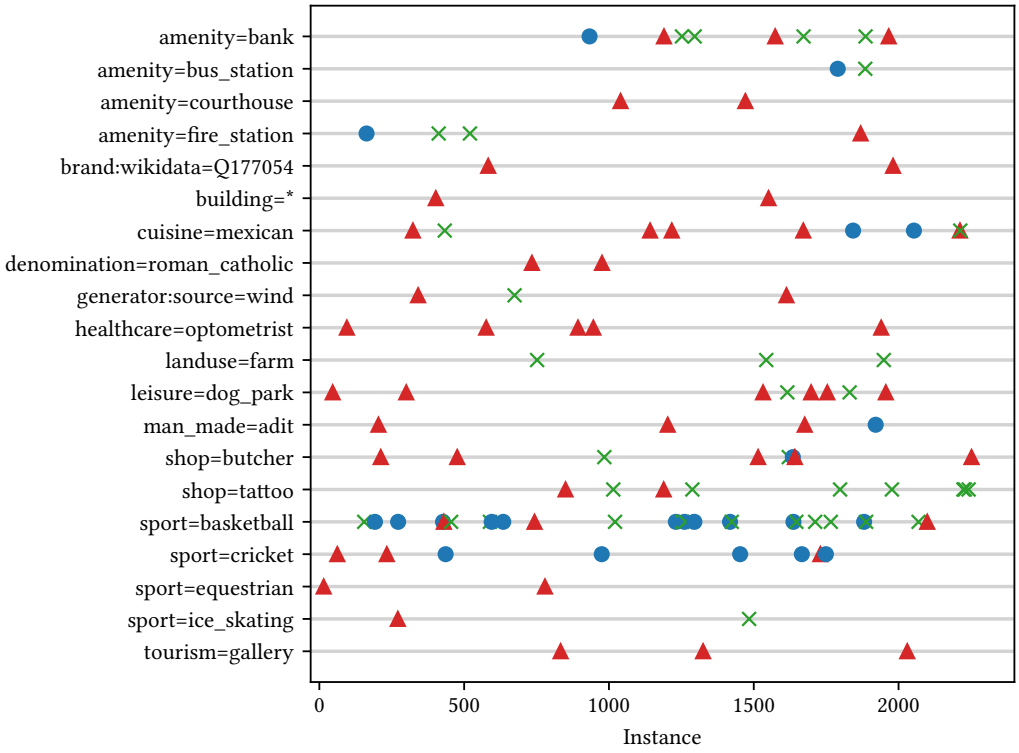



Figure 5.12.: Longitudinal analysis of predictions for twenty randomly selected tags during the online learning simulation on v4. Green crosses mean that the model's prediction is completely correct, blue circles mean that the tag in question was correctly predicted but there is some other error with the prediction and red triangles mean that the tag in question is not present in the prediction (implying also that the prediction is incorrect).

`tattoo` and `butcher`. While the NL queries asking for the former all involve the token *'tattoo'* (*'i want a tattoo ...'*, *'... closest tattoo studio ...'*, etc.) and the MRL queries' only target tag is `shop=tattoo`, the situation is more complex for butchers on both the NL and the MRL side: As shown in Figure 5.13, the NL queries may directly ask for butchers, but they may also ask for places to buy meat or even sausages. As a further complication, the MRL queries when asking for butchers and meat shops will only target the `shop=butcher` tag, but MRL queries when asking for places to buy meat or sausages will additionally target the `shop=supermarket` tag.

In the case of `amenity=fire_station`, the model fails to correctly use the tag for a fourth time after three successes and instead uses the non-existent `amenity=firestation` (dropping the underscore). This is one of several cases that illustrate the brittleness of the character-based model.

When is the meat shop next to Residencial Ibirapuera open?
meat shops in Auxerre (canton d'Auxerre-3)
show me butchers in Marchais-Beton
where can I buy meat in Babiny I?
Are there any butcher shops in Sommerland that is accessible by wheelchair?
how many butchers in cukurambar mahallesi in Podgaje
where can I buy sausages in Vlkov u Jaroměře?
where can i buy meat in rennes

Figure 5.13.: NL queries in the v4 training set that features `shop=butcher` in the corresponding MRL. Note that the blue instance with butchers in Podgaje is obscured by overlapping in Figure 5.12.

# 6. Discussion and Future Work

We spent significant work on analyzing the original NLMaps v2 dataset and on fixing any of its shortcomings as well as possible. The insight gained by this we also used for generating new synthetic queries. By combining the fixed old data with the new synthetic data into the NLMaps v3 dataset the data available for pretraining an NLMaps model is vastly improved as shown in the experiments conducted in Section 5.1. This success naturally calls for continuing this work by generating even more diverse synthetic data, which can be done by extending the table mapping NL terms to OSM tags or by designing new templates. New templates could incorporate verb-based questions like *'Where can I eat chinese food?'* or take inspiration from the newly collected queries in NLMaps v4.

The annotation experiment was successful for the most part and we now have a natural dataset on which new NLMaps models can be evaluated in a more sensible way than on the flawed and synthetic previous datasets. However, it must be acknowledged that the MRL queries the annotators produced reintroduce some inconsistencies in tag usage that were already observed in NLMaps v2. This is partly due to simple user error (e.g. when an annotator does not know of or does not think of a similar tag for the same thing, like using only `shop=tailor` and forgetting `craft=tailor`), but is partly also rooted in actual vagueness of the NL queries. E.g., what should really be selected when asking for a place to buy cigarettes? Only tobacco shops or also cigarette vending machines or even all kinds of places that might sell cigarettes like supermarkets or kiosks?

Two different approaches for handling this tag inconsistency come to mind: In a more "authoritarian" approach, an abstraction layer between OSM and MRL could be introduced. This could take the form of a table mapping concepts like "tailor" or "buy cigarettes" – which should then appear in the MRL – to a manually maintained OSM meaning like `or(shop=tailor, craft=tailor)` or `or(shop=tobacco, vending=cigarettes)`, respectively. But maintaining this mapping would amount to a lot of (semi-)manual work and it would mean losing the advantage of directly using OSM tags, which most OSM users are somewhat familiar with.

In contrast, the second approach is guided by thr MRL queries' denotation:

When several users issue queries with some minimum amount of overlap in their result sets after interpretation, this – perhaps together with a semantic similarity of the NL queries – can be taken as a sign that the users are actually asking for the same thing. It is then left for a system to observe which MRL formulation is the most popular (or – in a recall-focused approach – the most inclusive) and regard this as the canonical MRL.

However, even the NLMaps $v4_{raw}$ with its inconsistencies is shown in our simulations to be very usable for improving the parser in an online learning setting. Not surprisingly, manually removing the inconsistencies as well as possible yielded even better results. The Adam optimizer is shown to adapt faster to new examples while Adadelta is better suited to preserve the performance on the pre-training dataset. It must be noted that we did not conduct any true online learning experiment where the annotators would have made their annotations with the model changing over time. Moreover, albeit the online simulations did improve the accuracy on the new data, they were outperformed signifcantly by the simple offline batch learning.

Regardless of the learning setup, the main challenge of any NLMaps dataset remains predicting the correct tag combination while our models make only few errors in other parts of the MRL. Aside from collecting a lot more training data to make the tag distribution in the training set less sparse, zero-shot learning is a direction which could be explored. In one approach, the tag descriptions from the OSM wiki or even the whole wiki pages could be used as a knowledge source for learning about the meaning of tags, perhaps in a similar approach as in CLIP (Radford et al. 2021).

Part of the reason for the fact that selecting the correct tags poses the main challenge is that the MRL structure is actually very simple. In fact, it is too simplistic for representing some queries that OSM could answer. For example, it is not possible to ask for places whose name (or description or any other tag) includes a certain substring nor is it possible to query places which are *not* tagged with a certain tag. And also referring to one's own geographical position isn't possible resulting in the situation that trivial NL queries like *'closest bus stop near me'* have no MRL equivalent. These issues serve as pointers on how to extend the current MRL.

Due to its success in previous work, a character-based model was used for all work in this thesis, which has the advantages that no NER system is necessary for recognizing named entities and copying them to the MRL and that new tags can easily be accommodated without the need to add them to the target vocabulary.

The downside of the character-based model is that it is fairly sensitive to spelling variations on the NL side. As observed in the experiments, the spelling *'fire station'* was correctly mapped to `amenity=fire_station`, but the spelling variation *'firestation'* led to the model hallucinating the tag `amenity=firestation`. A related problem is that the character-based model will probably not be able to take advantage of semantic similarity of NL queries or even single words. The time is ripe for a modern seq-to-seq model operating on subword units like BPE at least on the NL side with a pointer mechanism (See et al. 2017) for copying the location names from NL to MRL. To leverage semantic similarity between terms and even whole NL queries, pre-trained language representations should be incorporated in a similar way as done by Chen et al. (2020).

As of now, NLMaps is unfortunately only available in English. However, the relative simplicity of most NL queries should make it possible to translate them into other languages in order to create datasets for parsing NL queries in languages other than English.

# 7. Conclusion

In this thesis, we conducted a detailed analysis of the original NLMaps v2 dataset finding several shortcomings, many of which were introduced by a flawed approach of generating synthetic data. We fixed these shortcomings as well as possible and extended the dataset by generating a linguistically diverse dataset using probabilistic templates. Training on the extended dataset greatly improves accuracy on unseen data, especially by making the resulting parser robust against new location names.

We built a web interface for issuing NL queries, which can also be used to correct wrong parses and which is capable of training the parser on the new feedback in an online fashion. With the help of hired annotators, we created the first large NLMaps dataset consisting of real user queries. This new dataset was used to demonstrate the effectiveness of our online learning setup in various simulations, although traditional offline learning still proved superior.

In our experiments and discussion, we gained new insight into what the main challenges of the NLMaps task are and proposed various directions for subsequent research.

# A. Acknowledgements

I want to thank my supervisor Prof. Dr. Stefan Riezler for always being ready to discuss the state of my work, for providing me with helpful guidance about which approaches to pursue and for his help in finding relevant related work. Additionally, I want to thank him for making the annotation experiment possible by financially enabling it with funds from the Google-supported project "Learning to Negotiate Answers in Multi-Pass Semantic Parsing".

Furthermore, I also want to thank Raphael Schumann and especially Michael Staniek for valuable comments on the ongoing research in numerous discussion sessions. Michael also made his parsing model available to me, which was used as one of the foundations of this thesis. An even more integral foundation is of course Carolin Lawrence's pioneering work on NLMaps, without which this thesis would not exist today.

Finally, I would like to acknowledge the great work of the motivated annotators from all over the world who participated in my annotation project, including the OSM mappers Benjámin Zachár, Mateusz Konieczny, Rabin Ojha. Especially Mateusz and another very experienced mapper (here unnamed) were a great help in debugging issues with the web interface, in discussing finer points of OSM tagging questions and by providing valuable hints for how to improve NLMaps in the future.

# B. Annotation Guidelines

## B.1. Requirements

The annotation website[1] is meant to be used with a recent version of a modern Browser. That means Firefox, Safari, Chrome or any other Chromium derivates (such as recent MS Edge). It is not optimized for mobile use, so please use a desktop computer.

## B.2. Principles

In essence, we need a dataset that fulfills three criteria:

1. The natural language (NL) queries should be linguistically diverse, i.e. a mix of short search-engine-style queries and full questions like you would ask another person.
2. The queries should cover a large part of the commonly used OpenStreetmap (OSM) tags that are relevant for our system.
3. The queries should cover the most useful OSM tags in particular depth. E.g., asking for restaurants and shops is arguably one of the most useful areas.

## B.3. Linguistic Diversity

When entering queries, please diversify your language use. Valid variations of the same question include:

- *'closest swimming pool near Eiffel Tower in Paris'*
- *'In Paris, give me the swimming pool that is closest to the Eiffel Tower!'*
- *'Closest place where I can take a swim near Eiffel Tower in Paris'*

[1] `https://nlmaps.gorgor.de/`

However, keep it natural and don't make your queries artificially complex. If a particular query style comes more natural to you, it's alright to use it more often. Just make sure that not all of you queries look alike.

Similarly, avoid using the same place name more than a couple of times. Also use place names in other languages than German and English, e.g. *'České Budějovice'* or *'València'*.

It's **very important** that you don't only base your wording on the name of the OSM tags. E.g., for `highway=speed_camera` you can ask for a *'speed camera'*, but you can also ask for a *'speed trap'* or a *'radar trap'*.

## B.4. Tag Diversity and Depth

Take a *'quick'* look at the most important OSM features[2] to get a feel for what things you can ask for in OSM. You can use this as an inspiration if you run out of ideas about what to ask. Choose tags that you find relevant.

**Beware of the `building=*` tag!** It is used to tag what the building was built as, not to tag its current use. E.g., a place tagged `building=church` may not be a church anymore; churches are tagged with the tags `amenity=place_of_worship` + `religion=christian`. If in doubt, don't use the `building=*` tag, at all.

In general, enter more than one query for a chosen tag or tag combination, especially if the system fails answering the query correctly.

### B.4.1. Most Relevant Keys

Please enter at least the given amount of queries for each of the following. The linked wiki pages give you a feel for what the most common tags in each category are.

- `shop=*`: 30.
- `leisure=*`: 20.
- `sport=*`: 20.
- `craft=*`: 20.
- `man_made=*`: 10.
- `amenity=cafe`: 10.

[2] `https://wiki.openstreetmap.org/wiki/Map_features`

- `amenity=restaurant`: 10.
- `amenity=fast_food`: 10.
- `cuisine=*`: 20.
- `diet:*=*`: 20.
- `wheelchair=yes`: 20. Just sprinkle phrases like *'wheelchair-accessible [place]'* or *'[places] that are wheelchair-accessible'* into your queries now and then.

Some tags *can and should* be combined. E.g., use `shop=massage,wheelchair=yes` for wheelchair-accessible massage shops or `club=sport,sport=tennis` for tennis clubs. But use only `sport=tennis` if you're just asking for places to play tennis at.

Especially the `cuisine=*` and `diet:*=*` tags can be combined productively. Some examples:

- `cuisine=japanese`: Places serving japanese food
- `cuisine=japanese,amenity=fast_food`: Fast food restaurants serving japanese food
- `or(diet:vegan=yes,diet:vegan=only),amenity=cafe`: Vegan cafes
- `or(diet:gluten_free=yes,diet:gluten_free=only), cuisine=burger,amenity=restaurant`: Restaurants serving gluten-free burgers

## B.5. Miscellaneous

- Sometimes deciding between QType `findkey('name')` and `latlong` is not obvious. By convention:
  - 'Which/What *restaurants/museums/etc. ...*': `findkey('name')`
  - 'Name *restaurants/museums/etc. ...*': `findkey('name')`
  - '*What are restaurants/museums/etc. ...* called?': `findkey('name')`
  - 'Give/Tell *(me) the* names *of restaurants/museums/etc. ...*': `findkey('name')`
  - 'Show/Give/Tell *(me) restaurants/museums/etc. ...*': `latlong`
  - 'Where *are restaurants/museums/etc. ...*': `latlong`

  - 'Location/Coordinates *of restaurants/museums/etc. ...*': `latlong`

- Don't repeat the same query with only a different location. Adjust the wording, as well.
- Some queries will not return results even if they are correct (e.g. rare tags like gluten-free etc.). Please base your judgement primarily on the mrl, not on the answer or map.
- Avoid querying too much data (*'trees in Berlin'*) or too large areas (*'restaurants in Bangladesh'*) to put less stress on servers and your browser.
- *'Show all restaurants in X that are wheelchair-accessible!'*: Target tags include `wheelchair=yes`, QType is 'latlong'
- *'Is X accessible by wheelchair?'*: Use QType `findkey('wheelchair')`, no `wheelchair=*` target tag
- Questions looking for the closest thing to some other thing should always have a maxdist of `DIST_INTOWN`. In theory, this doesn't make sense. It's just a limitation of the current system.
- Use the appropriate maxdist value according to the table in chapter 4 of the tutorial[3]. E.g., when using the word "near" in your query, use `DIST_INTOWN`.

## B.6. Counting Annotations

A natural language query and the corresponding mrl are considered one *complete annotation*. If you don't know what the correct mrl looks like, you can choose "Wrong, but I cannot help" and it will be considered an *incomplete annotation*. This is still valuable; four incomplete annotations will count as one complete annotation.

[3] `https://nlmaps.gorgor.de/tutorial?chapter=4`

# Bibliography


Andreas, Jacob, Andreas Vlachos, and Stephen Clark (2013). "Semantic Parsing as Machine Translation". In: *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*. ACL (Sofia, Bulgaria).

Arun, Abhishek and Philipp Koehn (2007). *Online Learning Methods For Discriminative Training of Phrase Based Statistical Machine Translation*.

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio (2015). "Neural Machine Translation by Jointly Learning to Align and Translate". In: *Proceedings of the 3rd International Conference on Learning Representations*. ICLR (San Diego, CA, USA).

Barrachina, Sergio, Oliver Bender, Francisco Casacuberta, Jorge Civera, Elsa Cubel, Shahram Khadivi, Antonio Lagarda, Hermann Ney, Jesús Tomás, Enrique Vidal, and Juan-Miguel Vilar (2009). "Statistical Approaches to Computer-Assisted Translation". In: *Computational Linguistics* 35.1, pp. 3–28.

Budhathoki, Nama R. (2010). "Participants' motivations to contribute geographic information in an online community". Dissertation. University of Illinois at Urbana-Champaign.

Casacuberta, Francisco, Jorge Civera, Elsa Cubel, Antonio L. Lagarda, Guy Lapalme, Elliott Macklovitch, and Enrique Vidal (2009). "Human interaction for high quality machine translation". In: *Communications of the ACM* 52.10, pp. 135–138.

Chen, Xilun, Asish Ghoshal, Yashar Mehdad, Luke Zettlemoyer, and Sonal Gupta (2020). "Low-Resource Domain Adaptation for Compositional Task-Oriented Semantic Parsing". In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*. EMNLP (Online), pp. 5090–5100.

Cho, Kyunghyun, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio (2014). "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*. EMNLP (Doha, Qatar), pp. 1724–1734.

Das, Maitraye, Brent Hecht, and Darren Gergle (2019). "The Gendered Geography of Contributions to OpenStreetMap: Complexities in Self-Focus Bias". In: *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. CHI (Glasgow, United Kingdom), pp. 1–14.

Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova (2019). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (Minneapolis, MN, USA), pp. 4171–4186.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). *Deep Learning*. MIT Press. URL: `https://www.deeplearningbook.org/` (visited on 04/19/2021).

Green, Bert F., Alice K. Wolf, Carol Chomsky, and Kenneth Laughery (1961). "Baseball: An automatic question answerer". In: *Proceedings of the Western Computing Conference*. Vol. 19, pp. 219–224.

Gwerder, Simon (2014). *Tag-Suchmaschine und Thesaurus für OpenStreetMap*. Student Research Project. HSR University for Applied Sciences Rapperswil. URL: `https://eprints.ost.ch/id/eprint/409/` (visited on 04/16/2021).

Haas, Carolin and Stefan Riezler (2016a). "A Corpus and Semantic Parser for Multilingual Natural Language Querying of OpenStreetMap". In: *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. NAACL (San Diego, CA, USA), pp. 740–750.

Hemphill, Charles T., John J. Godfrey, and George R. Doddington (1990). "The ATIS Spoken Language Systems Pilot Corpus". In: *Speech and Natural Language: Proceedings of a Workshop Held at Hidden Valley*. HLT (Hidden Valley, PA, USA).

Hochreiter, Sepp and Jürgen Schmidhuber (1993). "Long Short-Term Memory". In: *Neural Computation* 9.8, pp. 1735–1780.

Hwang, Wonseok, Jinyeong Yim, Seunghyun Park, and Minjoon Seo (2019). "A Comprehensive Exploration on WikiSQL with Table-Aware Word Contextualization". In: *Workshop: Knowledge Representation and Reasoning Meets Machine Learning*. NeurIPS (Vancouver, Canada).

Jurafsky, Dan and James H. Martin (2021). "Speech and Language Processing". Draft of the 3rd edition. URL: `https://web.stanford.edu/~jurafsky/slp3/` (visited on 04/30/2021).

Karimova, Sariya, Patrick Simianer, and Stefan Riezler (2018). "A User-Study on Online Adaptation of Neural Machine Translation to Human Post-Edits". In: *Machine Translation* 32, pp. 309–324.

Kennardi, Alvin, Gabriela Ferraro, and Qing Wang (2019). "Domain Adaptation for Low-Resource Neural Semantic Parsing". In: *Proceedings of the The 17th Annual Workshop of the Australasian Language Technology Association*. ALTA (Sydney, Australia), pp. 87–93.

Kreutzer, Julia, Jasmijn Bastings, and Stefan Riezler (2019). "Joey NMT: A Minimalist NMT Toolkit for Novices". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing: System Demonstrations*. EMNLP-IJCNLP (Hong Kong, China), pp. 109–114.

Kreutzer, Julia, Nathaniel Berger, and Stefan Riezler (2020). "Correct Me If You Can: Learning from Error Corrections and Markings". In: *Proceedings of the 22nd Annual Conference of the European Association for Machine Translation*. EAMT (Lisboa, Portugal), pp. 135–144.

Kreutzer, Julia, Stefan Riezler, and Carolin Lawrence (2020). "Learning from Human Feedback: Challenges for Real-World Reinforcement Learning in NLP". In: *Workshop: The Challenges of Real World Reinforcement Learning*. NeurIPS (Online).

Kreutzer, Julia, Artem Sokolov, and Stefan Riezler (2017). "Bandit Structured Prediction for Neural Sequence-to-Sequence Learning". In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*. ACL (Vancouver, Canada), pp. 1503–1513.

Lam, Tsz Kin, Julia Kreutzer, and Stefan Riezler (2018). "A Reinforcement Learning Approach to Interactive-Predictive Neural Machine Translation". In: *Proceedings of the 21st Annual Conference of the European Association for Machine Translation*. EAMT (Alacant, Spain), pp. 169–178.

Lawrence, Carolin and Stefan Riezler (2016). "NLmaps: A Natural Language Interface to Query OpenStreetMap". In: *Proceedings of the 26th International Conference on Computational Linguistics*. COLING (Osaka, Japan), pp. 6–10.

Lawrence, Carolin and Stefan Riezler (2018). "Improving a Neural Semantic Parser by Counterfactual Learning from Human Bandit Feedback". In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*. ACL (Melbourne, Australia), pp. 1820–1830.

Lawrence, Carolin, Artem Sokolov, and Stefan Riezler (2017). "Counterfactual Learning from Bandit Feedback under Deterministic Logging: A Case Study in Statistical Machine Translation". In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. EMNLP (Copenhagen, Denmark), pp. 2566–2576.

Lechner, Marco (2011). "Nutzungspotentiale crowdsource-erhobener Geodaten auf verschiedenen Skalen". Dissertation. University of Freiburg.

Lewis, Mike, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer (2020). "BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension". In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics* (Online), pp. 7871–7880.

Liang, Percy, Alexandre Bouchard-Côté, Dan Klein, and Ben Taskar (2006). "An End-to-End Discriminative Approach to Machine Translation". In: *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*. ACL/COLING (Sydney, Australia), pp. 761–768.

Lin, Xi Victoria, Richard Socher, and Caiming Xiong (2020). "Bridging Textual and Tabular Data for Cross-Domain Text-to-SQL Semantic Parsing". In: arXiv: `2012.12627 [cs.CL]`.

Mnih, Volodymyr, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Tim Harley, Timothy P. Lillicrap, David Silver, and Koray Kavukcuoglu (2016). "Asynchronous Methods for Deep Reinforcement Learning". In: *Proceedings of the 33rd International Conference on International Conference on Machine Learning*. ICML (New York, NY, USA), pp. 1928–1937.

Mollá, Diego and José Luis Vicedo (2007). "Question Answering in Restricted Domains: An Overview". In: *Computational Linguistics* 33.1, pp. 41–61.

Murphy, Kevin Patrick (2021). *Probabilistic Machine Learning: An Introduction*. MIT Press. URL: `http://probml.ai/` (visited on 04/19/2021).

Nguyen, Khanh, Hal Daumé III, and Jordan Boyd-Graber (2017). "Reinforcement Learning for Bandit Neural Machine Translation with Simulated Human Feedback". In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. EMNLP (Copenhagen, Denmark), pp. 1464–1474.

Ortiz-Martínez, Daniel (2016). "Online Learning for Statistical Machine Translation". In: *Computational Linguistics* 42.1, pp. 121–161.

Ortiz-Martínez, Daniel, Ismael García-Varea, and Francisco Casacuberta (2010). "Online Learning for Interactive Statistical Machine Translation". In: *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*. NAACL (Los Angeles, CA, USA), pp. 546–554.

Peris, Álvaro and Francisco Casacuberta (2019). "Online learning for effort reduction in interactive neural machine translation". In: *Computer Speech & Language* 58, pp. 98–126.

Peris, Álvaro, Luis Cebrián, and Francisco Casacuberta (2017). "Online Learning for Neural Machine Translation Post-editing". In: arXiv: `1706.03196 [cs.LG]`.

Radford, Alec, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever (2021). "Learning Transferable Visual Models From Natural Language Supervision". In: arXiv: `2103.00020 [cs.CV]`.

Schmidt, Manuela and Silvia Klettner (2013). "Gender and Experience-Related Motivators for Contributing to OpenStreetMap". In: *International Workshop on Action and Interaction in Volunteered Geographic Information* (Leuven, Belgium), pp. 13–18.

See, Abigail, Peter J. Liu, and Christopher D. Manning (2017). "Get To The Point: Summarization with Pointer-Generator Networks". In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*. ACL (Vancouver, Canada), pp. 1073–1083.

Shannon, Claude Elwood (1948). "A Mathematical Theory of Communication". In: *The Bell System Technical Journal* 27.3, pp. 379–423.

Shaw, Peter, Ming-Wei Chang, Panupong Pasupat, and Kristina Toutanova (2020). "Compositional Generalization and Natural Language Variation: Can a Semantic Parsing Approach Handle Both?" In: arXiv: `2010.12725 [cs.CL]`.

Stahlberg, Felix (2020). "Neural Machine Translation: A Review and Survey". In: *Journal of Artificial Intelligence Research* 69, pp. 343–418.

Staniek, Michael (2020). "Towards Error-Aware Interactive Semantic Parsing". M.A. Thesis. Heidelberg University.

Stark, Hans-Jörg (2010). *Empirische Untersuchung der Motivation von Teilnehmenden bei der freiwilligen Erfassung von Geodaten*. Presentation.

Turchi, Marco, Matteo Negri, M. Amin Farajian, and Marcello Federico (2017). "Continuous Learning from Human Post-Edits for Neural Machine Translation". In: *The Prague Bulletin of Mathematical Linguistics* 108, pp. 233–244.

Wang, Bailin, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson (2020). "RAT-SQL: Relation-Aware Schema Encoding and Linking for Text-to-SQL Parsers". In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. ACL (Online), pp. 7567–7578.

Watanabe, Taro, Jun Suzuki, Hajime Tsukada, and Hideki Isozaki (2007). "Online Large-Margin Training for Statistical Machine Translation". In: *Proceedings of*

*the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*. EMNLP/CoNLL (Prague, Czech Republic), pp. 764–773.

Williams, Ronald J. (1992). "Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning". In: *Machine Learning* 8 (3–4), pp. 229–256.

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

Xu, Xiaojun, Chang Liu, and Dawn Song (2017). "SQLNet: Generating Structured Queries From Natural Language Without Reinforcement Learning". In: arXiv: `1711.04436 [cs.CL]`.

Yu, Tao, 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), pp. 3911–3921.

Zelle, John M. and Raymond J. Mooney (1996). "Learning to Parse Database Queries Using Inductive Logic Programming". In: *Proceedings of the Thirteenth National Conference on Artificial Intelligence*. AAAI (Portland, OR, USA), pp. 1050–1055.

Zhong, Victor, Caiming Xiong, and Richard Socher (2017). "Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning". In: arXiv: `1709.00103 [cs.CL]`.

# Online Resources


Astrakhan, Yuri. *Sophox*. URL: https://wiki.openstreetmap.org/wiki/Sophox (visited on 04/08/2021).

Facebook Inc. *PyTorch*. URL: https://pytorch.org (visited on 04/16/2021).

*GeoChicas*. URL: https://wiki.openstreetmap.org/wiki/GeoChicas (visited on 04/30/2021).

Gwerder, Simon. *OSM TagFinder*. URL: https://tagfinder.herokuapp.com/ (visited on 04/16/2021).

Haas, Carolin and Stefan Riezler (2016b). *NLmaps*. URL: https://www.cl.uni-heidelberg.de/statnlpgroup/nlmaps/ (visited on 04/08/2021).

Hipp, D. Richard, Dan Kennedy, and Joe Mistachkin. *SQLite*. URL: https://sqlite.org/ (visited on 04/16/2021).

Hoffmann, Sarah, Marc Tobias Metten, and Brian Quinion. *Nominatim*. URL: https://nominatim.org/ (visited on 04/08/2021).

Kreutzer, Julia, Jasmijn Bastings, and Stefan Riezler. *Joey NMT*. URL: https://github.com/joeynmt/joeynmt (visited on 04/16/2021).

Lawrence, Carolin. *Overpass NLmaps*. URL: https://github.com/carhaas/overpass-nlmaps (visited on 04/08/2021).

Mocnik, Franz-Benjamin (2017). *OSMPythonTools*. URL: https://github.com/mocnik-science/osm-python-tools (visited on 04/02/2021).

*Nominatim Special Phrases*. URL: https://wiki.openstreetmap.org/wiki/Nominatim/Special_Phrases/EN (visited on 04/08/2021).

Olbricht, Roland. *Overpass API*. URL: https://overpass-api.de/ (visited on 04/08/2021).

Open Data Commons. *Open Data Commons Open Database License (ODbL)*. URL: https://opendatacommons.org/licenses/odbl/ (visited on 04/08/2021).

OpenStreetmap Foundation. *OpenStreetMap*. URL: https://www.openstreetmap.org/about (visited on 04/08/2021).

*OSM Mailing List* talk. URL: https://lists.openstreetmap.org/listinfo/talk (visited on 04/30/2021).

Pallets Projects. *Flask*. URL: https://palletsprojects.com/p/flask/ (visited on 04/16/2021).

Raifer, Martin. *Overpass Turbo*. URL: https://overpass-turbo.eu/ (visited on 04/08/2021).

Schaub, Tim, Andreas Hocevar, Tom Payne, Frédéric Junod, Eric Lemoine, and Christopher Schmidt. *OpenLayers*. URL: https://openlayers.org/ (visited on 04/16/2021).

*Subreddit r/openstreetmap*. URL: https://www.reddit.com/r/openstreetmap (visited on 04/30/2021).