SemaSK: Answering Semantics-aware Spatial Keyword Queries with Large Language Models

Zesong Zhang
The University of Melbourne
Melbourne, Australia
zesongz@student.unimelb.edu.au

Jianzhong Qi*
The University of Melbourne
Melbourne, Australia
jianzhong.qi@unimelb.edu.au

Xin Cao
The University of New South Wales
Sydney, Australia
xin.cao@unsw.edu.au

Christian S. Jensen
Aalborg University
Aalborg, Denmark
csj@cs.aau.dk

ABSTRACT

Geo-textual objects, i.e., objects with both spatial and textual attributes, such as points-of-interest or web documents with location tags, are prevalent and fuel a range of location-based services. Existing spatial keyword querying methods that target such data have focused primarily on efficiency and often involve proposals for index structures for efficient query processing. In these studies, due to challenges in measuring the semantic relevance of textual data, query constraints on the textual attributes are largely treated as a keyword matching process, ignoring richer query and data semantics. To advance the semantic aspects, we propose a system named SemaSK that exploits the semantic capabilities of large language models to retrieve geo-textual objects that are more semantically relevant to a query. Experimental results on a real dataset offer evidence of the effectiveness of the system, and a system demonstration is presented in this paper.

1 INTRODUCTION

Keyword-based retrieval by mobile users often has local intent, in that it concerns content that is near the query user. As such, the geographical location of content has gained in importance with the prevalence of smartphones. This has motivated studies on new retrieval methods for data with both spatial and textual attributes, which is often referred to as *geo-textual data*.

Existing studies focused primarily on efficient retrieval of geo-textual data. They formulate *spatial keyword queries* [8, 10, 16, 17, 19] and propose index structures such as the IR-trees [12] for fast query processing. Spatial keyword queries retrieve data objects that satisfy both spatial (e.g., within a given region or near a given location) and textual constraints of the queries. Due to challenges in measuring the semantic relevance of textual data to query keywords, query constraints on the textual attributes are largely treated as query keywords to be matched by the textual attributes of the data objects.

Figure 1 shows an example with Google Maps, where a user searches for "café" in Melbourne CBD. All returned results contain the keyword "café", while cafés without such keywords, e.g., "Industry Beans" (a popular local café) or "Starbucks", are missing.

To improve on the accuracy of keyword-based selection of data objects, we propose a system named SemaSK that exploits the semantic capabilities of *large language models* (LLMs) to assess the semantic relevance between a spatial keyword query and a geo-textual object.

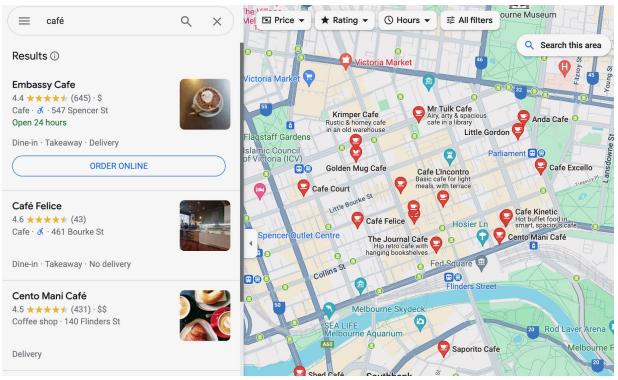

Figure 1: Querying "café" in Melbourne CBD.

Since LLMs lack universal knowledge of all geo-textual objects available for querying, SemaSK adopts a *retrieval-augmented generation* (RAG) [11]-based, filtering-and-refinement query procedure. When a query is issued, SemaSK first uses spatial constraints and embeddings of the object attributes and query keywords to quickly retrieve relevant geo-textual objects from a database. Then, an LLM is prompted to refine and re-rank the retrieved objects based on their semantic relevance to the query, the result of which is returned to the user as the final query result.

Overall, we make the following contributions: (1) We propose to answer spatial keyword queries with a semantics-aware procedure enabled by LLMs. (2) We prepare a dataset using Yelp [1] data with rich textual attributes and use it to show the effectiveness of semantics-aware spatial keyword query processing with LLMs. While this dataset cannot be redistributed due to Yelp's license requirements, we present detailed steps to make it easy for future studies to construct similar datasets. (3) We implement and demonstrate the SemaSK system that showcases the new LLM-based query procedure. Source code of the system (including the complete LLM prompts) is available on GitHub.[1]

2 RELATED WORK

Spatial keyword queries typically return geo-textual object(s) that satisfy given spatial constraints (e.g., within a query range or near a query point) and match given query keywords [9]. Existing studies on such queries generally focus on efficiency. Many index structures have been proposed, the general idea of which is to exploit the pruning capability of spatial and keyword attributes to reduce the search space. The IR-tree [12], for example, adds an inverted index to each node of an R-tree, to index all keywords appearing in the sub-tree of the node. Latest studies incorporate machine learning techniques to optimize the index structures,

*Corresponding author

[1] https://github.com/Bigtable123/SemaSK

such as the *WISK* index [17] which learns a hierarchical grouping of the geo-textual objects based on a given query workload to optimize *spatial keyword range query* processing. The TF-IDF measure that is often used in these studies to measure textual relevance [8] ignores the broader semantics of the keywords.

Semantic relevance is thus underexplored. Several studies [15, 16, 18] consider *spatial keyword query with semantics*, where semantic relevance is measured with the *Latent Dirichlet Allocation* (LDA) model. LDA represents documents (i.e., the keywords of a query or data object) as random mixtures of latent topics, each described by a distribution of words. The semantic relevance between documents is then defined as the similarity of their corresponding distributions of words. LDA is based on the "bag-of-words" assumption and ignores the ordering of words in a document. It is less effective than LLMs in capturing text semantics. Our system addresses this issue.

LLMs are already employed in the geospatial domain. Existing studies generally adopt one of two directions: (1) To study the geospatial understanding capabilities of LLMs, e.g., whether LLMs can make predictions (such as population density) corresponding to different locations [14]; or (2) To build applications exploiting such capabilities, e.g., using LLMs to recommend routes when integrated with map APIs [20]. Our system differs from these studies in that it exploits LLMs for their textual understanding capabilities.

3 THE SEMASK SYSTEM

Consider a set of n geo-textual objects $O = \{o_1, o_2, \ldots, o_n\}$. Each object o_i consists of a set of attributes represented as key-value pairs, one of which is the location attribute $o_i.l$ (i.e., a pair of geo-coordinates). All attribute keys are textual, while the attribute values may be numerical, categorical, or textual, with at least one being textual (for keyword-based querying). We denote the set of attributes excluding $o_i.l$ by $o_i.A$.

A user query q is represented by a range $q.r$, which defines a region (e.g., a rectangle), and a textual constraint $q.T = \{t_1, t_2, \ldots, t_m\}$ consisting of m tokens (e.g., a sentence or a set of search keywords). An example textual constraint is: "*I am looking for a bar to watch football that also serves delicious chicken. Do you have any recommendations?*" Our goal is to return all geo-textual objects from O that are within $q.r$ and relevant to the textual query constraint $q.T$.

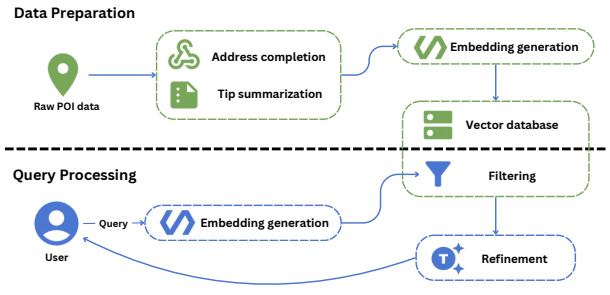

Figure 2: Overview of the SemaSK System.

The SemaSK system has two main modules: a *data preparation* module and a *query processing* module, as shown in Figure 2. The data preparation module pre-processes the dataset O and prepares it for query processing. It computes summaries to shorten long textual attribute values, and it converts the non-location attributes $o_i.A$ of each object o_i into an embedding $o_i.\mathbf{a}$, to enable a lightweight filtering step during query processing. The query processing module answers queries with an RAG-based, filtering-and-refinement procedure. Given a query q, its embedding $q.\mathbf{t}$ is computed from $q.T$. The top-k most similar objects are fetched from O based on embedding similarity (to limit the LLM costs of the refinement step). These objects are refined and re-ranked by an LLM to produce the final query answer. We detail the two modules in the subsections below.

3.1 Data Preparation

We exploit a *Yelp* dataset [1] with rich textual POI descriptions, i.e., "tips", which are brief reviews given by users. The dataset also comes with users' longer reviews on the POIs, which are quite noisy and may contain information irrelevant to the POIs. We ignore such data. Table 1 shows a sample record. The raw dataset contains 81,500 POIs. In our experiments, we use POIs from the five cities with the most POIs, which together have 19,795 POIs. The POIs have an average of 11 tips (147 tokens together). Note that popular POI datasets such as OpenStreetMap are unsuitable as they focus on POI locations and have only a few category keywords for each POI.

We prepare the geo-textual dataset for query processing with the following steps.

Address completion. The dataset has incomplete POI addresses. We employ reverse geocoding [5] to obtain city, county, suburb, and neighborhood information based on coordinates.

Table 1: A Sample Record from the Yelp Dataset

Attribute	Value
business_id	oaboaRBUgGjbo2kfUIKDLQ
name	Mike's Ice Cream
address	129 2nd Ave N
city	Nashville
state	TN
latitude	36.162649
longitude	-86.775973
stars	1.5
tip_count	10
is_open	1
categories	Ice Cream & Frozen Yogurt, Fast Food, …
hours	'Monday': '0:0-0:0', 'Tuesday': '6:0-21:0', …
tips	"Amazing ice cream! So creamy" , …

Tip summarization. The tips vary in length. We prompt an LLM, *GPT-3.5 Turbo* [2] (for its lower costs), to summarize the tips of each POI as follows:

You are a master of summarizing reviews. Now I have some reviews, they are in the form of lists in Python and split with commas. I would like you to help me make a summary. Here are some examples:

list:['Love Sonic but orders are constantly wrong', …, 'Foods always been good. Shakes r delicious!']

Summary: The feedback highlights a mix of experiences at Sonic. While there is love for the brand and appreciation for the quality of food and delicious shakes, there is also frustration over frequent inaccuracies in order fulfillment.

[A second example]

Now it is your turn: [tips to summarize]

On average, each generated summary has 55 tokens. We verify the quality of the summaries by manually examining 100 of them. We find that the summaries are of high quality in general and include the key information from the raw tips.

Embedding generation. Prompting an LLM to examine every POI record against a query online is expensive both in terms of time and LLM API call (monetary or energy) costs. For more efficient POI pruning, we pre-compute POI embeddings, which can be compared with the query embeddings online without any LLM API calls.

We adopt the off-the-shelf model, *text-embedding-3-small* [6], to compute the embeddings, taking as input the POI name, address, categories, hours, and tip summary. The embeddings (1,536 dimensions as defined by the model) are stored in the *Qdrant* [7] vector database system to support online query processing.

3.2 Query Processing

We process queries with a filtering-and-refinement procedure.

Filtering. When a query q arrives, we first filter the POIs by the given query range $q.r$. Then, we convert the query text into an embedding using the *text-embedding-3-small* model. With the embeddings of the query and the POIs in the query range, we run an approximate k nearest neighbor (kNN) query using the built-in HNSW algorithm [13] of *Qdrant*. This step can efficiently identify the top-k query answer POIs without LLM API calls.

Refinement. Next, we prompt an LLM (we use *GPT-4o* [3]) to re-rank the top-k POIs, exploiting LLM's capabilities to understand the semantics of both the query and the POI attributes. This extra refinement step is because the embeddings generated by a smaller model may not be as accurate in reflecting the semantics. We use the following prompt:

You are an assistant for location information sorting tasks. Below is the location information retrieved from the database, which will be given to you in JSON format. You are asked to filter and sort this information based on the question asked. You first need to determine whether the information is relevant to the question, and then sort all the relevant information. The ones that best match the question and help answer it have the highest priority. The format of your output must be a Python dictionary, where the key is the name of the location and the value is the reason why you chose this location and ranked it there. The location with the highest priority is placed higher, i.e., index is 0. Please note that there could be more than one result in the dictionary. If the information about a location could only partially match the question asked, you could also put it in the dictionary, but specify the advantages and disadvantages of this place in the value of the dictionary. If you could not complete the task or do not know the answer, just return the empty dictionary and don't refer to any additional knowledge.

Information: [Raw POI attributes]

Query: [User query]

4 EXPERIMENTAL STUDY

Our experiments show the effectiveness of SemaSK, using a laptop computer with the Apple M2 CPU and 8 GB memory.

Dataset: We use the processed Yelp dataset as described above for the experimental study. We use POIs in five cities in the USA to form five test sets: Indianapolis (**IN**, 4,235), Nashville (**NS**, 3,716), Philadelphia (**PH**, 7,592), Santa Barbara (**SB**, 1,790), and Saint Louis (**SL**, 2,462), where the numbers in parentheses are the numbers of POIs in each city.

As there are no queries that come with the dataset, we construct test queries as follows. We start by randomly selecting a point in each city. The query range is formed by a 5 km × 5 km region (to ensure enough POIs in the region) centered at the point (which can be easily extended to allow users to define their own query range, e.g., to query within their current map view). Within this range, a POI is randomly chosen. We input the attributes of this POI into an LLM (*OpenAI o1-mini* [4], for better query quality) and instruct the LLM to generate a query targeting the POI, following two manually crafted examples with the prompt below:

You are an expert in spatial keyword searching and I am now trying to perform spatial keyword searching using a large language model. In order to get a test set, I need you to help me write query questions based on the information I provide. In particular, I am asking to think of some questions that are difficult to answer with simple keyword matching, but are easier with the semantic capabilities of large language models, such as "Find Japanese restaurants in Center City that offer a variety of sushi options", where "Japanese restaurants" and "sushi" can be easily handled by keyword matching, while "a variety of options" may require semantic understanding. Also, please don't mention any location information in the query!

Information: Pep Boys is located at Lafayette Road and primarily serves the category of Automotive, Tires, Oil Change Stations, Auto Parts & Supplies, Auto Repair. It is open for business at these hours: ['Monday': '8:0-19:0', 'Tuesday': '8:0-19:0', 'Wednesday': '8:0-19:0', 'Thursday': '8:0-19:0', 'Friday': '8:0-19:0', 'Saturday': '8:0-19:0', 'Sunday': '9:0-17:0']. Customers often highlight: 'The reviews consistently praise the staff for being friendly, knowledgeable, and helpful, creating a positive and welcoming atmosphere for customers.'

Question: My car needs repair. Which service center is the most reliable?

[A second example]

Now it is your turn.

Information: [A POI input]

Question:

We generate 100 queries for each city, which are manual reviewed and adjusted to filter queries that can be easily answered by keyword matching. Afterwards, we manually inspect the corresponding query range to determine the answer set (there may be other POIs besides the target POI that also satisfy the generated query). In the end, 30 queries are harvested and used for testing on each city.

Competitors: As the test sets are relatively small, query efficiency is not an issue. We focus on the effectiveness and compare with two baseline algorithms: **LDA** and **TF-IDF**, which use the LDA model (following a previous study [16]) and TF-IDF vector similarity to assess text relevance and subsequently rank the POIs in the query range.

We also compare with two system variants: **SemaSK-EM** forgoes the refinement step of SemaSK (i.e., it queries POIs by the embeddings). **SemaSK-O1** uses *OpenAI o1-mini* instead of *GPT-4o* for query result refinement.

We assess the performance of the algorithms using **F1@k** which is the F1 score of the top-k POIs returned by each algorithm, averaged across all test queries of each city.

Results. Due to space limit, we only present results for $k = 10$ in Table 2. Similar result patterns are observed when k is varied (e.g., for $k = 25$) and hence are omitted.

We observe that our solutions SemaSK-O1 and SemaSK outperform the two baselines across all test sets, with the average

Table 2: Performance Results in F1@k (best results are in boldface; "Avg." means "Average"; numbers in parentheses represent performance gains over best baseline results)

City	LDA	TF-IDF	SemaSK-EM	SemaSK-O1	SemaSK
IN	0.11	0.22	0.28	0.62	**0.72**
NS	0.03	0.22	0.31	**0.57**	0.56
PH	0.03	0.17	0.29	**0.54**	0.50
SB	0.01	0.15	0.23	0.44	**0.49**
SL	0.09	0.20	0.30	0.63	**0.69**
Avg.	0.05	0.19	0.28 (+47%)	0.56 (+195%)	**0.59** (+211%)

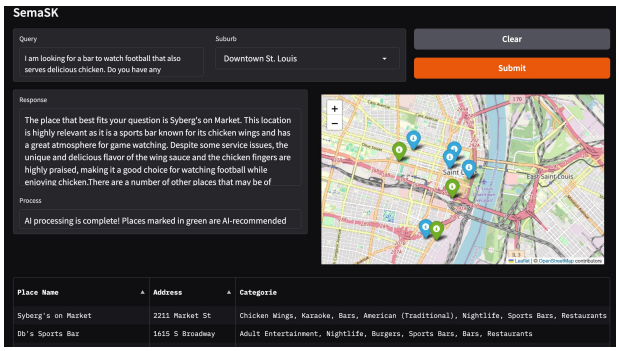

Figure 3: A screenshot of the SemaSK system.

F1@k improved by 195% and 211% compared to the best baseline, TF-IDF, respectively.

A further investigation reveals that both baselines (and similarly SemaSK-EM) have low precision which leads to their low F1 scores. In contrast, our systems use LLMs to refine the query answers, thereby enhancing precision and the overall F1 score.

Among our system variants, SemaSK achieves better results overall, while SemaSK-O1 has a higher F1 score on PH and is slightly better on NS. Despite being a newer model, *OpenAI o1-mini* is not better for the spatial keyword query task. Considering its higher cost, we default to using *GPT-4o*.

Between the two baselines, TF-IDF is more accurate, despite being a simpler model. This is because the queries and POI attributes are relatively short, making it difficult for LDA to learn accurate distributions for textual relevance measurement.

In terms of the query time, it takes 0.04 seconds on average to run the filtering step of SemaSK, while the refinement step depends on the LLM, which typically takes 2-3 seconds per query.

5 DEMONSTRATION

Our SemaSK system demo allows users to input queries and observe the query answers on a map. Figure 3 shows the system UI. There is a user input panel at the top, for users to select the region to query (we limit the query range to the different suburbs for simplicity) and enter a short sentence to describe the query target. Here, the sample query is: "*I am looking for a bar to watch football that also serves delicious chicken. Do you have any recommendations?*" in the neighborhood of "*Downtown St. Louis*".

When the query is submitted, SemaSK executes its query algorithm and displays the results in a map view located at the middle right of the UI. The green markers represent POIs recommended by the LLM, while the blue markers represent POIs fetched based on embedding similarity but filtered out by the LLM. Details of the POIs are listed at the bottom of the UI, while the top recommendation is detailed to the left of the map view. When a POI marker is clicked on the map, details on why the LLM has or has not recommended it will also be shown to the left of the map view.

6 CONCLUSION

We leveraged the strong semantic capabilities of LLMs and presented a system named SemaSK for spatial keyword query processing. SemaSK takes an RAG-based, filtering-and-refinement query procedure. It computes embeddings for geo-textual objects and input queries, which are used to efficiently retrieve geo-textual objects relevant to a query. Raw attributes of the retrieved objects are then fed into an LLM together with the query for result refinement. Our system demonstrates the potential of using LLMs for semantics-aware spatial keyword query processing, opening opportunities for further studies on semantics-aware query processing and practical deployment of the system.

ACKNOWLEDGMENTS

This work is in part supported by the Australian Research Council (ARC) via Discovery Projects DP230101534 and DP240101006. Jianzhong Qi is supported by ARC Future Fellowship FT240100170.

REFERENCES

[1] 2019. https://www.yelp.com/dataset
[2] 2022. https://platform.openai.com/docs/models/gpt-3-5#gpt-3-5-turbo
[3] 2023. https://platform.openai.com/docs/models#gpt-4o
[4] 2023. https://platform.openai.com/docs/models#o1
[5] 2024. https://geocode.maps.co/
[6] 2024. https://openai.com/index/new-embedding-models-and-api-updates/
[7] 2024. https://qdrant.tech/qdrant-vector-database/
[8] Lisi Chen, Shuo Shang, Chengcheng Yang, and Jing Li. 2020. Spatial Keyword Search: A Survey. *GeoInformatica* 24, 1 (2020), 85–106.
[9] Gao Cong and Christian S. Jensen. 2016. Querying Geo-Textual Data: Spatial Keyword Queries and Beyond. In *SIGMOD*. 2207–2212.
[10] Gao Cong, Christian S. Jensen, and Dingming Wu. 2009. Efficient Retrieval of the Top-k Most Relevant Spatial Web Objects. *Proceedings of the VLDB Endowment* 2, 1 (2009), 337–348.
[11] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *NeurIPS*. 9459–9474.
[12] Zhisheng Li, Ken C.K. Lee, Baihua Zheng, Wang-Chien Lee, Dik Lee, and Xufa Wang. 2011. IR-Tree: An Efficient Index for Geographic Document Search. *IEEE Transactions on Knowledge and Data Engineering* 23, 4 (2011), 585–599.
[13] Yu A. Malkov and D. A. Yashunin. 2020. Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 42, 4 (2020), 824–836.
[14] Rohin Manvi, Samar Khanna, Marshall Burke, David Lobell, and Stefano Ermon. 2024. Large Language Models are Geographically Biased. In *ICML*. 34654–34669.
[15] Zhihu Qian, Jiajie Xu, Kai Zheng, Wei Sun, Zhixu Li, and Haoming Guo. 2016. On Efficient Spatial Keyword Querying with Semantics. In *DASFAA*. 149–164.
[16] Zhihu Qian, Jiajie Xu, Kai Zheng, Pengpeng Zhao, and Xiaofang Zhou. 2018. Semantic-aware Top-k Spatial Keyword Queries. *World Wide Web* 21, 3 (2018), 573–594.
[17] Yufan Sheng, Xin Cao, Yixiang Fang, Kaiqi Zhao, Jianzhong Qi, Gao Cong, and Wenjie Zhang. 2023. WISK: A Workload-aware Learned Index for Spatial Keyword Queries. *Proceedings of the ACM on Management of Data* 1, 2 (2023), 187:1–187:27.
[18] Jiabao Sun, Jiajie Xu, Kai Zheng, and Chengfei Liu. 2017. Interactive Spatial Keyword Querying with Semantics. In *CIKM*. 1727–1736.
[19] Hongfei Xu, Yu Gu, Yu Sun, Jianzhong Qi, Ge Yu, and Rui Zhang. 2020. Efficient Processing of Moving Collective Spatial Keyword Queries. *The VLDB Journal* 29, 4 (2020), 841–865.
[20] Chiqun Zhang, Antonios Karatzoglou, Helen Craig, and Dragomir Yankov. 2023. Map GPT Playground: Smart Locations and Routes with GPT. In *SIGSPATIAL*. 52:1–52:4.