PS-Radar: A High-Precision Geoparsing Solution for Real-Time Location Analysis

Anonymous ACL submission

Abstract

PS-Radar is an advanced geoparsing solution utilizing OSM Nominatim to analyze text messages and generate a curated list of potential locations mentioned within. These results are then disambiguated and ranked based on their likelihood of accurately representing the target locations. Our system integrates data from OSM and other sources, enabling real-time map visualization, geofencing, area-based filtering, and alerting for monitoring social media message streams. Furthermore, we introduce the SonarChallenge dataset, comprising 1489 annotated messages containing location references. In our evaluation using the SonarChallenge dataset, our solution achieves an 88.49% recall rate for identifying mentioned locations and a 92.51% precision rate for accurately pinpointing the output locations.

1 Introduction

In the era of big data, much information exists as unstructured text, notably on platforms like X (formerly Twitter), where users post an average of 6,000 tweets per second (liv). Some tweets are crucial for identifying and reporting events such as terrorist attacks, accidents, infrastructure issues, or natural disasters, where the *location* of the event is critical.

For safety responders, who operate within specific geographical areas, sifting through numerous messages is challenging. Converting unstructured text that includes locations into geographical coordinates can significantly aid in event detection and response. In information retrieval, the finding of location strings is known as location entity recognition or location extraction.

The initial step involves identifying the substring that denotes the location within the message. Various methods exist for this, including co-occurrence or n-gram matching against databases or gazetteers like OpenStreetMaps or GeoNames, Named Entity Recognition (NER) techniques, and the Question-Answering (QA) models used by our AnonymousSubmission Key Insights engine. These methods output a string representing the location, such as *"in the museum"* from *"A fire started in the museum."*, which may not always contain a proper noun toponym.

The next step is toponym resolution or place name disambiguation, where the geographical coordinates of the identified toponyms are determined, aligning with the definition of geoparsing as described in The GIS Encyclopedia (gis).

> Geoparsing handles ambiguous references in unstructured discourse, such as *"Al Hamra"*, which is the name of several places, including towns in both Syria and Yemen.

Geoparsing is complex, particularly with repeated place names. For instance, *"I love Paris"* could refer to Paris, France, or Paris, Ontario, Canada, based on context. An effective geoparser must discern the most likely geographical coordinates using surrounding contextual information.

This paper introduces PS-Radar, a light and accurate geoparsing algorithm, and the SonarChallenge, a benchmark dataset for evaluating unsupervised geoparsers. PS-Radar utilizes Nominatim as its primary OpenStreetMap (OSM) search engine, with Geonames as a secondary parser. The algorithm operates unsupervised and is compatible with English and Dutch, with potential for adaptation to other languages supported by Nominatim. PS-Radar processes up to 50 messages per second on a single process setup, achieving a top score of 80.90% in the SonarChallenge, with precision of 92.51% and recall of 88.49%.

The paper is structured as follows: Section 2 reviews related work in geoparsing. Section 3 details the data types used and introduces the SonarChallenge benchmark. Section 4 describes our algorithmic solution, PS-Radar. Section 5 discusses the performance of PS-Radar and its results on the SonarChallenge dataset, comparing it with other tools and analyzing its commercial applicability. Section 6 concludes the paper. Finally, Section 7 outlines limitations and future research avenues.

2 Related Work

2.1 Location Disambiguation

Toponym extraction is well-explored in NLP, with Named Entity Recognition (NER) being prominent for location extraction.

There are various approaches to derive final coordinates from raw location strings. Most of the final solutions are integrations with a blend of rule-based, ranking, and ML techniques.

Rule-based algorithms, as delineated in foundational works such as Smith (2001), employ heuristics to verify the accuracy of candidates identified by search engines. These methodologies have undergone significant refinement over time and continue to hold relevance in contemporary applications. A notable advancement in this domain is the incorporation of context-based heuristics, further developed by Wang (2010). This approach advocates for the analysis of potential contextual overlaps with other location mentions within a text to resolve ambiguities. For instance, in the sentence *"Oxford Street, my favourite street in London"*, the mention of *"London"* serves to disambiguate the otherwise common street name *"Oxford Street"*.

Ranking algorithms prioritize candidates based on various factors such as population density (Karimzadeh et al., 2019), administrative level (Shahi, 2016), co-occurrences, and proximity to an anchor coordinate. These factors help in determining the most likely location among the identified candidates.

Machine learning algorithms utilize model training to perform candidate selection. These methods integrate confidence filtering and have been implemented using various techniques such as Expectation-Maximization (Manning and Schutze, 1999), Neural Networks (Halterman, 2019), and Support Vector Machines (Avvenuti et al., 2018). Recent advancements include the integration of external databases like DBpedia (Nizzoli et al., 2020) and Wikidata (Laparra and Bethard, 2020), which enhance the accuracy of location identification.

The literature distinguishes methods based on the desired granularity. Gelernter and Balaji (2013) describe techniques tailored for local geoparsing, such as identifying streets and buildings. However, highly localized methods may risk overfitting, as noted by Karimzadeh et al. (2019). The concept of spatial minimality or proximity, suggested by Lieberman et al. (2010), posits that toponyms within a document are likely to form spatial clusters. This leads to methods that favor candidates in close spatial proximity. While these methodologies prioritize minimizing spatial distance, they may do so at the expense of generalization.

2.2 Corpora

Key datasets for testing algorithms include GeoCorpora, a dataset of 6711 tweets, with 2122 containing at least one place name (Wallgrün et al., 2018). Other datasets focus on specific events, such as natural disasters (Middleton et al., 2014; Gelernter and Balaji, 2013).

Dataset curation for geospatial analysis faces challenges like geographical distribution bias, ambiguity in place names, annotation disagreements, and source text quality. Geographical distribution often skews towards North America and Europe (Leetaru et al., 2013; Wallgrün et al., 2018). Ambiguity in place names can lead to non-valid data points or default values based on criteria like population density. Annotation disagreements stem from interpretative variations, and source text quality, particularly from social media, impacts the location accuracy due to errors like typos and misspellings (e.g., *Colosseum* vs. *Coliseum*, *Hawaii* vs. *Hawaai*) .

3 SonarChallenge data

The SonarChallenge dataset is open-source and available for academic research[1]. It includes 1489 tweets, each containing at least one location verified as ground truth, covering various topics. Expert annotators have disambiguated each location, converting them into OSM JSON format. The dataset provides the following structured information[2]:

- `"Message"`: The original tweet text containing the location(s).
- `"whereEvent"`: The specific text segment for geoparsing analysis.
- `"NER"`: Named Entity Recognition results, following IOB format, including the original token string, text span, entity type, and confidence score.
- `"Language"`: The ISO language code of the tweet.

The dataset features tweets mentioning locations in 60 countries, primarily the Netherlands (46.5%), the USA (17.1%), and the UK (7.1%). It includes 101 distinct location types, such as *administrative*, *residential*, and *suburb*, along with natural elements like water areas, national parks, and peaks.

[1]For participation and evaluation details, contact AnonymousSubmission@AnonymousSubmission.com

[2]Schema shown in Appendix A

3.1 Scoring System

The scoring system rewards precision and penalizes false positives, with the maximum score achieved by perfect alignment with the ground truth. Accurate predictions receive full scores, while partial overlaps (e.g., predicting a town instead of a street) result in proportional score reductions. Predictions more specific than the ground truth (e.g., predicting a street when the actual location is a town) are similarly adjusted. A deviation of up to 500 meters is allowed for roads to account for discrepancies in OSM's lower administrative levels. False positives result in complete score deductions.

The highest attainable score is 9236.5 points[3]. In addition to the primary scoring system, the dataset provides precision, recall, Mean Squared Error (MSE), and Root Mean Squared Error (RMSE) metrics. Error calculations for MSE and RMSE are based on the shortest Haversine distance between false positive coordinates and their actual locations.

4 PS-Radar

PS-Radar combines optimized heuristics with machine learning (ML) to map event locations, using distance and naming similarities. It leverages external curated knowledge and contextual information. The ML model measures the confidence of the algorithm's output, allowing for precision tuning through confidence thresholds.

4.1 Input data processing

The PS-Radar algorithm processes the following inputs:

- **Search String**: Primary text for location extraction, ideally a focused concatenation of words pinpointing the location.
- **NER**: Named Entities of type person (PER) and location (LOC) found in the message.

[3]Perfect algorithmic scores are impossible due to inherent location placement challenges on OSM objects, such as segmented streets.

- **Language**: Language of the message, used to tailor queries to Nominatim.

The preprocessing phase involves several key steps. Initially, stopwords are removed, except those in proper nouns (e.g., "Gulf of Biscay"). Acronyms are then mapped to their full location names using a curated list (e.g., UvA → Universiteit van Amsterdam). Demonyms are substituted with place nouns (e.g., "The Canadian town of Paris" → "Canada Paris"). Additionally, NER person (PER) entities are excluded from the search query to avoid incorrect results.

4.2 Contextual knowledge

Disambiguation in geoparsing is enhanced by analyzing the full textual context of a message. This process includes extracting and analyzing all NER-identified locations, excluding NER person names, applying demonym substitutions, and performing a country-level scan. Using the extracted location strings, a contextual framework is constructed for each message. This framework employs OpenStreetMap (OSM) objects to assess location affinity and rank candidates for geolocating the whereEvent string.

4.3 Candidates dynamic ranking

This stage uses the whereEvent string as the primary geocoding objective.[4] The algorithm initially correlates the whereEvent with contextual NER locations or scanned countries, aiming for overlap at the lowest administrative level. For instance, if the hypothesis suggests a street, the lowest level of affinity would be its neighborhood; for a city, it would be the province, region, or state. The algorithm assigns an internal importance score to rank against other OSM objects retrieved in queries.

Part-of-Speech (POS) tagging helps to detect proper nouns indicative of locations. The whereEvent then undergoes cleaning while preserving linking stopwords.

Example 4.1. Examples of the cleaning phase:

- `in a bar in Rotterdam` → `Rotterdam`
- `around the Tower of London` → `Tower of London`
- `somewhere close to road A5067, in the Trafford Park suburbs of Manchester` → `A5067 Trafford Park Manchester`

This cleaning phase is crucial as the Nominatim engine does not parse the input text, it requires clear toponyms. Indiscriminate deletion of stopwords can negatively impact the results.

Example 4.2. Examples of naive deletion of stopwords:

- **Query:** outside the Stade de France
- **Indiscriminate deletion of stopwords:** `Stade France` ⇒ `Stade, Bû, Dreux, Eure-y-Loir, Centre-Val de Loire, 28410, France` ✗
- **Conservation of linking stopwords:** `Stade de France` ⇒ `Stade de France, Avenue du Stade de France, Franc-Moisin - Bel-Air - Stade de France, Saint-Denis, Sena-Saint Denis, Ile-de-France, 93200, France` ✓

After cleaning, if the whereEvent result is too long[5], the algorithm defers to NER-identified locations. Otherwise, dynamic iteration disambiguates potential locations within the whereEvent. The algorithm uses max n-gram logic, favoring the longest n-gram possible.

Example 4.3. Long match preference:

- **`New York`** → `New York, USA` ↛ `York, UK`
- **`Santiago Bernabeu`** → `Santiago Bernabeu, Madrid, Spain` ↛ `Santiago, Chile`
- **`New Mexico`** → `New Mexico, USA` ↛ `Mexico`

[4] Formalization in Appendix H.

[5] Optimal value: 6 words

The approach starts with an n-gram iteration process, merging tokens forming natural locations into a singular *gram* (e.g., *"Mediterranean Sea"*, *"Lake Michigan"*). This reduces false positives from querying tokens separately. Identified natural elements are sent to Nominatim for geolocation. Subsequently, a coordinate parser extracts coordinate strings for reverse geocoding to derive corresponding OSM objects.

The n-gram iterator first processes NER-identified locations, focusing solely on the NER output. In its second pass, it incorporates both the whereEvent string and the NER locations. The algorithm iterates through results from the longest combined length to individual tokens, evaluating each for potential matches with previously disambiguated locations or earlier results. This dynamic function transforms both the n-gram iterator buffer and the temporary output as matches or affinities are found between queries.

Matches are defined as overlaps that prioritize specific, detailed locations over broader areas. Affinities, in contrast, are continuous ranges rather than binary. These include fuzzy matching scores, sequential anchoring, static anchoring, OSM importance, and internal importance. As matches and affinities evolve, they influence the ranking of Nominatim outputs and the trajectory of iterations. The process excludes n-grams corresponding to already disambiguated locations and removes contextual information used in disambiguation for a final output candidate.

4.4 Final cleaning and filtering score

The final step involves thorough output cleaning to eliminate redundancies and assign a filtering score for noise reduction. This includes sanity checks to remove stopwords, incomplete strings, or parsing inaccuracies. It verifies whether entries are supersets of others, removing redundant data.

The filtering score is designed to mitigate noise, particularly when dealing with highly ambiguous entities. For example, the location name *Zaragoza* appears in seven different countries. This scenario presents a dilemma: whether to prioritize recall by presenting all possible matches, or to focus on precision by selecting the most probable match based on the initial ranking. The filtering score balances this trade-off by assigning a value based on a confluence of factors, including the place class, place type from OpenStreetMap (OSM), and importance metrics. This approach enables effective noise filtration, particularly when deploying the geocoding solution in end-user applications, as demonstrated in the confidence and sensitivity section application.

4.5 Fine tuning

PS-Radar uses static variables as thresholds in heuristic and rule-based decisions. We employ a grid-search methodology to optimize performance, systematically exploring a predefined multidimensional parameter space and assessing algorithmic performance against precision, recall, and the SonarChallenge score. This exercise enhances the algorithm's generalization and robustness through optimal static parameter values.

In Figure 1, the impact of fine-tuning these variables on precision and SonarChallenge scores is illustrated.[6] The figure presents three pairwise iteration examples. The first graph demonstrates that stricter thresholds for fuzzy matching correlate with increased precision. The second graph plots the importance difference for a fuzzy match to be ranked in the temporary output on the x-axis, the importance difference allowed for a NER country location candidate compared to the top-ranked candidate for inclusion in the temporary output on the y-axis, and the SonarChallenge score on the z-axis. The third graph reveals an improvement in precision when the distance threshold for low administrative level reinforced candidates—those with overlaps from higher administrative levels—is tightened when in comparison to another higher-ranked rein-

[6]For full pairwise comparisons and metric analyses, refer to Appendix J.

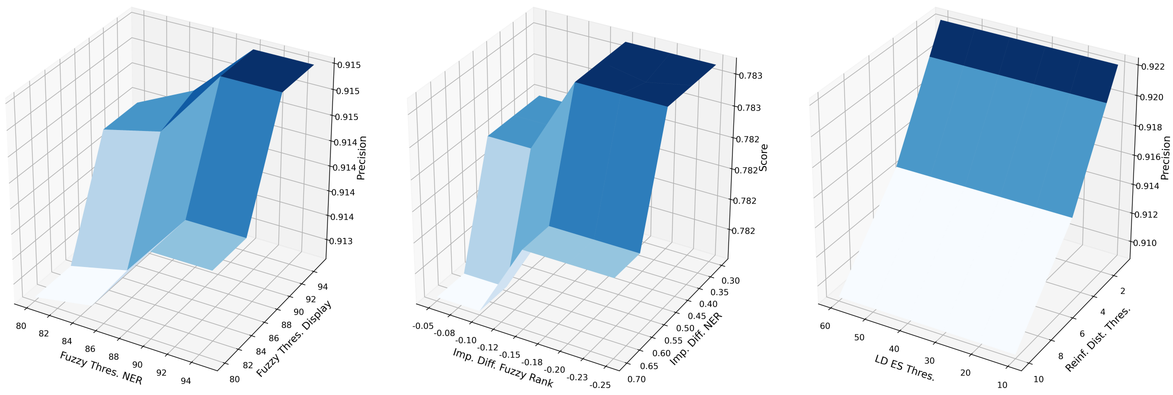

Figure 1: Example Pairwise Comparison: SonarChallenge score results

forced output. This implies that a candidate is disregarded if it lies beyond this threshold. Additionally, it becomes evident that the Levenshtein distance applied to ElasticSearch Geonames does not significantly impact our algorithm, given the fuzzy-search feature in ElasticSearch that is triggered if the search query yields no initial findings.

5 Results

We conduct a comparative analysis of our algorithm against two notable open-source solutions: Geoparsepy (Middleton et al., 2018) and Mordecai (Halterman, 2017). To ensure a fair comparison, we modify our SonarChallenge dataset to include only city locations, aligning with the precomputed dataset type in Geoparsepy. Mordecai's reliance on the Geonames format prevents direct comparison with the SonarChallenge's OpenStreetMap (OSM) input specifications. We evaluate also the performance of PS-Radar, Geoparsepy, and Mordecai using Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE), calculated based on the Euclidean distance between the actual and predicted locations.

Table 1: Performance Metrics (Part 1)

	Max. Score	Score (%)	TP/FP/Total
Geoparsepy	1343	709 (52.79)	251/93/354
PS-Radar	1343	**1086.5** (**80.90**)	**316**/**21**/354
Mordecai	NA	NA	NA/NA/NA

Our solution demonstrates superior performance in overlap completion as measured by the SonarChallenge[7]. PS-Radar excels in recalling true locations, achieving a rate of 89.3%, and in precision when predicting locations, with a rate of 93.8%. The algorithm maintains a reliable balance between recall and precision, evidenced by a 91.5% F1-Score.

Table 2: Performance Metrics (Part 2)

	Precision	Recall	MAE	RMSE
Geoparsepy	72.97%	70.90%	**83.08**	**391.98**
PS-Radar	**93.77%**	**89.27%**	97.85	572.92
Mordecai	NA	NA	598.25	2389.68

However, in terms of average distance error, Geoparsepy surpasses our solution. PS-Radar occasionally produces false positives at significantly distant locations, adversely affecting distance error metrics. Our integration with a full-planet OpenStreetMap (OSM) database allows our engine to retrieve a broader range of candidates, increasing the risk of distant false positives. In contrast, Geoparsepy's restriction to city-level output mitigates this risk. Consequently, while PS-Radar is highly precise, with a Type I error of only 6.2% compared to Geoparsepy's 27%, it deviates, on average, 14.8 km more from the true coordinates in instances of false positives than Geoparsepy.

5.1 Ablation tests

In our ablation study, we dissect the algorithm into distinct blocks to determine the

[7] For result metrics, we consistently utilize the highest-ranked entry (best-guess approach).

contribution of each component. Starting with version zero (v0), we incrementally add blocks, enabling the atomization of each component's impact.

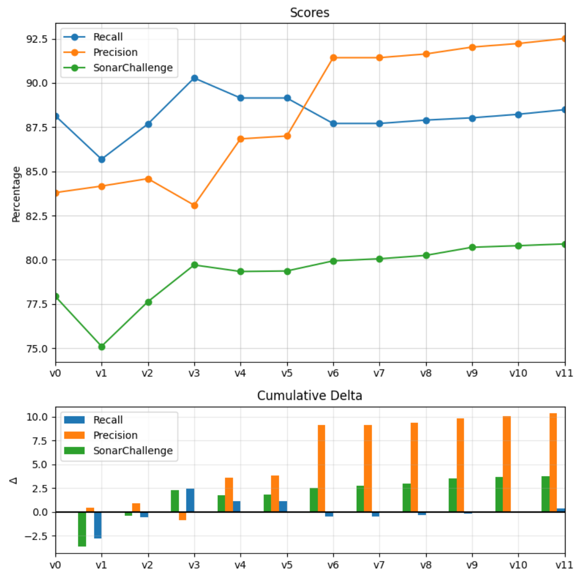

Figure 2: Results: Ablation Tests

The most significant improvements occur in versions v3, v4, and v6.

Version 3 (v3) incorporates an early stopping mechanism, enhancing recall by 2.97% compared to version 2 (v2), with a cumulative improvement of 2.42%. This mechanism directs the parsing of long whereEvent strings to rely solely on NER-identified locations, with an optimal cut-off length of 6 tokens, simplifying input to the parser.

Version 4 (v4) introduces a "distrust condition" to demote street-level entries coinciding with nucleus-level locations more than 10 km away from the nucleus' centroid. This targets cases where street names match city names (e.g., *"Oxford Street"*, *"Calle Madrid"*, or *"Rue de Bordeaux"*). This enhancement increases precision by 4.51% over v3, with a cumulative delta of 3.63%.

Version 6 (v6) adds a reinforcement process to the temporary output, strengthening entries with administrative affinity and dynamically reranking them. Entries without overlaps are removed if distant from the reinforced coordinate, significantly reducing false positives. This yields a 5.09% precision improvement over version 5 (v5) and a cumulative delta of 9.11% from v0.

In summary, the incorporation of refinement blocks into the core algorithm (v0) results in a recall gain of 0.39% and a notable precision improvement of 10.39%. Detailed descriptions and metrics of the ablation tests can be found in Appendix I.

5.2 Confidence and Sensitivity

The PS-Radar algorithm integrates a voting regressor (Wolpert, 1992) to assign a probability score to each output candidate independently and employs a rank-dependent sensitivity level.

This voting regressor includes a Gradient Boosting regressor, a Random Forest, and a Linear Regression. The input vector comprises variables such as the entry's OSM type, fuzzy matching score, location rank, importance, and correlations with other entities in the message. The outcomes are summarized in Table 3.

Table 3: Voting Regressor results

	Precision	Recall	F1-Score
False Output	86.83	82.24	84.47
True Output	83.92	87.11	85.49
Accuracy			85.65
Samples			167,750

A sensitivity level, ranging from 1 to 8, is introduced based on the importance score and OSM location types. Level 1 signifies the highest-ranked entry, while level 8 represents the lowest.

The sensitivity level categorizes output entries, indicating the preferred options for a specific whereEvent and facilitating output reduction. For instance, with a whereEvent like *"London"* without additional context, all potential matches are outputted. A more conservative sensitivity level filters these locations to higher-ranked ones. Opting for level 1 would focus on the highest probability match, in this case *"London, UK"*. This represents a sliding recall/precision tradeoff, as illustrated in Figure 3.

On the other hand, the confidence score

Figure 3: Sensitivity slider: $8 \to 4 \to 1$

within the PS-Radar algorithm is designed to evaluate the likelihood of each entry being a true positive. This metric assesses entries independently, irrespective of their order in the output. To illustrate this, consider the previous scenario involving the whereEvent of *"London"*. We analyze the first entry in the output (*London, UK* for both) but with different contexts in the whereEvent:

whereEvent	**London**	**London, UK**
Sensitivity Level	1	1
Confidence Score	75.84	98.68

From this analysis, it is evident that the specificity of the *"UK"* context in the second option significantly enhances the confidence level, logically suggesting a higher likelihood of it being a true positive.

In summary, PS-Radar uses a standardized procedure to assign sensitivity levels, enabling consistent visualization and filtration of output locations for specific precision/recall balances or map pollution levels. Additionally, a confidence score gauges the probability of each entry being a true positive.

6 Conclusion

The PS-Radar geoparser is a real-time processing tool designed to extract locations from messages by employing Named Entity Recognition (NER) and Question-Answering (QA) techniques. It parses text strings, geocodes potential matches using OpenStreetMap (OSM) Nominatim, and disambiguates the final location based on contextual information. Our analysis demonstrates how PS-Radar surpasses major open-source alternatives, while being able to process large datasets in real-time through the AnonymousSubmission platform. The service provides various information keys for each result, including a sensitivity level that allows end-users to balance between noise/recall and precision enhancement.

PS-Radar boasts a broad spectrum of applications. It enables real-time visualization of locations mentioned in message streams, which can be pivotal in tracking the geographical progression of events like natural disasters. Geocoding messages enhances filtering capabilities for monitoring specific cases. The implementation of geofencing in social media contexts allows for the exclusion of irrelevant messages, thereby reducing noise and increasing efficiency for both intelligence services and end-users. This is especially crucial when tracking ambiguous locations such as *London* or *Main Street*, where results can span a wide geographical range due to their common occurrence. Effective toponym disambiguation ensures focus on relevant hits within the area of interest. Additionally, PS-Radar augments alert systems by incorporating geographical dimensions that are critical for report triggering.

7 Limitations

The algorithm remains open to modifications and could derive significant benefits from the integration of additional information and sources. One pertinent factor is the incorporation of enhanced contextual data. When an individual posts content on social media, they often presuppose that their immediate network is aware of their background, leading to the omission of detailed and explicit addresses. This forces algorithmic approaches to take assumptions on inferences and educated guesses. A substantial enhancement would entail analyzing historical data to ascertain the user's primary geographical sphere of activity, which could then be utilized during the disambiguation stage. This approach could ameliorate instances where a user references a street name that could correspond to multiple cities. If it were established that the user's main sphere of influence is in a specific city, this could significantly improve disambiguation efforts.

Furthermore, another avenue for improvement is the utilization of external Knowledge Graphs to exploit additional identifiers, such as the locations of festivals, concerts, or sports events, the whereabouts of notable individuals, or the names and affected areas of natural disasters.

A limitation of the PS-Radar algorithm arises when dealing with the disambiguation of numerous entities in a message. We adapted the model to also work with only NER locations. removing the dependency of a whereEvent extraction. If the model is required to analyze all NER locations and there are many, the strong disambiguation logic aimed at precision can become a bottleneck. The algorithm attempts to form combinations of these locations to find overlaps or matching OSM entries, which can be computationally intensive and may affect quality performance in such scenarios.

The final area of development for the PS-Radar concerns its capability to integrate with geocoding services beyond OpenStreetMap (OSM). The inclusion of alternative providers—primarily commercial, paid services—could enhance and strengthen the algorithm's output.

References

Geoparsing - gis wiki | the gis encyclopedia.

Twitter usage statistics - internet live stats.

Marco Avvenuti, Stefano Cresci, Leonardo Nizzoli, and Maurizio Tesconi. 2018. Gsp (geo-semantic-parsing): geoparsing and geotagging with machine learning on top of linked data. In *European Semantic Web Conference*, pages 17–32. Springer.

Judith Gelernter and Shilpa Balaji. 2013. An algorithm for local geoparsing of microtext. *GeoInformatica*, 17:635–667.

Andrew Halterman. 2017. Mordecai: Full text geoparsing and event geocoding. *The Journal of Open Source Software*, 2(9).

Andrew Halterman. 2019. Geolocating political events in text. *arXiv preprint arXiv:1905.12713*.

Morteza Karimzadeh, Scott Pezanowski, Alan M MacEachren, and Jan O Wallgrün. 2019. Geotxt: A scalable geoparsing system for unstructured text geolocation. *Transactions in GIS*, 23(1):118–136.

Egoitz Laparra and Steven Bethard. 2020. A dataset and evaluation framework for complex geographical description parsing. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 936–948.

Kalev Leetaru, Shaowen Wang, Guofeng Cao, Anand Padmanabhan, and Eric Shook. 2013. Mapping the global twitter heartbeat: The geography of twitter. *First Monday*.

Michael D Lieberman, Hanan Samet, and Jagan Sankaranayananan. 2010. Geotagging: Using proximity, sibling, and prominence clues to understand comma groups. In *proceedings of the 6th workshop on geographic information retrieval*, pages 1–8.

Christopher Manning and Hinrich Schutze. 1999. *Foundations of statistical natural language processing*. MIT press.

Stuart E Middleton, Giorgos Kordopatis-Zilos, Symeon Papadopoulos, and Yiannis Kompatsiaris. 2018. Location extraction from social media: Geoparsing, location disambiguation, and geotagging. *ACM Transactions on Information Systems (TOIS)*, 36(4):1–27.

Stuart E Middleton, Andrea Zielinski, Öcal Necmioğlu, and Martin Hammitzsch. 2014. Spatio-temporal decision support system for natural crisis management with tweetcomp1. In *Decision Support Systems III-Impact of Decision Support Systems for Global Environments: Euro Working Group Workshops, EWG-DSS 2013, Thessaloniki, Greece, May 29-31, 2013, and Rome, Italy, July 1-4, 2013, Revised Selected and Extended Papers*, pages 11–21. Springer.

Leonardo Nizzoli, Marco Avvenuti, Maurizio Tesconi, and Stefano Cresci. 2020. Geo-semantic-parsing: Ai-powered geoparsing by traversing semantic knowledge graphs. *Decision Support Systems*, 136:113346.

Dikshant Shahi. 2016. *Apache solr*. Springer.

David A Smith and Gregory Crane. 2001. Disambiguating geographic names in a historical digital library. In *International Conference on Theory and Practice of Digital Libraries*, pages 127–136. Springer.

Jan Oliver Wallgrün, Morteza Karimzadeh, Alan M MacEachren, and Scott Pezanowski. 2018. Geocorpora: building a corpus to test and train microblog geoparsers. *International Journal of Geographical Information Science*, 32(1):1–29.

Xingguang Wang, Yi Zhang, Min Chen, Xing Lin, Hao Yu, and Yu Liu. 2010. An evidence-based approach for toponym disambiguation. In *2010 18th International Conference on Geoinformatics*, pages 1–7. IEEE.

David H Wolpert. 1992. Stacked generalization. *Neural networks*, 5(2):241–259.

A SonarChallenge data structure

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "type": "object",
  "properties": {
    "Message": { "type": "string" },
    "whereEvent": { "type": "string" },
    "NER": {
      "type": "array",
      "items": [
        { "type": "string" },
        {
          "type": "array",
          "items": [
            { "type": "integer" },
            { "type": "integer" }
          ]
        },
        { "type": "string" },
        { "type": "number" }
      ]
    },
    "Language": { "type": "string" }
  }
}
```

B SonarChallenge sample


```
{
   "Message":"A #CobbCounty man has been named as the
        suspect in Tuesday's fatal stabbing of a #
        Pentagon officer in #Arlington, #Virginia, the #
        FBI confirmed to the MDJ.",
   "whereEvent":"in # Arlington , # Virginia",
   "NER":[
      ["A", [0, 1], "O", 0.9999631618908261],
      ["#CobbCounty", [2, 13], "O", 0.5230842600430057],
      ["man", [14, 17], "O", 0.9999011493900142],
      ["has", [18, 21], "O", 0.9999778098743163],
      ["been", [22, 26], "O", 0.999987045345547],
      ["named", [27, 32], "O", 0.9999845801362041],
      ["as", [33, 35], "O", 0.9999840281632058],
      ["the", [36, 39], "O", 0.9999846365657179],
      ["suspect", [40, 47], "O", 0.9999774981836242],
      ["in", [48, 50], "O", 0.9999894255242104],
      ["Tuesday's", [51, 60], "B-TIM", 0.9987443381939265
           ],
      ["fatal", [61, 66], "O", 0.9999532689098133],
      ["stabbing", [67, 75], "O", 0.9999621381247745],
      ["of", [76, 78], "O", 0.9999887037609607],
      ["a", [79, 80], "O", 0.999963093788084],
      ["#Pentagon", [81, 90], "O", 0.5760812053361055],
      ["officer", [91, 98], "O", 0.9997355455505741],
      ["in", [99, 101], "O", 0.9998647856427583],
      ["#Arlington", [102, 112], "B-LOC", 0.9656087453283
           584],
      [",", [112, 113], "O", 0.9224836796891772],
      ["#Virginia", [114, 123], "B-LOC", 0.78543622657498
           69],
      [",", [123, 124], "O", 0.9996642582381823],
      ["the", [125, 128], "O", 0.9996824898159645],
      ["#FBI", [129, 133], "B-ORG", 0.9072086458363313],
      ["confirmed", [134, 143], "O", 0.9999827591593275],
      ["to", [144, 146], "O", 0.9999885614584061],
      ["the", [147, 150], "O", 0.999975305586951 8],
      ["MDJ", [151, 154], "B-ORG", 0.861251877685755],
      [".", [154, 155], "O", 0.9999499362226958]],
   "Language":"EN"
}
```


C N-gram iterator

Algorithm 1 n_gram_iterator

Require: sentence: the input sentence

1: words ← split(sentence)
2: $n \leftarrow \min(4, \text{len(words)})$ ▷ Starting n-gram size
3: **while** $n > 0$ **do**
4: **for** $i \leftarrow 0$ **to** $\text{len(words)} - n$ **do**
5: **yield** $\text{join(words}[i : i+n])$ ▷ Generate an n-gram
6: **end for**
7: $n \leftarrow n - 1$ ▷ Decrement n for the next iteration
8: **end while**

D Fuzzy matching

Algorithm 2 fuzzy_max

Require: string: The target string for which you want to find the best fuzzy match.
Require: osm_names: The full naming list from the OpenStreetMap (OSM) candidate.

max_len $\leftarrow$ length(osm_names) $\triangleright$ Calculating the maximun length of the split OSM full address string and OSM naming lists.
fuzzy_rates $\leftarrow$ empty list
for L $\leftarrow 0$ **to** max_len **do**
  for all subset $\in$ combinations(osm_names, L) **do**
    fuzzy_score $\leftarrow$ int: fuzz.ratio(string, subset)
    append(fuzzy_rates, fuzzy_score)
  end for
end for
return max(fuzzy_rates) $\triangleright$ Returning the maximum fuzzy matching score achieved.

Algorithm 3 fuzz_ratio

Require: str1: the first string, str2: the second string

$T \leftarrow$ len(str1) + len(str2) $\triangleright$ Total number of characters in both strings
$M \leftarrow$ computeMatches(str1, str2) $\triangleright$ Number of matches in the two strings
$fuzzy\_score \leftarrow$ int(round((2.0 * M / T) * 100)) $\triangleright$ Compute fuzzy score
return $fuzzy\_score$

E Sequential & Static anchoring

Sequential anchoring delivers an anchor which is an average of the last one hundred previous locations disambiguated and it is thought for a real-time or sequential application to leverage location correlation through time in a message flow from a related topic. Static anchors can be set if we know the origin of the source, e.g. we have a message flow from a local news provider from Trinity County, and it helps skew preference towards of ambiguous locations that are closer to those coordinates.

Algorithm 4 skew_importance_anchor

Require: results: a list of results with 'fuzzy', 'importance' and OSM tags
Require: anchor_value: the anchor value in coordinates for distance calculation
Require: results_sorted: a sorted list of results

distance2avg $\leftarrow$ empty list $\triangleright$ Initial list for storing distance to average values
for each $output$ in results **do**
  distance $\leftarrow$ distance_calculus(anchor_value, $output['latLon']$)
  distance2avg.append(distance)
end for
if length(distance2avg) > 0 **then**
  distance2avg_norm_rank $\leftarrow$ normalize(1/normalize(distance2avg))
  for each $output$, $distance\_rank$ in zip(results, distance2avg_norm_rank) **do**
    $output['importance'] \leftarrow (5/6) * output['importance'] + (1/6) * distance\_rank$
  end for
end if

F OSM importance

The primary determinant of importance in OpenStreetMap (OSM) is predominantly derived from the count of Wikipedia links associated with a particular object. In cases where no corresponding Wikipedia article exists, the fundamental importance score is established based on the object's hierarchical rank (e.g., country, county, city). Additionally, there are nuanced adjustments to the Wikipedia-derived importance metric, notably involving the re-ranking process based on the precision of the match with the query. Specifically, the re-ranking considers the exactness of the match between the query and the display name, encompassing the address details of the result. The significance of a result is positively correlated with the extent to which the words from the query appear verbatim in the display name. Hence, the mentioned significance of a correct parsing and cleaning process of the query string.

G Internal importance

The internal importance metric is the modified original OSM importance of the output candidate depending on the matches with other entities, the distance from its centroid to those of the other candidates and the anchoring distance values.

H Dynamic Ranking Algorithm Formalization

Detailed formalization of the dynamic ranking algorithm used in the geocoding process. The algorithm's core objective is to accurately identify and rank geographical locations based on the whereEvent string input, leveraging contextual Named Entity Recognition (NER) locations.

H.1 Identification of Location Affinity

The algorithm initiates by mapping each location l to a set of potential location matches from a predefined set of known locations L, which includes both NER-identified locations and scanned countries. The mapping function $f(L)$ identifies locations with a significant affinity to each l element of L:

$$f(L) = \{l \in L \mid \text{LocationAffinity}(l, L)\}$$

This creates a temporary output for NER locations to be leveraged in the dynamic ranking phase. LocationAffinity is a rule-based gate with a compound input format.

H.2 Hypothesizing Geographical Location with Affinity and Overlap

Considering the whereEvent as W, the algorithm hypothesizes the most likely geographical location h, focusing on identifying the lowest possible administrative level of overlap while simultaneously seeking the highest affinity. This dual objective is encapsulated in the determination of both the administrative level $A(l)$ and a measure of affinity $M(L, W)$ for each potential location l in L:

$$\text{LowestAffinityLevelAndOverlap}(h) = \min_{l \in L} (A(h, l), -M(h, l)) \tag{1}$$

Here, $A(h, l)$ represents the administrative level overlap achieved of location h with leverage context l, and $M(h, l)$ quantifies the affinity of l to L, with the goal of maximizing affinity while minimizing the administrative level match.

H.3 Internal Importance Score

An internal importance score S is then assigned to each location based on the degree of overlap and affinity with the hypothesized location. The score $S(l, h)$ reflects the relevance and specificity of location l in relation to h:

$$S(l, h) = \text{ScoreFunction}(l, h)$$

H.4 Ranking Against Other OSM Objects

Finally, the algorithm ranks the identified locations against other OSM objects o retrieved in the process. The ranking is based on the internal importance scores, generating a sorted list of locations:

$$\text{RankedList} = \text{sort}\left(\{(o, S(o, h)) \mid o \in O\}\right)$$

H.5 Dynamic Iteration and N-Gram Processing

The dynamic ranking mechanism incorporates a max n-gram logic to enhance accuracy by favoring matches with the longest n-gram possible.

Let n_i denote the length of the n-gram being considered during iteration i. Let h be the hypothesis candidate. Given an overlap match, the algorithm prioritizes n-grams according to their length. Specifically, for any two n-grams n_1 and n_2, if $n_1 < n_2$, then the priority enhancement E satisfies:

$$\forall n_1, n_2 \in \mathbb{N}, n_1 < n_2 \Rightarrow E(n_1) < E(n_2)$$

This prioritization process can be expressed as:

$$\max_i(E(n_i))$$

where $\max_i$ represents the selection of the n-gram with the maximum length at each iteration i.

Additionally, the algorithm introduces a buffer B to store potential contextual references. This buffer helps in refining the hypothesis candidate h by discarding iterations corresponding to already accepted hypotheses in the whereEvent W. Thus, the dynamic ranking mechanism iterates through n-grams, leveraging the buffer and discard mechanism to clean iterations and refine the hypothesis candidate h based on overlap matches and contextual references.

I Ablation tests

v1

Inclusion of NER person entities to avoid false positives when parsing through people's names.

v2

Inclusion of longer n-grams in the algorithm.

v3

Early stopping mechanism when parsing long strings.

v4

Distrust check for street level entries if matched nucleus found and this is further than 10 km.

v5

Remove road and highway formatted output if no reference or reinforcement present. This avoids weak candidates for locations such as A-62, CV-203, N89, etc.

v6

Removal of overlapping entities of higher level from the output. Surviving candidates are reinforced with the deleted ones, if applicable.

v7

Country control in order to tackle ambiguous references as *London, Canada* and *London, UK* in the same message.

v8

Nominatim engine update to more recent version (start of 2023).

v9

Fine-tuning of static parameters. These parameters are location distance, fuzzy matching, anchor distances or importance ranking thresholds.

v10

Enrichment of alternative naming from OSM.

v11

Full-distance matching. The overlap match rule continues to check for better and more precise matches through NER locations and the whereEvent. The previous version stopped iterating when it found the first match.

Table 4: Ablation Tests: Analysis of Algorithm Performance

Algo. Version	Recall	Recall Δ	Recall Cum. Δ	Precision	Precision Δ	Precision Cum. Δ	Sonar Challenge	Sonar Challenge Δ	Sonar Challenge Cum. Δ
v0	88.15%			83.80%			77.94%		
v1	85.69%	-2.79%	-2.79%	84.17%	0.44%	0.44%	75.11%	-3.63%	-3.63%
v2	87.68%	2.32%	-0.53%	84.59%	0.50%	0.94%	77.63%	3.36%	-0.39%
v3	90.28%	2.97%	2.42%	83.09%	-1.77%	-0.85%	79.71%	2.68%	2.28%
v4	89.15%	-1.25%	1.13%	86.84%	4.51%	3.63%	79.34%	-0.46%	1.80%
v5	89.15%	0.00%	1.13%	87.00%	0.18%	3.82%	79.37%	0.04%	1.84%
v6	87.71%	-1.62%	-0.50%	91.43%	5.09%	9.11%	79.94%	0.72%	2.57%
v7	87.71%	0.00%	-0.50%	91.43%	0.00%	9.11%	80.06%	0.15%	2.73%
v8	87.90%	0.22%	-0.28%	91.64%	0.23%	9.36%	80.25%	0.24%	2.97%
v9	88.03%	0.15%	-0.14%	92.03%	0.43%	9.82%	80.71%	0.57%	3.56%
v10	88.23%	0.23%	0.09%	92.23%	0.22%	10.06%	80.80%	0.11%	3.68%
v11	88.49%	0.29%	0.39%	92.51%	0.30%	10.39%	80.90%	0.12%	3.80%

J Fine-tuning results

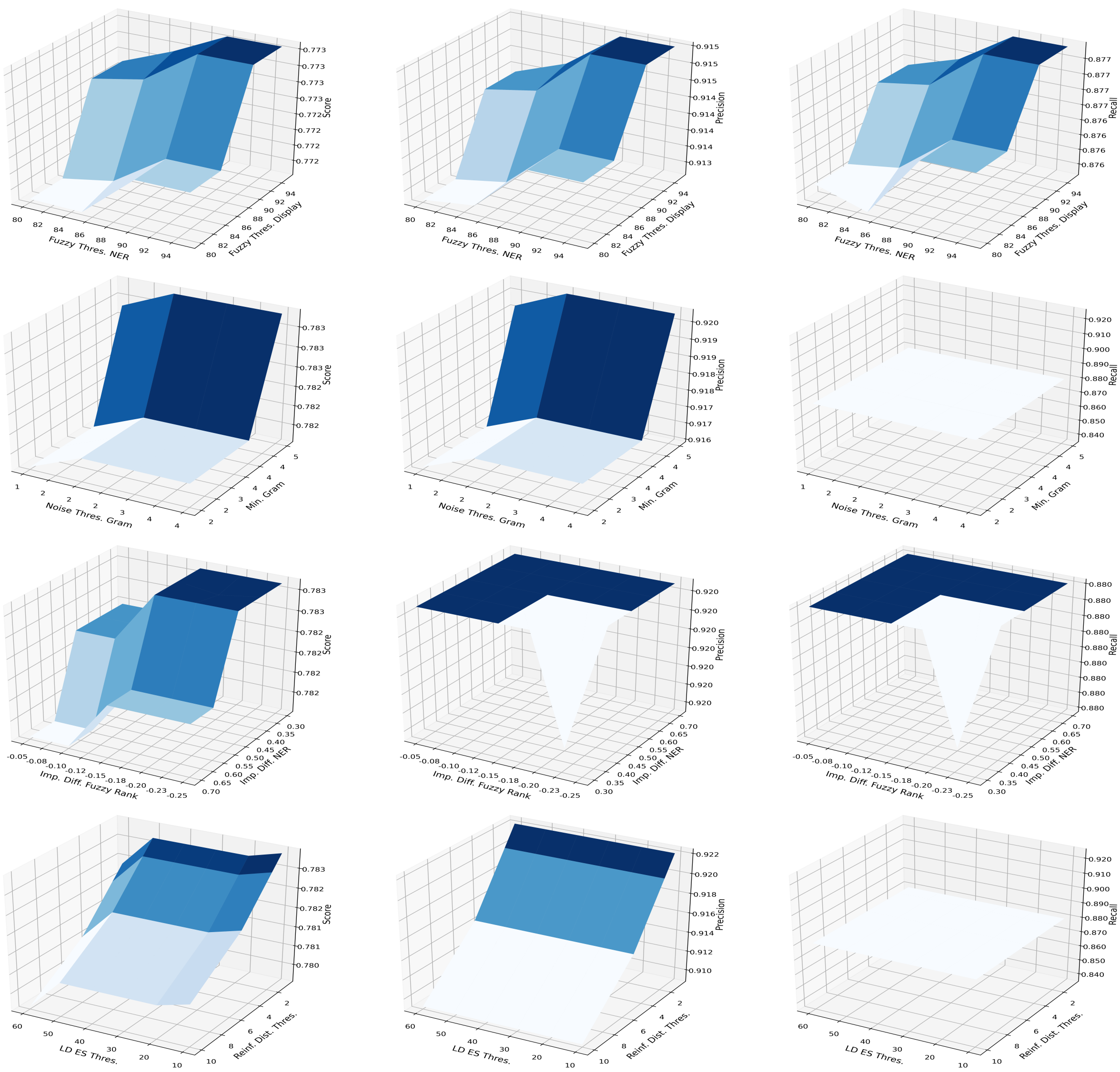

Figure 4: Pairwise comparison results