

Research Article

MapQaTor: A System for Efficient Annotation of Map Query Datasets

Mahir Labib Dihan¹, Mohammed Eunos Ali¹, Md Rizwan Parvez²

1. Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Bangladesh; 2. Qatar Computing Research Institute, Doha, Qatar

Mapping and navigation services like Google Maps, Apple Maps, Openstreet Maps, are essential for accessing various location-based data, yet they often struggle to handle natural language geospatial queries. Recent advancements in Large Language Models (LLMs) show promise in question answering (QA), but creating reliable geospatial QA datasets from map services remains challenging. We introduce MapQaTor, a web application that streamlines the creation of reproducible, traceable map-based QA datasets. With its plug-and-play architecture, MapQaTor enables seamless integration with any maps API, allowing users to gather and visualize data from diverse sources with minimal setup. By caching API responses, the platform ensures consistent ground truth, enhancing the reliability of the data even as real-world information evolves. MapQaTor centralizes data retrieval, annotation, and visualization within a single platform, offering a unique opportunity to evaluate the current state of LLM-based geospatial reasoning while advancing their capabilities for improved geospatial understanding. Evaluation metrics show that, MapQaTor speeds up the annotation process by at least 30 times compared to manual methods, underscoring its potential for developing geospatial resources, such as complex map reasoning datasets. The website is live at: <https://mapqator.github.io/> and a demo video is available at: https://youtu.be/7_aV9Wmhs6Q.

1. Introduction

In recent years, mapping and navigation services have transformed the way individuals access and interact with location-based information. Platforms such as Google Maps¹ and Apple Maps² have become essential tools, providing users with features like route planning, nearby points of interest (POIs), and contextual data, including reviews and operating hours. However, while these services offer extensive geospatial data, they often struggle with understanding and processing natural

language queries. This limitation hampers their effectiveness for users seeking to obtain specific information or engage in more complex question-answering (QA) tasks.

Recent advancements in multi-agent and tool-augmented large language models (LLMs) demonstrate significant promise for complex reasoning, decision-making, and generation tasks across various application domains, including those that interact with domain-specific tools such as maps^{[1][2]}. Notable tasks like WebArena^[3] and VisualWebArena^[4] have been proposed with practical real-life applications involving map usage. However, despite these developments, there remains no straightforward method for LLMs to access the vast databases of map services. Currently, there are no dedicated platforms designed to efficiently annotate language-map reasoning tasks, such as question answering. This gap leads to significant challenges in creating reliable datasets for training and evaluating LLMs for geospatial reasoning tasks, as many existing approaches rely on manual data collection methods that result in inconsistencies, lack of reproducibility, and difficulties in tracking the origins of information.

To address these issues, we present MapQaTor, a web application designed to streamline the creation of map-based QA datasets. MapQaTor empowers researchers to seamlessly integrate with any map API in a plug-and-play manner, enabling them to gather, visualize, and annotate geospatial data with minimal setup. By caching API responses, the platform ensures a consistent ground truth, which enhances the reliability of the datasets, even as real-world information evolves over time.

In summary, in this demo we have made the following key contributions:

1. We propose a novel framework, MapQaTor, first of its kind, which simplifies the creation of reproducible map-based QA datasets and reduces reliance on manual data collection through seamless integration with any map API (e.g., Google Maps, Apple Maps, Openstreet maps, etc.) in a plug-and-play manner.
2. We provide visualization tools that facilitate better understanding and annotation of geospatial information.
3. We implement caching of API responses to ensure a consistent ground truth, enhancing the reliability of QA tasks over time.
4. We evaluate MapQaTor to estimate its usefulness and efficiency.

We have published the code on GitHub³ under the Apache 2 license.

2. MapQaTor

MapQaTor is a web-based platform designed to streamline the creation of reproducible, map-based question-answering (QA) datasets that can be used to evaluate and advance the geospatial reasoning abilities of large language models (LLMs). By integrating with any map API in a plug-and-play manner, MapQaTor enables users to efficiently gather, annotate, and visualize map data to support complex, location-based QA tasks. This section details the main components of the platform, its architecture, and its unique features. Figure 1 outlines the proposed framework, which enables users to interact with map APIs by submitting queries, processing responses, and visualizing data. The framework allows users to design question-answer pairs and export the dataset in JSON format for downstream applications. The whole working flow is shown using ten key steps. Example of step-by-step annotation process is provided in Appendix E.

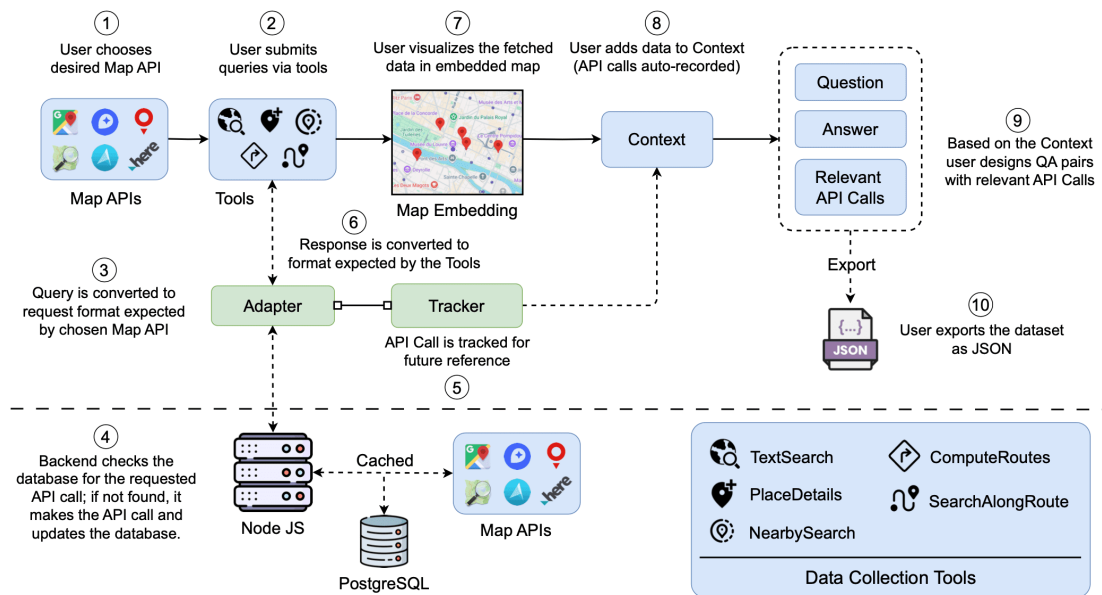


Figure 1. Overview of the annotation and visualization process of MapQaTor.

Tool	API Provider	API Endpoint
Text Search	Google Maps	Text Search (New) Places API
		Text Search Places API
	OpenStreetMap	Search queries Nominatim
	Mapbox	Suggest Search Box API
	TomTom	Point of Interest Search
	HERE	Discover Geocoding and Search
	Azure Maps	Search - Get Search Fuzzy
Place Details	Google Maps	Place Details (New) Places API
	OpenStreetMap	Place details Nominatim
	Mapbox	Retrieve Search Box API
	TomTom	Place by ID
	HERE	Lookup Geocoding and Search
	Azure Maps	Search - Get Search Fuzzy
Nearby Search	Google Maps	Nearby Search (New) Places API
	TomTom	Nearby Search
Compute Routes	Google Maps	Get a route Routes API
	OpenStreetMap	Routing API GraphHopper
	TomTom	Calculate Route
Search Along Route	Google Maps	Search along route
	TomTom	Along Search Route

Table 1. Current API Support for Data Collection Tools in MapQaTor

2.1. Plug-and-Play Architecture

A core design principle of MapQaTor is its adaptable architecture, allowing seamless integration with multiple map APIs. The platform incorporates an adapter layer that standardizes API requests and responses, making it compatible with any map service (e.g., Google Maps, Openstreet Maps, etc.). For each map API, MapQaTor employs a custom adapter (Details in A) that translates incoming requests into the format required by the target API and then converts the API's responses back to a standardized format that the platform can process. This plug-and-play architecture allows for easy expansion, enabling users to incorporate different map providers with minimal configuration.

2.2. Caching Mechanism

To enhance efficiency and ensure consistency, MapQaTor caches API responses in a PostgreSQL database. This caching mechanism not only reduces the number of repeated API calls, saving time and resources, but also ensures that the ground truth data remains consistent over time. By storing API responses in a JSONB column, the platform enables efficient retrieval of previously fetched data, which is particularly valuable when querying the same locations or routes multiple times. The caching mechanism thereby contributes to faster performance and more reliable QA dataset creation, even as real-world map data continues to evolve.

2.3. Visualization Tools

For visualizing geospatial data, MapQaTor utilizes the Google Maps JavaScript API⁴ to display places and routes directly on an embedded map. Users can view places as markers and visualize route paths, offering an interactive and intuitive experience for exploring map data. This visualization capability supports users in understanding spatial relationships, which is essential for designing complex map-based questions and ensuring that annotations accurately reflect the geospatial context. For further details, please refer to Appendix B.

2.4. Data Collection Tools

In MapQaTor, we have implemented five essential tools/features that leverage the capabilities of various map APIs for efficient data collection. The current API support for these tools is summarized in Table 1, highlighting the specific APIs integrated with each tool and illustrating how easily new APIs

can be incorporated to meet diverse user needs. For demonstration purposes, we have ensured that each tool integrates with at least two different APIs.

2.4.1. Text Search

This feature allows users to perform searches for specific places or types of locations using natural language queries as shown in Figure 2. By utilizing the search capabilities inherent to most map APIs, users can quickly retrieve relevant results, enhancing the efficiency of data collection.

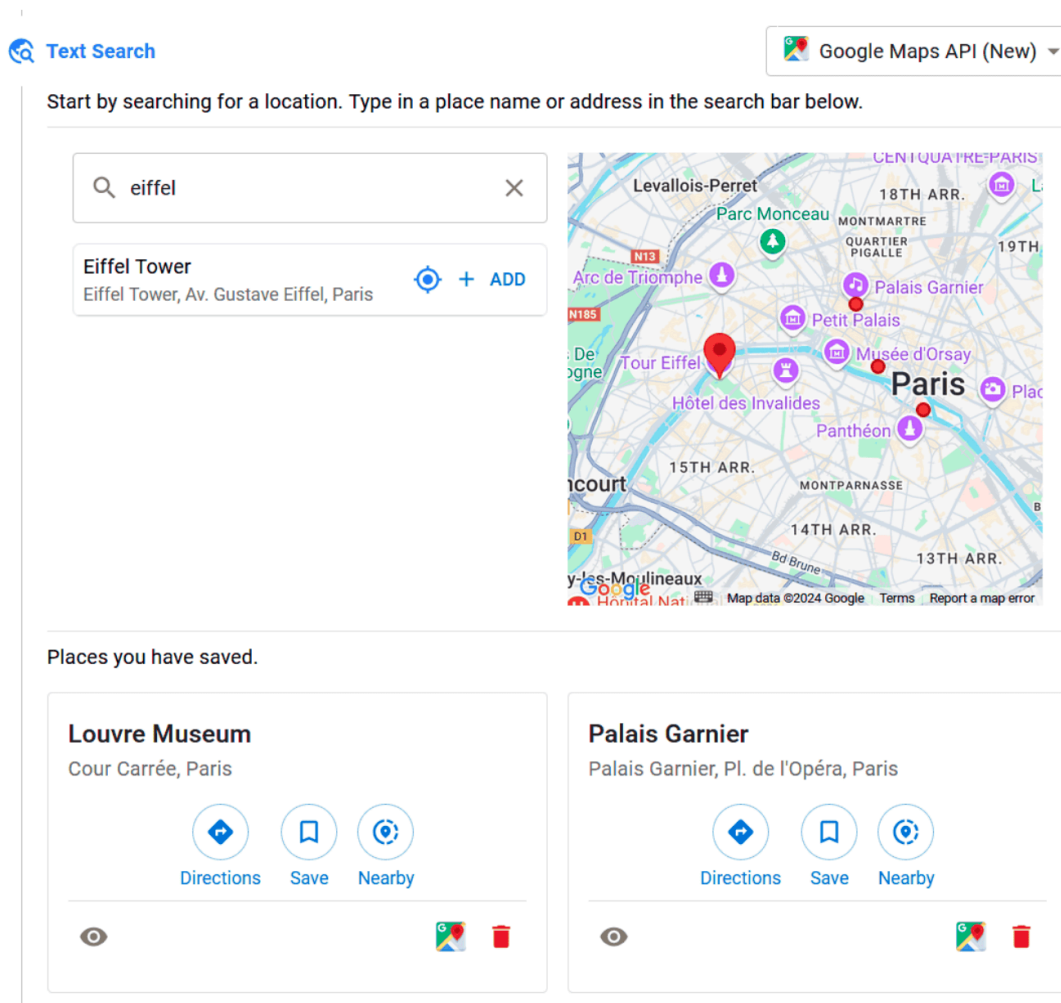


Figure 2. Search for a place

2.4.2. Place Details

With this tool, users can obtain detailed information about selected locations (see Figure 3). By accessing attributes such as name, address, ratings, opening hours, and reviews, users can enrich their understanding of each place, which is crucial for creating accurate QA pairs that capture essential context.

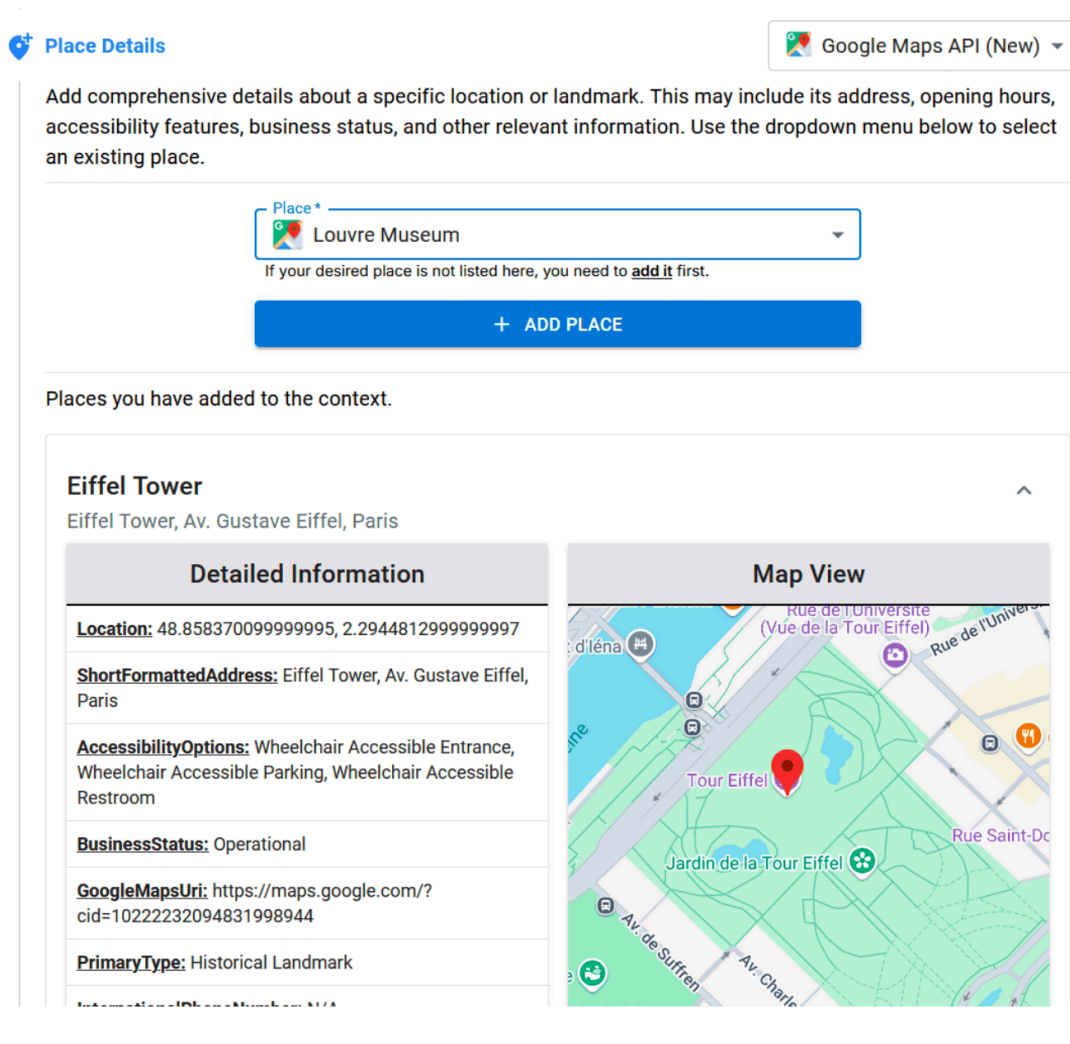


Figure 3. Fetch full details of a place

2.4.3. Nearby Search

This feature empowers users to discover points of interest in close proximity to a specified location (see Figure 4). By leveraging the nearby search functionality commonly available in map APIs, users

can explore related locations, providing valuable context for their QA pairs.

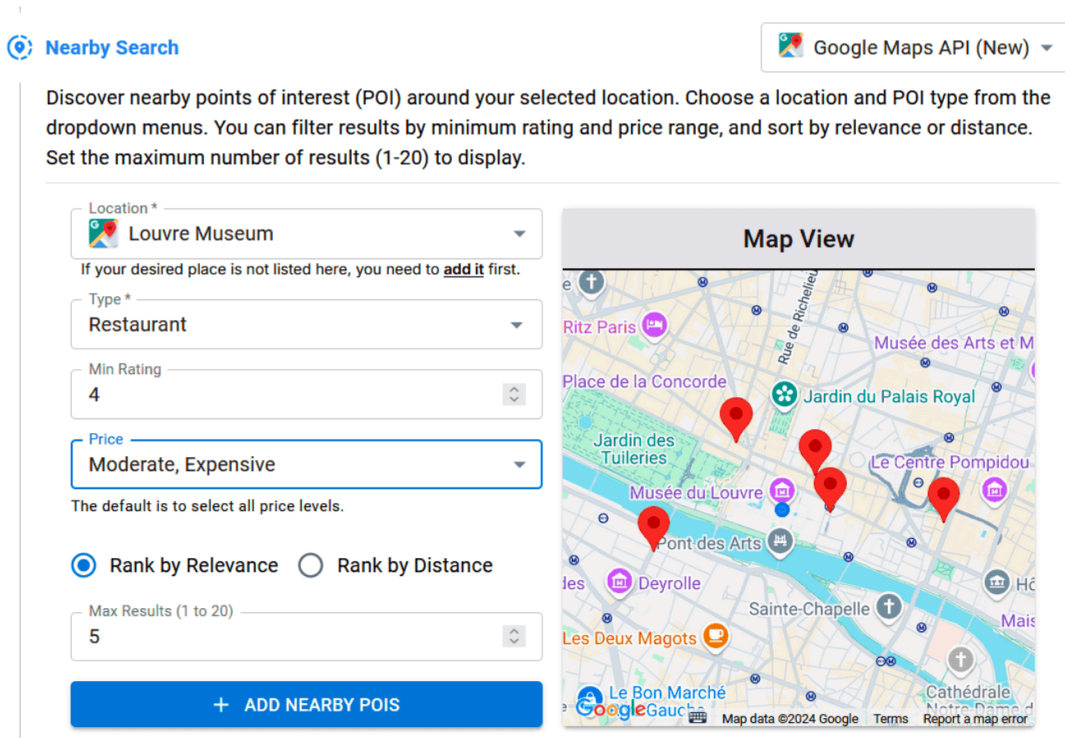


Figure 4. Search Nearby Places

2.4.4. Compute Routes

This feature finds alternate directions/routes from one place to another for different travel modes (see Figure 5). Users can view different routes and step-by-step navigation. This functionality is particularly useful for developing questions related to navigation. For multi-stop routes, intermediate waypoints can be optimized, which is useful for trip-related queries.

Compute Routes

Google Maps API (New)

Find routes between two points. Select origin, destination, and travel mode. Optionally, add intermediate stops, avoid specific road features, and compute alternative routes. Toggle 'Optimize intermediates order' to find the most efficient route through all points.

Origin *
Louvre Museum
If your desired place is not listed here, you need to [add it](#) first.

Destination *
Eiffel Tower
If your desired place is not listed here, you need to [add it](#) first.

Travel mode *
Driving

Avoid
Tolls, Highways

Compute alternative routes

Intermediates *
Palais Garnier Cathédrale Notre-Dame de Paris
If your desired place is not listed here, you need to [add it](#) first.

Optimize intermediates order

[+ ADD ALTERNATIVE ROUTES](#)

Map View

The map view displays a route in Paris, France, starting at the Louvre Museum and ending at the Eiffel Tower. The route is shown as a blue line on a map. Intermediate stops are marked with red pins at the Palais Garnier and the Cathédrale Notre-Dame de Paris. The map includes labels for various Parisian landmarks and districts, such as Levallois-Perret, Montmartre, and Montparnasse. The map data is attributed to Google, 2024.

Figure 5. Find routes between places

2.4.5. Search Along Route

This feature allows users to identify points of interest along a defined route (see Figure 6). Users can input their desired path and receive a list of relevant locations, enhancing the contextual richness of their QA pairs by exploring how different places relate to one another during travel.

Search Along Route

Google Maps API (New)

Find places of interest along a route. Select origin, destination, and travel mode. Specify the type of place, minimum rating, and price range. Choose to rank results by relevance or distance from origin.

Origin *
Louvre Museum
If your desired place is not listed here, you need to [add it](#) first.

Destination *
Eiffel Tower
If your desired place is not listed here, you need to [add it](#) first.

Travel mode *
Driving

Avoid
Tolls

Type *
Restaurant

Min Rating
4

Price
Very Expensive, Expensive, Moderate
The default is to select all price levels.

Rank by Relevance Rank by Distance

Max Results (1 to 20)
10

[+ ADD PLACES](#)

Map View

The map view displays a route from the Louvre Museum to the Eiffel Tower. The route is shown as a blue line on a map of Paris. Several red pins are placed along the route, indicating search results for restaurants. The map includes labels for various landmarks and streets, such as Parc Monceau, Palais Garnier, Petit Palais, Place de la Concorde, Le Bon Marché Rive Gauche, and Tour Montparnasse. The map data is attributed to Google, 2024.

Figure 6. Search places along a route

After collecting necessary data, users need to store it as Context, which represents a collection of API calls and their responses (Details in Appendix D.1).

2.5. Question Design and Annotation

The Question Design and Annotation feature in MapQaTor facilitates the creation and management of questions, enhancing the process of generating high-quality QA pairs (see Figure 7). It supports four question formats: Yes/No, Single Choice, Multiple Choice, and Open Ended, allowing users to select the format that best suits their needs (Details in Appendix D.3).

Users can assign categories to each question, enabling better organization and retrieval based on thematic relevance. Also, while writing question/answer user will get Place Name suggestions to ensure consistency and uniqueness (Details in Appendix D.2).

Since multiple questions can be designed based on a single Context and not all the information (e.g., API responses) in the Context is relevant to each QA pair, users need to specify which pieces of information in the Context are needed to answer each question (Details in Appendix D.4). This process is crucial, as it allows the system to link relevant API calls to the corresponding QA pairs seamlessly. This integration streamlines the question-creation process while ensuring that each QA pair is backed by verifiable data sourced from the underlying map APIs. By maintaining a clear connection between questions and API calls, MapQaTor enhances the reliability and traceability of the generated datasets, making it easier for researchers to validate and reproduce results.

Context: [EDIT CONTEXT](#)

Visualize Context ▼

(1) Detailed information of Eiffel Tower
 (2) Nearby Restaurants of Louvre Museum with a minimum rating of 4 and price levels Moderate or Expensive
 (3) Optimized Driving route from Louvre Museum to Eiffel Tower via Palais Garnier, Cathédrale Notre-Dame de Paris (Avoiding tolls, highways)

Question 1: ^

I will drive from Louvre Museum to Eiffel Tower via Palais Garnier, Cathédrale Notre-Dame de Paris. What is the most efficient itinerary? I want to avoid tolls and highways.

Category * i ▼

Trip

Answer Type:

Open Ended ▼

Correct Answer:

Louvre Museum -> Cathédrale Notre-Dame de Paris -> Palais Garnier -> Eiffel Tower

References ⓘ

3 ▼

[+ ADD QUESTION](#)

Figure 7. Create a question, provide options, and annotate the correct answer.

2.6. Tracking and Traceability

MapQaTor includes a tracking feature that logs all API calls made during the annotation process. As users interact with the platform, every API call—whether retrieving nearby points of interest, route information, or other geospatial data—is automatically logged. This logging process ensures that each

interaction is documented, capturing the request details, response data, and the corresponding timestamps. This feature allows users to associate specific API calls with each question-answer pair, ensuring that the origin of each answer is traceable to a specific data source. Additionally, this could provide valuable insights for training API retrieval models^[2]. This level of traceability is crucial for creating reproducible datasets, as it allows users to review and verify the data used for each question and answer, reinforcing the reliability of the QA dataset.

2.7. Technology Stack

The following section details the technology stack employed in the development and operation of our system.

Frontend: The user interface is built using Next.js, a React framework that provides server-side rendering and static site generation.

Backend: Our server-side logic is implemented in Node.js with Express.js, providing a robust and scalable environment for handling API requests and managing business logic.

Database: We utilize PostgreSQL for its relational database capabilities, supporting complex queries and ensuring data integrity.

2.8. Application Scenarios

MapQaTor is primarily designed to support the creation of a comprehensive and everyday-use geospatial question answering (QA) dataset, with the potential to benchmark large language models (LLMs), assess their capabilities, and identify areas for improvement in geospatial reasoning tasks. Using MapQaTor's plug-and-play architecture, users have the flexibility to evaluate the richness and capabilities of any available map service across a wide range of location-based services.

3. Experiments and Evaluation

Evaluating MapQaTor presented unique challenges due to the lack of any existing system with similar capabilities for creating and managing map-based question-answering datasets. As a first-of-its-kind platform, direct comparisons to other tools were infeasible. Instead, we designed an experiment to measure the efficiency of MapQaTor in comparison to manual data collection methods currently used by researchers (see Table 2).

To simulate the manual method, two human participants were tasked with retrieving map data from Google Maps using common search functions and then manually copying the necessary textual information to ensure traceability. This included gathering information on Place Details, Nearby Search, Route Computation, and Search Along Route. We then used MapQaTor to retrieve the same data, noting the time taken for each method across these four features. The results demonstrate a significant improvement in data retrieval speed, with MapQaTor requiring at least 30 times less time than the manual approach.

Task	MapQaTor	Manual
Place Details	10.17 sec	487 sec
Nearby Search	12.50 sec	456 sec
Compute Routes	14 sec	516.5 sec
Search Along Route	15.66 sec	476 sec

Table 2. Quantitative comparison between our system and manual methods

4. Related Works

Recent research has highlighted the potential of map data in mimicking real-world planning tasks through various tools^{[5][6]}. Additionally, studies emphasize the significance of caching API call results to establish a stable database for evaluation purposes^{[7][5]}. The development of web-based platforms for integrating geospatial data has also been explored, focusing on streamlining data collection and enhancing the usability of geospatial information for research and development^{[8][9][10]}.

While tool-calling datasets like ToolBench^[2] and APiBank^[11] include location-based tasks, their data collection processes lack traceability and reproducibility. This limitation highlights a significant gap in the current landscape: the development of datasets for geospatial question answering is still in its infancy. Existing resources often fail to capture the rich contextual information provided by modern map services. Therefore, there is a pressing need for innovative approaches that effectively leverage the extensive data available from map services to create comprehensive geospatial QA datasets.

5. Conclusion

In this paper, we have proposed a novel framework, MapQaTor, first of its kind, to automatically fetch rich contextual map service data, which forms the basis to develop language-map benchmark datasets for evaluating SoTA LLMs. Our developed web platform simplifies data collection for users by offering precise spatial information, user-friendly search, and efficient data retrieval by using Map APIs. Our application also enables user to create geospatial questionnaire. Experimental evaluation suggests that MapQaTor is highly effective in developing geospatial question answer datasets. We believe this approach introduces a new task in geospatial question answering, which has the potential to open a new research direction in the intersection of language models and spatial reasoning. Furthermore, our framework can be adapted to other domains.

Limitations

Despite the capabilities of MapQaTor, several limitations should be acknowledged. The platform utilizes several paid map APIs, which may incur costs based on usage. During the current public demonstration period, users can explore its features without immediate expenses; however, in the long run, users will need to host the platform independently and integrate their own API keys to access paid functionalities. This requirement necessitates an understanding of the pricing structures associated with the various APIs, potentially impacting accessibility for some users. The platform's functionality is heavily dependent on the availability and stability of external map APIs, meaning that any changes or deprecations in these APIs could negatively impact performance. The quality of the generated QA pairs is contingent on the retrieved data and users' ability to formulate meaningful questions, which can introduce variability in dataset quality. The evaluation metrics used might not encompass all aspects of usability, possibly overlooking qualitative user feedback. In addition to map service data, other platforms such as Trip Advisor can also be a rich source of additional context for geospatial queries.

Appendix A. Adapter Layer

The Adapter Layer in MapQaTor provides base classes for various tools, including 'TextSearch', 'PlaceDetails', 'NearbySearch', 'ComputeRoutes', and 'SearchAlongRoute'. To integrate a new API under any tool, developers need to extend the specific base class and implement two methods:

'convertRequest' and 'convertResponse'. These methods standardize the request and response formats to align with MapQaTor's requirements, ensuring consistent interaction across APIs. Figure 8 demonstrates an implementation example of extending the 'TextSearch' base class to support the TomTom API. Note that "key:TOMTOM_API_KEY" is used as a placeholder, which will be replaced in the backend using environment variables to prevent exposing the API key on the frontend.


```

class TomTomApi extends TextSearch {
  constructor() {
    super();
    this.family = "tomtom";
  }

  convertRequest = (query) => {
    return {
      url: "https://api.tomtom.com/
search/2/poiSearch/" + query + ".
json",
      method: "GET",
      params: {
        key: "key:TOMTOM_API_KEY",
        limit: 5,
        language: "en-US",
      },
    };
  };

  convertResponse = (data) => {
    const places = data.results.map((
place) => ({
  id: place.id,
  displayName: {
    text: place.poi.name,
  },
  shortFormattedAddress: place.
address.freeformAddress,
  location: {
    latitude: place.position.lat,
    longitude: place.position.lon,
  },
}));
    return { places };
  };
}

```

Figure 8. Implementing the TomTom API Adapter for Text Search in MapQaTor

Appendix B. Visualization Tools

For visualizing geospatial data, MapQaTor utilizes the Google Maps JavaScript API to display places (see Figure 10) and routes (see Figure 11) directly on an embedded map. Markers are used in the map to

represent places. It just needs the latitude and longitude of the place. On the other hand, routes are actually encoded polylines⁵. But embeded map expects a set of coordinates along the path to visualize the route. So, we need to first decode the encoded polyline and convert it to a set of latitude and longitude. The decoding algorithm is shown in Figure 9.

```
function decodePolyline(polylineStr) {
  let index = 0, lat = 0, lng = 0;
  const coordinates = [];
  const changes = { latitude: 0,
    longitude: 0 };

  while (index < polylineStr.length) {
    for (const unit of ["latitude", "
      longitude"]) {
      let shift = 0;
      let result = 0;

      while (index < polylineStr.length)
      {
        let byte = polylineStr.
          charCodeAt(index++) - 63;
        result |= (byte & 0x1f) << shift;
        shift += 5;
        if (byte < 0x20) break;
      }

      changes[unit] = result & 1 ? ~(
        result >> 1) : result >> 1;
    }

    lat += changes.latitude;
    lng += changes.longitude;

    coordinates.push({ lat: lat / 1e5,
      lng: lng / 1e5 });
  }

  return coordinates;
}
```

Figure 9. Polyline Decoding Algorithm

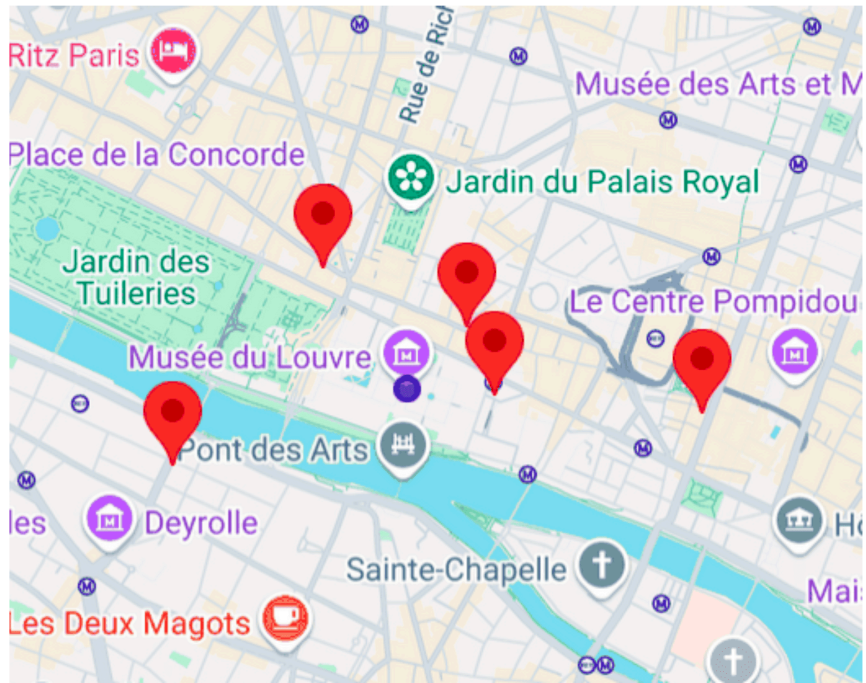


Figure 10. Set of markers indicating different places

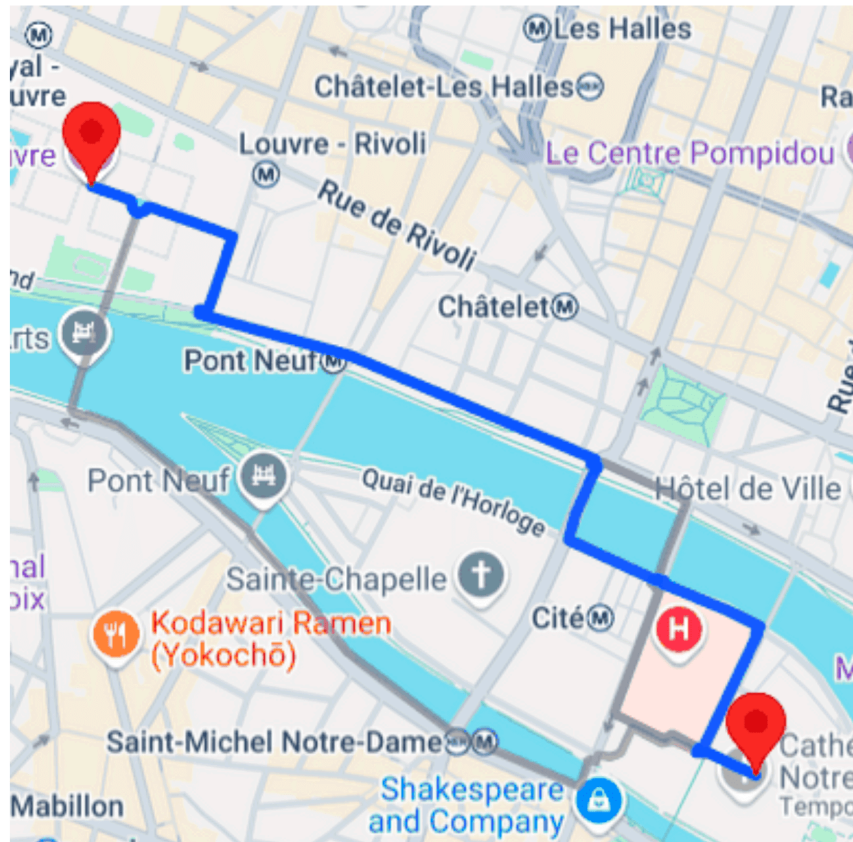


Figure 11. Visualizing routes between places

Appendix C. Exclusion of Temporal Variations in Routing APIs

To ensure reproducibility in the dataset, we have implemented specific measures to eliminate temporal variations in routing responses. These measures are essential for maintaining consistency across identical queries, regardless of when they are made.

Traffic Awareness Setting: In MapQaTor, routing APIs are set to "TRAFFIC_UNAWARE," meaning no real-time traffic data is used in generating routes and travel times. Real-time traffic data would introduce variability in travel times, as identical requests made at different times would yield different results based on current traffic conditions. By disabling traffic awareness, we ensure that each query consistently returns the same travel time, independent of real-time conditions, thereby enhancing dataset reproducibility.

Exclusion of Transit Mode: We have also excluded the "TRANSIT" travel mode from 'ComputeRoutes' and 'SearchAlongRoutes' tools. Transit routes, which include public transportation options like buses

and trains, are heavily influenced by the specific time a request is made due to schedules, service frequency, and operational hours. Including transit mode would introduce further temporal variability, as the same query could yield different results depending on schedule changes, service frequency, and operational hours. This would undermine the consistency of the dataset, as identical queries could yield different travel times and routes.

Benefits of These Measures: By setting routes to be traffic-unaware and omitting transit mode, we ensure that:

- Responses remain consistent: Identical queries yield the same routes and travel times, regardless of request timing.
- Benchmarking is simplified: Model evaluations can focus purely on spatial and routing reasoning without accounting for real-time variations.
- Reproducibility is maintained: The dataset provides a stable baseline for testing and comparing model performance over time.

These adjustments support our goal of creating a reproducible benchmark dataset, free from the temporal dependencies that could otherwise complicate consistent evaluation of geospatial reasoning tasks.

Appendix D. Question Design and Annotation

D.1. Context Preview

During data collection process, users need to save the necessary fetched information, which we will refer to as Context. Context consists of a collection of API calls and their responses. After creating a context, annotators will proceed to create questions. They can view the created context in two mode.

1. **Summarized Context:** A textual summarized view as a list of fetched information. One liner for each information (see Figure 12).

Context: ✎ EDIT CONTEXT

Visualize Context ▾

- (1) Detailed information of Eiffel Tower
- (2) Nearby Restaurants of Louvre Museum with a minimum rating of 4 and price levels Moderate or Expensive
- (3) Optimized Driving route from Louvre Museum to Eiffel Tower via Palais Garnier, Cathédrale Notre-Dame de Paris (Avoiding tolls, highways)

Figure 12. Summarized Context

2. **Visual Context:** In visual context, full details of each information in the context is shown with corresponding Map View ((see Figure 12).

Summarize Context ▲

Places you have added to the context.

Eiffel Tower ▾
Eiffel Tower, Av. Gustave Eiffel, Paris

List of places whose nearby pois are added to the context

Louvre Museum ▲

5 Restaurants Rating ≥ 4

Prices: Moderate, Expensive

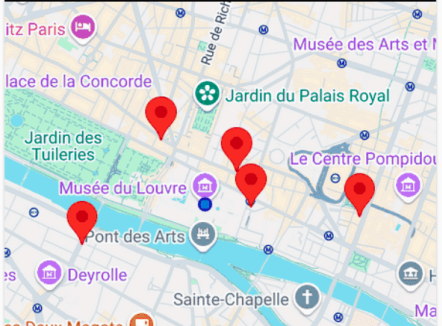
Nearby Restaurants	Map View
<p>Le Fumoir 4.4* (3890) Moderate 🚶 242s (300m)</p>	
<p>The Little Machon 4.1* (401) Moderate 🚶 264s (306m)</p>	
<p>Café de la Régence 4.3* (3087) Moderate 🚶 460s (541m)</p>	
<p>Chouchou 4.5* (5505) Moderate 🚶 903s (1098m)</p>	
<p>Les Antiquaires</p>	

Figure 13. Visual Context

D.2. Place Name Suggestion

Annotators gather different places using TextSearch tool. At the time of writing question, on pressing '@' annotator will get the option to choose from available place names. On pressing enter, the selected name will be entered on the cursor position. This ensures the consistency of place names between context and questions in a hassle free way. This is also applicable in answer field.

Question 1:

I will drive from @ to Eiffel Tower via Palais Garnier, Cathédrale Notre-Dame de Paris. What is the most efficient itinerary? I want to avoid tolls and highways.

Tip: Press "@" for place suggestions

Cathédrale Notre-Dame de Paris

Palais Garnier

Louvre Museum

Eiffel Tower

Figure 14. Suggesting available places from the context

D.3. Answer Formats

We incorporate four distinct answer formats, which provide flexibility in how users interact with the system. These formats are crucial for creating a diverse set of training and evaluation data for large language models (LLMs). By supporting a range of response types, the dataset captures various ways that models must handle user queries and generate appropriate answers. The four answer formats are as follows:

1. **Open Ended:** The Open Ended format allows for unrestricted, free-text responses, enabling models to generate detailed and contextually rich answers. This format tests the model's ability to understand and generate responses that may require creativity, explanation, or reasoning. It is particularly useful for questions that demand nuanced answers or elaboration.

Answer Type:
Open Ended

Correct Answer:
Louvre Museum -> Cathédrale Notre-Dame de Paris -> Palais Garnier -> Eiffel Tower

Tip: Press "@" for place suggestions

Figure 15. Answer format: Open Ended

2. **Yes/No:** The Yes/No format tests the model's ability to provide binary, definitive answers. This format is valuable for questions that require a clear-cut response, such as confirming or denying information. The model's performance in this format reflects its ability to handle factual questions with straightforward answers.

Answer Type:
Yes/No

Correct Answer:
 No
 Yes

Figure 16. Answer format: Yes/No

3. **Multiple Choice:** The Multiple Choice format allows the model to choose from a predefined set of possible answers, potentially selecting more than one option. This format challenges the model to identify the most relevant responses from a set of alternatives and is useful for assessing its ability to discern multiple correct answers or filter out incorrect options.

Answer Type:

Multiple Choice ▼

Options:

Option 1 *
Option 1
Tip: Press "@" for place suggestions

Option 2 *
Option 2
Tip: Press "@" for place suggestions

Option 3 *
Option 3
Tip: Press "@" for place suggestions

Option 4 *
Option 4
Tip: Press "@" for place suggestions

Correct Answer:

Option 1

Option 2

Option 3

Option 4

Figure 17. Answer format: Multiple Choice

4. **Single Choice:** Similar to Multiple Choice, but with the restriction that only one option can be selected. This format is used to evaluate the model's ability to determine the most appropriate single answer from a set of alternatives. It tests the model's decision-making and accuracy in selecting the correct response.

Answer Type:

Single Choice

Options:

Option 1 *

Option 1

Tip: Press "@" for place suggestions

Option 2 *

Option 2

Tip: Press "@" for place suggestions

Option 3 *

Option 3

Tip: Press "@" for place suggestions

Option 4 *

Option 4

Tip: Press "@" for place suggestions

Correct Answer:

Option 1

Option 2

Option 3

Option 4

Figure 18. Answer format: Single Choice

D.4. Reference Field

After creating a QA pair, annotator need to select which of the informations in the context is actually relevant to this pair (see Figure 19). We are focusing on the informations fetched through PlaceDetails, NearbySearch, ComputeRoutes and SearchAlongRoute tool. This step is necessary to create a well-grounded and traceable dataset. This ensures that all the necessary information to answer a question is available in the context. Under the hood, corresponding api calls (Relevant API Calls) to each information are linked with the QA pair (see Fig. 35).

Correct Answer:

Louvre Museum -> Cathédrale Notre-Dame de Paris -> Palais Garnier -> Eiffel Tower

Tip: Press "@" for place suggestions

References ⓘ

3

- (1) Detailed information of Eiffel Tower
- (2) Nearby Restaurants of Louvre Museum with a minimum rating of 4 and price levels Moderate or Expensive
- (3) Optimized Driving route from Louvre Museum to Eiffel Tower via Palais Garnier, Cathédrale Notre-Dame de Paris (Avoiding tollshighways)

Figure 19. Choosing relevant informations to provide the correct answer.

Appendix E. Example Annotation

On entering MapQaTor, we will see 3 major steps (see Fig. 20). First we need to design a context, which contains all the necessary information to design question-answer (QA) pairs. Then we will create QA pairs. And finally we can review and save the dataset.

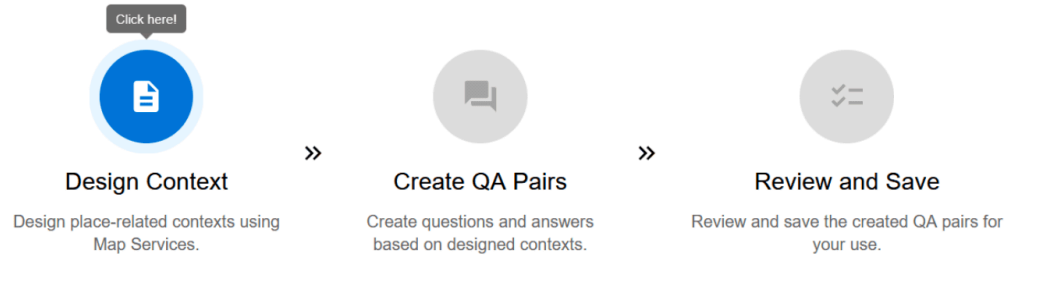


Figure 20. Major steps of MapQaTor

As described in section 2.4, we can design context using 5 data collection tools (see Fig. 21).

Guidelines

Welcome to the Context Generator! This tool helps you create rich, place-related contexts using Maps APIs. Here's how it works:



Text Search

Search for and add key locations to your context.



Place Details

Add detailed information about a place.



Nearby Search

Discover nearby places and points of interest.



Compute Routes

Find routes between places or multi-stop routes.



Search Along Route

Find places along a route.

Let's begin!

START

Figure 21. Overview of data collection tools integrated into MapQaTor, showcasing essential functionalities.

Now let's start designing a context. Let's say we want to create questions on Louvre museum and Eiffel tower. For that we first need to search them using TextSearch tool and add them to context (see Fig. 22).

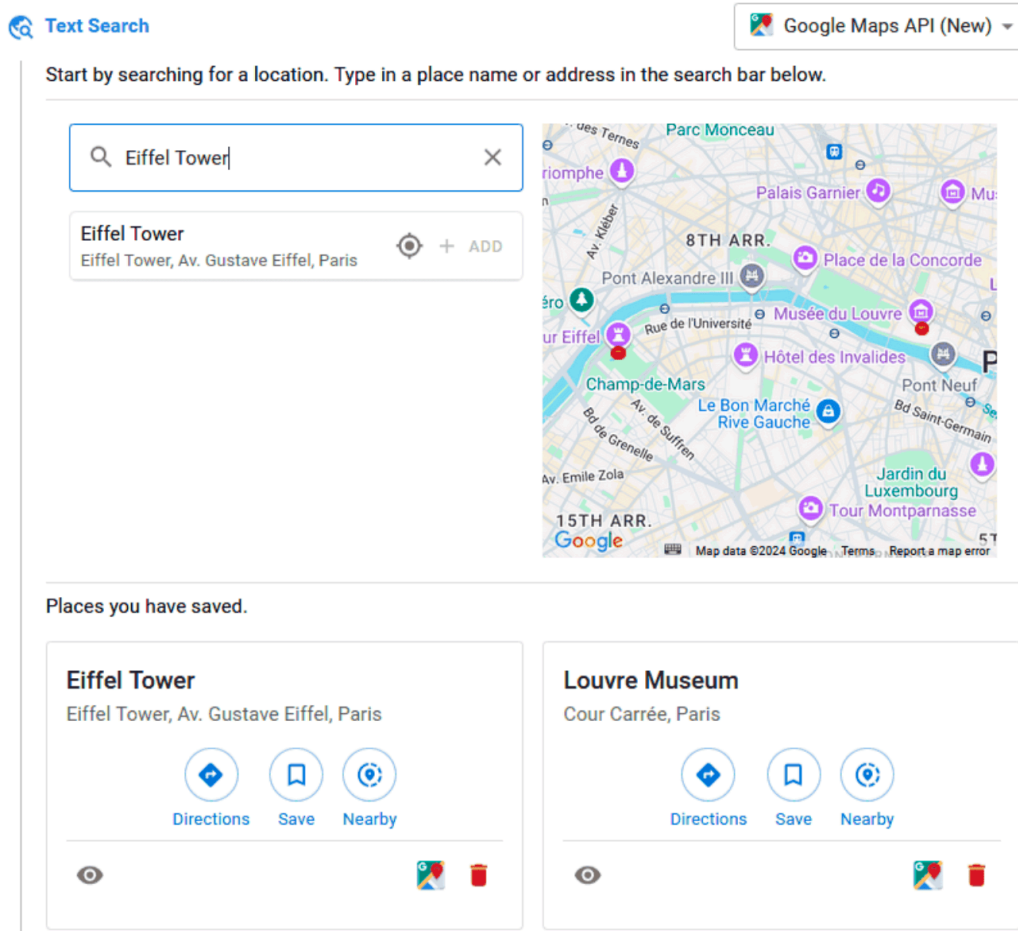


Figure 22. Example use of TextSearch tool

Next, we can search for the nearby restaurants of Eiffel Tower (see Fig. 23). Once we submit query, nearby places of Eiffel Tower will be added to the context and we can see the list of places with some key details (see Fig. 24).

Nearby Search

Google Maps API (New) ▾

Discover nearby points of interest (POI) around your selected location. Choose a location and POI type from the dropdown menus. You can filter results by minimum rating and price range, and sort by relevance or distance. Set the maximum number of results (1-20) to display.

Location *
Eiffel Tower ▾
If your desired place is not listed here, you need to **add it** first.

Type *
Restaurant ▾

Min Rating
0 ▾

Price
▾
The default is to select all price levels.

Rank by Relevance Rank by Distance

Max Results (1 to 20)
5 ▾

+ ADD NEARBY POIS

CONTINUE [BACK](#)

Map View

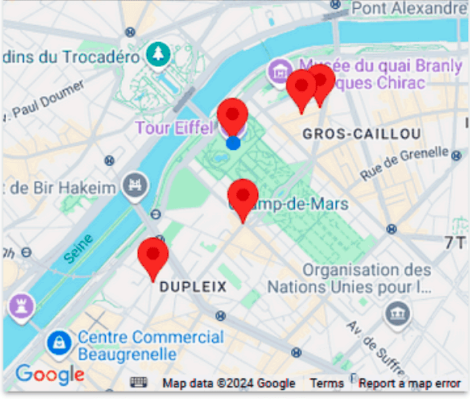
The map view displays a street map of Paris, France, centered on the Eiffel Tower. Several red location pins are placed on the map, indicating nearby restaurants. The pins are located in the Gros-Caillois district, near the Seine river. Labels on the map include 'Tour Eiffel', 'GROS-CAILLOU', 'DUPLEIX', 'Centre Commercial Beaugrenelle', 'Museum du quai Branly Jacques Chirac', and 'Organisation des Nations Unies pour l...'. The map also shows the Seine river and various streets like 'Rue de Grenelle' and 'Av. de Suffren'. The Google logo and 'Map data ©2024 Google' are visible at the bottom of the map.

Figure 23. Example use of NearbySearch tool

List of places whose nearby pois are added to the context

Eiffel Tower

5 Restaurants

Nearby Restaurants	Map View
Café Fleur 4.3* (2766) 🚶 440s (540m)	
Jules Verne 4.4* (2842) Very Expensive 🚶 0s (undefine...)	
Linette 4.5* (1543) Moderate 🚶 466s (561m)	
L'Abreuvoir 4.6* (3451) Moderate 🚶 980s (1185m)	
Au bon accueil 4.5* (732) Moderate 🚶 369s (440m)	

CONTINUE **BACK**

Figure 24. Nearby restaurants of Eiffel Tower

We can also search for available routes from Louvre Museum to Eiffel tower using ComputeRoutes tool (see Fig. 25). After submitting query, all the available routes with distance, duration and step-by-step navigation will be added to the context (see Fig. 26).

Compute Routes

Google Maps API (New)

Find routes between two points. Select origin, destination, and travel mode. Optionally, add intermediate stops, avoid specific road features, and compute alternative routes. Toggle 'Optimize intermediates order' to find the most efficient route through all points.

Origin *
Louvre Museum
If your desired place is not listed here, you need to **add it** first.

↕

Destination *
Eiffel Tower
If your desired place is not listed here, you need to **add it** first.

Travel mode *
Driving

Avoid

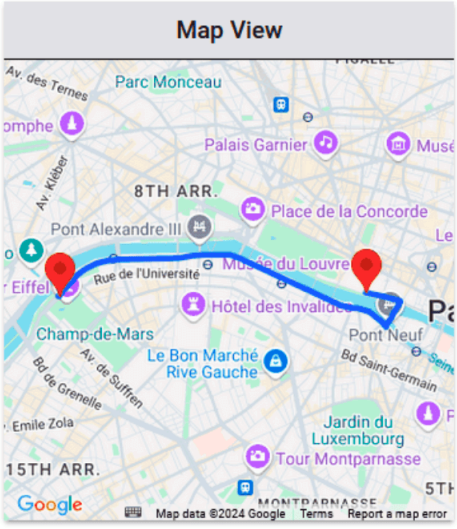
Compute alternative routes

Intermediates *
If your desired place is not listed here, you need to **add it** first.

Optimize intermediates order

+ ADD ALTERNATIVE ROUTES

Map View






The map view shows a route in Paris, France, starting at the Louvre Museum and ending at the Eiffel Tower. The route is highlighted in blue and passes through the 8th and 15th arrondissements. Key landmarks and streets visible include Parc Monceau, Palais Garnier, Place de la Concorde, Pont Alexandre III, Rue de l'Université, Musée du Louvre, Hôtel des Invalides, Le Bon Marché Rive Gauche, Jardin du Luxembourg, and Tour Montparnasse. The map also shows the 8th and 15th arrondissements, the Seine river, and various streets like Av. des Ternes, Av. Kléber, Av. de Grenelle, and Bd Saint-Germain.

Figure 25. Example use of ComputeRoutes tool

List of origin - destination pairs whose alternative routes are added to the context

Louvre Museum
↓
Eiffel Tower

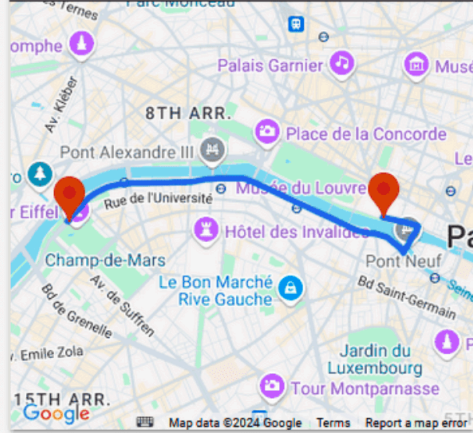



 1 route

Via Quai d'Orsay**15 mins (4.5 km)**

Navigation Instructions
Head east on Quai François Mitterrand toward Pont des Arts
1 min (0.2 km)
Continue onto Quai du Louvre
1 min (0.2 km)
Turn right onto Pont Neuf
2 mins (0.3 km)
Turn right onto Quai de Conti
1 min (0.4 km)
Quai de Conti turns slightly left and becomes Quai Malaquais

Map View



CONTINUEBACK

Figure 26. Available routes from Louvre museum to Eiffel tower by car

Thus we can finish designing context, and move forward to create questions based on the context. We can start with an example question "Fastest time to go from Louvre Museum to Eiffel Tower by car?" and assign it to "Routing" category (see Fig. 27).

Create QA Pairs based on the Context

Context: ✎ EDIT CONTEXT

Visualize Context ▾

(1) Nearby Restaurants of Eiffel Tower
(2) Driving route from Louvre Museum to Eiffel Tower

Question 1: ^

Fastest time to go from Louvre Museum to Eiffel Tower by car?

Tip: Press "@" for place suggestions

Category *
Routing i ▾

Figure 27. Example question creation (1)

Then we need to provide answer to the question. For our example question, we can get the answer from the context, which is "15 mins" (see Fig. 26). We can choose the answer type and provide the ground truth (see Fig. 28).

Answer Type:

Single Choice

Options:

Option 1 *
14 mins
Tip: Press "@" for place suggestions

Option 2 *
15 mins
Tip: Press "@" for place suggestions

Option 3 *
16 mins
Tip: Press "@" for place suggestions

Option 4 *
17 mins
Tip: Press "@" for place suggestions

Correct Answer:

Option 1

Option 2

Option 3

Option 4

Figure 28. Example answer creation (1)

Upto this point, only the annotator knows which information in the context is used to derive the ground truth. To maintain traceability we need to link the QA pair with relevant information in the context. In this example, only the routing information from Louvre Museum to Eiffel Tower is relevant. Nearby Restaurants of Eiffel Tower is irrelevant. So, we will select "Driving route from Louvre Museum to Eiffel Tower" in "References" field (see Fig. 29).

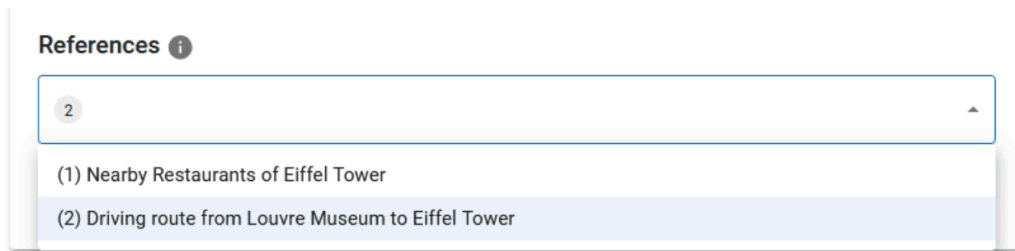


Figure 29. Choosing relevant informations (1)

Similarly, we can add a second question "Nearby restaurant of Eiffel Tower with over 2000 user ratings and at least 4.5* rating" (see Fig. 30), provide answer (see Fig. 31) and link relevant informations (see Fig. 32).

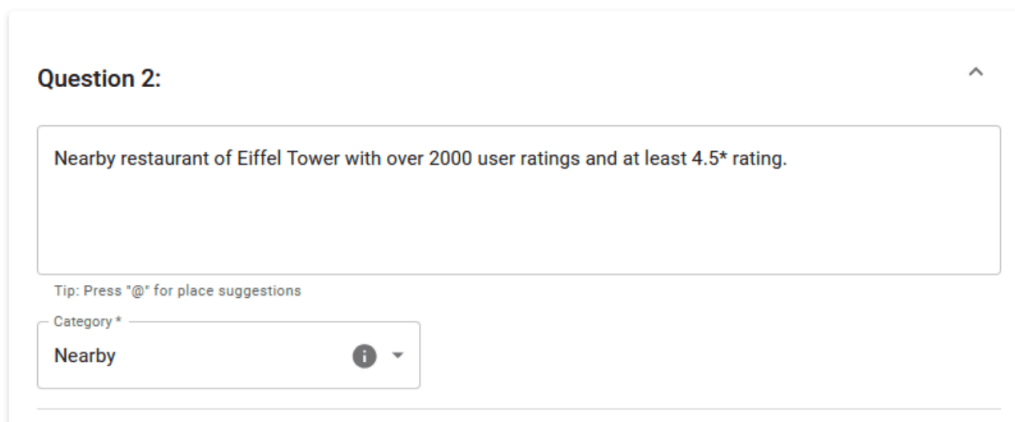


Figure 30. Example question creation (2)

Answer Type:

Single Choice

Options:

Option 1 *

Café Fleur

Tip: Press "@" for place suggestions

Option 2 *

Jules Verne

Tip: Press "@" for place suggestions

Option 3 *

Linette

Tip: Press "@" for place suggestions

Option 4 *

L'Abreuvoir

Tip: Press "@" for place suggestions

Correct Answer:

Option 1

Option 2

Option 3

Option 4

Figure 31. Example answer creation (2)

References ⓘ



1

(1) Nearby Restaurants of Eiffel Tower

(2) Driving route from Louvre Museum to Eiffel Tower



Figure 32. Choosing relevant informations (2)

Finally, we can review the context and QA pairs (see Fig. 33) and download the dataset in JSON format (see Fig. 34). Downloaded dataset format is shown in Figure 35.

 **Context**  EDIT CONTEXT

Visualize Context ▾

(1) Nearby Restaurants of Eiffel Tower
(2) Driving route from Louvre Museum to Eiffel Tower

 **Questions**  EDIT QUESTIONS

Question 1:

Fastest time to go from Louvre Museum to Eiffel Tower by car?

Options:

(1) 14 mins	(2) 15 mins ✓
(3) 16 mins	(4) 17 mins


Question 2:

Nearby restaurant of Eiffel Tower with over 2000 user ratings and at least 4.5* rating.

Options:

(1) Café Fleur	(2) Jules Verne
(3) Linette	(4) L'Abreuvoir ✓

Figure 33. Review context and QA pairs


Want to save this to dataset?
Log in to save your QA pairs for future reference.

[LOG IN TO SAVE](#)
[DOWNLOAD JSON](#)

[← PREV](#)

● ● ●
Step 3 of 3

[START NEW →](#)

Figure 34. Download dataset as JSON

```

{
  "questions": [
    {
      "title": "Fastest time to go from Louvre Museum to Eiffel Tower by car?",
      "answer": {
        "type": "single-choice",
        "options": [
          "14 mins", "15 mins",
          "16 mins", "17 mins"
        ],
        "correct": 1
      },
      "classification": "routing",
      "relevant_api_calls": [ 98006 ]
    },
    {
      "title": "Nearby restaurant of Eiffel Tower with over 2000 user ratings and at least 4.5* rating",
      "answer": {
        "type": "single-choice",
        "options": [
          "Cafe Fleur", "Jules Verne",
          "Linette", "L'Abreuvoir"
        ],
        "correct": 3
      },
      "classification": "nearby",
      "relevant_api_calls": [ 12492 ]
    }
  ],
  "api_call_logs": [
    {
      "url": "https://routes.googleapis.com/directions/v2:computeRoutes",
      "method": "POST",
      "headers": { .... },
      "data": {
        "origin": { .... },
        "destination": { .... },
        "travelMode": "DRIVE",
        ....
      },
      "uuid": 98006,
      "result": { .... }
    },
    {
      "url": "https://places.googleapis.com/v1/places:searchText",
      "method": "POST",
      "headers": { .... },
      "data": {
        "locationBias": { .... },
        "includedType": "restaurant",
        ....
      },
      "uuid": 12492,
      "result": { .... }
    }
  ],
  { .... }
}

```


Figure 35. Example dataset where each QA pair is linked with relevant api calls

Footnotes

¹ <https://mapsplatform.google.com/>

² <https://www.apple.com/au/maps/>

³ <https://github.com/mapqator/>

⁴ <https://developers.google.com/maps/documentation/javascript/overview>

⁵ <https://developers.google.com/maps/documentation/utilities/polylinealgorithm>

References

- ¹ [Liu X, Yu H, Zhang H, Xu Y, Lei X, Lai H, Gu Y, Ding H, Men K, Yang K, et al. \(2023\). "Agentbench: Evaluating llms as agents". arXiv preprint arXiv:2308.03688.](#)
- ² [a, b, cQin Y, Liang S, Ye Y, Zhu K, Yan L, Lu Y, Lin Y, Cong X, Tang X, Qian B, et al. \(2023\). "Toolllm: Facilitating large language models to master 16000+ real-world apis". arXiv preprint arXiv:2307.16789.](#)
- ³ [LZhou S, Xu FF, Zhu H, Zhou X, Lo R, Sridhar A, Cheng X, Bisk Y, Fried D, Alon U, et al. \(2023\). "WebArena: A Realistic Web Environment for Building Autonomous Agents". arXiv preprint arXiv:2307.13854.](#)
- ⁴ [LKoh JY, Lo R, Jang L, Duvvur V, Lim MC, Huang PY, Neubig G, Zhou S, Salakhutdinov R, Fried D \(2024\). "VisualWebArena: Evaluating Multimodal Agents on Realistic Visual Web Tasks". arXiv preprint arXiv:2401.13649.](#)
- ⁵ [a, bXie J, Zhang K, Chen J, Zhu T, Lou R, Tian Y, Xiao Y, Su Y \(2024\). "Travelplanner: A benchmark for real-world planning with language agents". arXiv preprint arXiv:2402.01622.](#)
- ⁶ [LZheng HS, Mishra S, Zhang H, Chen X, Chen M, Nova A, Hou L, Cheng HT, Le QV, Chi EH, et al. \(2024\). "NATURAL PLAN: Benchmarking LLMs on Natural Language Planning". arXiv preprint arXiv:2406.04520.](#)
- ⁷ [LGuo Z, Cheng S, Wang H, Liang S, Qin Y, Li P, Liu Z, Sun M, Liu Y \(2024\). "StableToolBench: Towards Stable Large-Scale Benchmarking on Tool Learning of Large Language Models". arXiv preprint arXiv:2403.07714.](#)

8. [^]Choimeun S, Phumejaya N, Pomnakchim S, Chantrapornchai C. Tool for collecting spatial data with Google Maps API. In: *U-and E-Service, Science and Technology: International Conference UNESST 2010, Held as Part of the Future Generation Information Technology Conference, FGIT 2010, Jeju Island, Korea, December 13-15, 2010. Proceedings. Springer; 2010. p. 107-113.*
9. [^]Cai C, Hovy E (2010). "Summarizing textual information about locations in a geo-spatial information display system". In: *Proceedings of the NAACL HLT 2010 Demonstration Session. pp. 5-8.*
10. [^]Zheng Y, Bao Z, Shou L, Tung AKH (2014). "MESA: A map service to support fuzzy type-ahead search over geo-textual data". *Proceedings of the VLDB Endowment. 7 (13): 1545-1548.*
11. [^]Li M, Zhao Y, Yu B, Song F, Li H, Yu H, Li Z, Huang F, Li Y (2023). "Api-bank: A comprehensive benchmark for tool-augmented llms". *arXiv preprint arXiv:2304.08244.*

Declarations

Funding: No specific funding was received for this work.

Potential competing interests: No potential competing interests to declare.