MapQaTor: An Extensible Framework for Efficient Annotation of Map-Based QA Datasets

Mahir Labib Dihan[1], **Mohammed Eunus Ali**[1,2], **Md Rizwan Parvez**[3]
[1]Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology (BUET)
[2]Faculty of Information Technology,
Monash University, Melbourne, Australia
[3]Qatar Computing Research Institute (QCRI)
{mahirlabibdihan, mohammed.eunus.ali}@gmail.com, mparvez@hbku.edu.qa
https://mapqator.github.io/project/

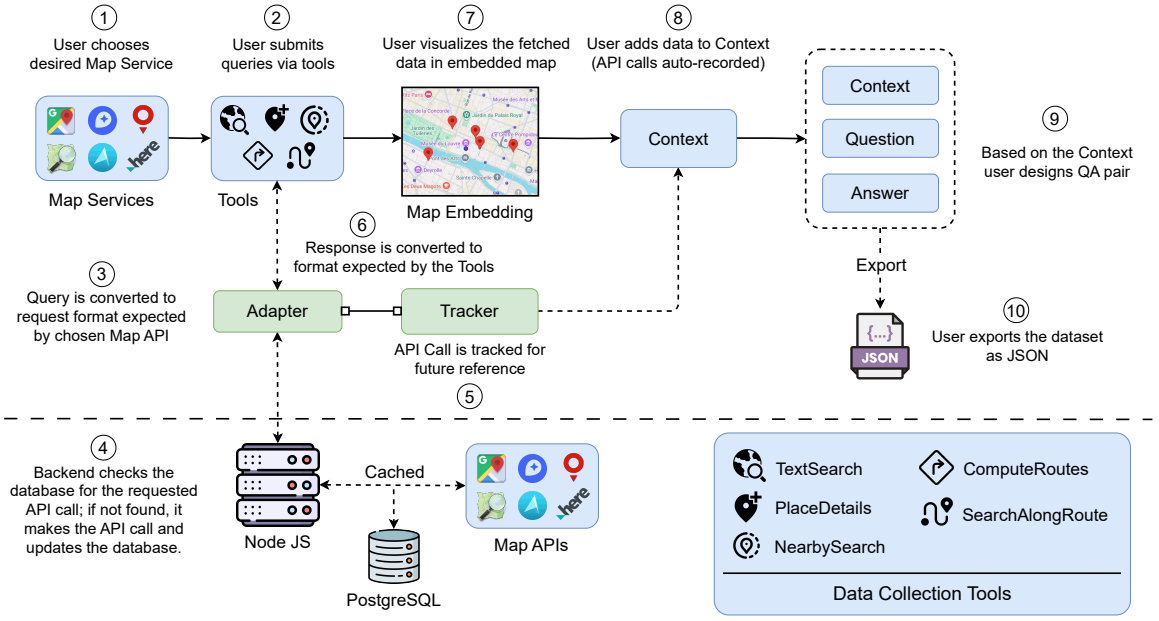

Figure 1: Overview of the annotation and visualization process of MapQaTor .

Abstract

Mapping and navigation services like Google Maps, Apple Maps, OpenStreetMap, 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, an extensible open-source framework that streamlines the creation of reproducible, traceable map-based QA datasets. 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/bVv7-NYRsTw.

.

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 Maps and Apple Maps have become essential tools, providing users with features like route planning, nearby points of interest (POIs), and contextual data, including reviews and oper-

Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations), pages 1–10

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

ating hours. However, while these services offer extensive geospatial data, they often struggle to understand and process 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 (Liu et al., 2024; Qin et al.). Notable tasks like WebArena (Zhou et al.) and VisualWebArena (Koh et al., 2024) 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, enabling them to gather, visualize, and annotate geospatial data from desired map API 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 its extensible architecture, enabling seamless integration with any map API (e.g., Google Maps, Apple Maps, OpenStreetMap).
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 useful-

ness and efficiency.

We have published the code on GitHub[1] 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, 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.

2.1 Context Designer

The core function of MapQaTor is to generate Context[2] using data collection tools, enabling structured and efficient QA pair creation.

2.1.1 Data Collection Tools

MapQaTor 's data collection framework (Figure 2) integrates five modular tools—Text Search, Place Details, Nearby Search, Compute Routes, and Search Along Route—to unify diverse map API functionalities under a standardized interface. Each tool follows a consistent design pattern:

- Inputs: User-defined parameters (e.g., location coordinates, filters, natural language queries).
- Outputs: Structured API responses (e.g., places, routes, metadata) normalized for downstream tasks.
- Context Integration: All inputs, raw API outputs, and processed data are stored as reusable Context, preserving traceability, and enabling QA generation.

The tools abstract API-specific complexities through configurable adapters while maintaining provider flexibility. Below, we outline their roles and workflows, with visual examples.

Text Search: Allows users to search for places by entering free-text queries (e.g., "Eiffel Tower" or "Starbucks near Central Park"). This tool leverages map API search capabilities to retrieve place names, addresses, and coordinates, making it efficient for locating points of interest (Figure 5).

Place Details: Fetches granular metadata (e.g., opening hours, accessibility) for a selected location (Figure 6). It resolves API schemas into unified fields, supporting factual queries like "Does the Louvre Museum offer wheelchair access?"

Nearby Search: Finds points of interest (POIs) near a location (Figure 7). Users can filter by price tiers, ratings, and ranking logic, enabling spatial QA pairs like "List nearby restaurants of Eiffel Tower with at least a 4 rating."

Compute Routes: Generates navigation paths between locations (Figure 8), supporting multi-stop optimization and travel mode selection (e.g., driving, walking), with step-by-step instructions and route metrics.

Search Along Route: Identifies POIs along a route (Figure 9). Users specify filters and route parameters, enriching trip-planning contexts like "Find gas stations along Highway 1 from San Francisco to Los Angeles."

2.1.2 Context Management

Each tool's execution appends a Context entry containing:

- Raw API Data: Original JSON responses for debugging and reproducibility.
- Normalized Fields: Extracted attributes (e.g., coordinates, ratings) in a unified schema.
- Metadata: Timestamps, API provider, and query parameters.

This layered organization ensures flexibility: raw data supports provider-specific analysis, while normalized fields streamline QA generation.

2.1.3 Impact on Reproducibility

The architecture guarantees that identical queries produce the same structured outputs, even if the underlying API changes. For example, a Nearby Search for "restaurants near Louvre Museum" returns normalized fields like `rating`, `price`, and `coordinates`, regardless of whether Google Maps or OpenStreetMap is used. This consistency is critical for long-term dataset validity.

[1] https://github.com/mapqator/

[2] Context refers to the data and information necessary to design a QA pair, ensuring that the answer to each question exists within the context.

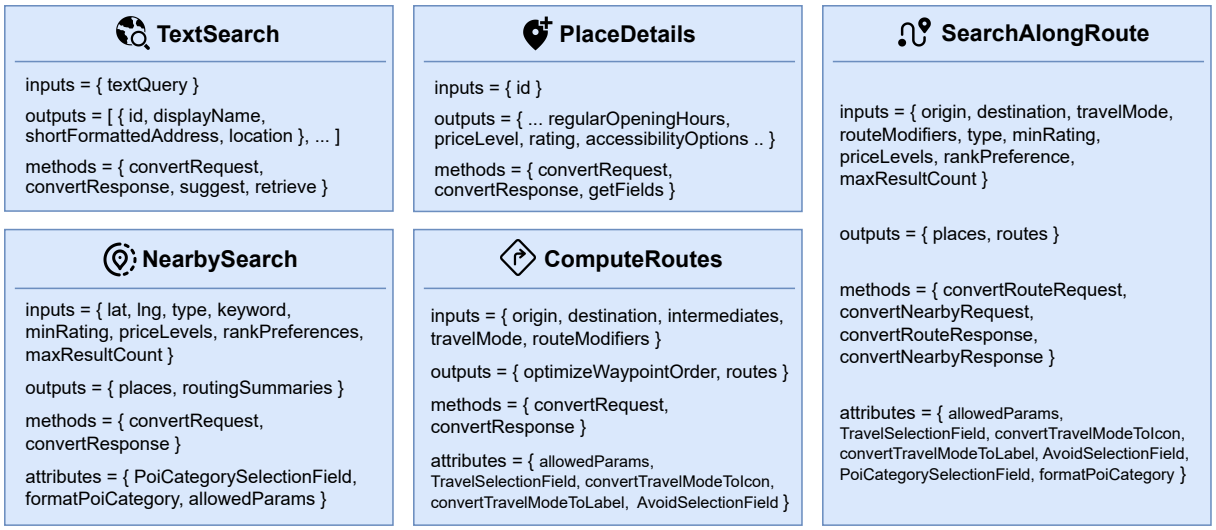

Figure 2: Standardized schema for data collection tools, unifying inputs, outputs, methods, and attributes.

2.1.4 Visualization Tools

For visualizing geospatial data, MapQaTor utilizes the Google Maps JavaScript API[3] to display places and routes directly on an embedded map. Users can view places as markers and visualize route paths (Figures 5–9). To render routes, MapQaTor decodes polyline-encoded data from map APIs into latitude-longitude coordinates using polyline decoding algorithm [4], ensuring accurate visualization of complex routes. These visualization tools help users understand spatial relationships, facilitating the creation of precise and context-aware map-based questions.

2.2 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 (Figure 3). It supports four answer formats: Yes/No, Single Choice, Multiple Choice, and Open Ended, allowing users to select the format that best suits their needs. 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 (Appendix E). The system also supports AI-assisted question generation, leveraging Gemini-2.0-Flash (DeepMind, 2025) with few-shot prompting to automatically generate sample question from context, further enhancing the annotation process. Once QA pairs are created, they can be evaluated using the Prompt Design Interface (see Appendix B). This interface allows users to structure prompts, compare model's responses against ground truth, and assess the performance.

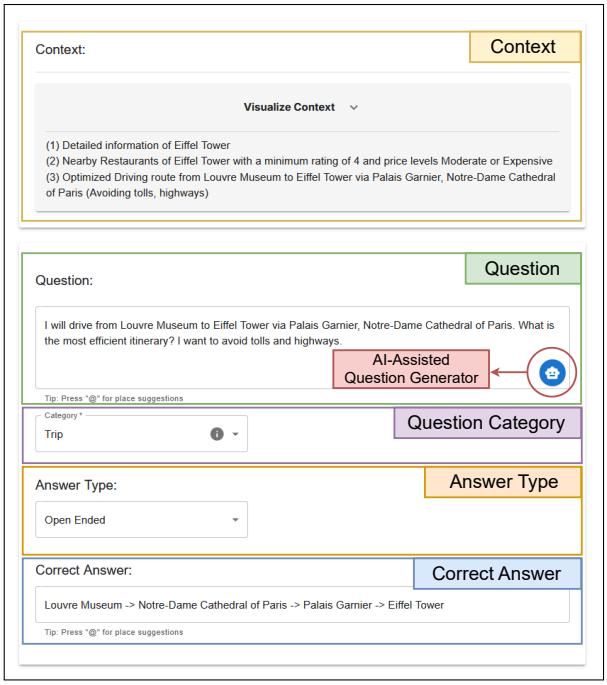

Figure 3: QA design and annotation interface.

2.3 Context Optimization

The structured context generated by MapQaTor's data collection tools is often large and complex, containing detailed raw data and numerous meta-

[3] https://developers.google.com/maps/documentation/javascript/overview

[4] https://developers.google.com/maps/documentation/routes/polylinedecoder

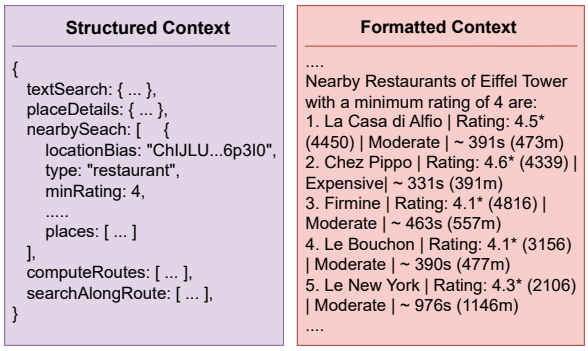

Figure 4: Comparison of structured and formatted context for improved readability and reduced size.

data elements. While this structure is necessary to ensure complete traceability and data accuracy, it can be cumbersome when used directly in downstream tasks. To address this challenge, we convert the structured context into a more formatted context, which is a more compact, human-readable version (See figure 4). This transformation retains the key information needed for evaluating LLMs for QA tasks, while eliminating unnecessary complexity. By simplifying the context, we significantly reduce token usage and improve processing efficiency, making it more suitable for large-scale evaluations and effective LLM-based analysis.

2.4 API Extensibility

New APIs can be integrated into MapQaTor by extending base tool classes (e.g., `NearbySearch`) and implementing abstract methods (e.g., `convertRequest`, `convertResponse`) as shown in Figure 12. Attributes like `PolCategorySelectionField` and `allowedParams` (Figure 2) handle provider-specific UI elements, such as point-of-interest (POI) categories, which vary across APIs (e.g., Google Maps vs. OpenStreetMap). To date, MapQaTor has integrated 20 APIs from 6 providers (Table 1), including both paid and free options. This modular design ensures adaptability to diverse map APIs while maintaining a consistent user experience.

2.5 Secure API Handling

MapQaTor 's backend securely mediates interactions between frontend tools (e.g., Nearby Search, Text Search) and third-party map APIs through two critical steps:

Tool-to-Backend Requests: As shown in Figure 12, frontend tools send API-agnostic requests containing credential placeholders (e.g., `key:TOMTOM_API_KEY`) and provider-specific parameters.

API Key Injection: The backend replaces placeholders with environment-stored credentials. Sensitive keys are never exposed in client-side code.

2.6 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, 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.

2.7 Application Scenarios

MapQaTor is primarily designed to support the creation of both training and evaluation datasets for geospatial question answering (QA), enabling the benchmarking (See Section 3.2) and improvement of large language models (LLMs) in geospatial reasoning tasks. In addition to evaluation, MapQaTor can be used to create high-quality training datasets for supervised fine-tuning (SFT) and alignment. Using MapQaTor's extensible architecture, users have the flexibility to evaluate the richness and capabilities of any available map services.

3 Experiments and Evaluation

3.1 Comparison with Manual Methods

We conducted a controlled experiment to quantify MapQaTor 's efficiency gains in geospatial data collection compared to manual methods. Two final-year undergraduate (BSc) students with Google Maps experience performed four geospatial tasks both manually and via MapQaTor. The results (Table 2) demonstrate a significant improvement in data retrieval speed, with MapQaTor requiring at least 30 times less time than the manual approach.

Task Definitions Four core geospatial operations were evaluated:

- **Place Details**: Retrieve name, address, rating, opening hours, reviews for the Louvre Museum

- **Nearby Search**: List 20 nearby restaurants of Louvre Museum, sorted by distance
- **Compute Routes**: Generate two alternative driving routes from Eiffel Tower to Louvre Museum
- **Search Along Route**: List 20 restaurants along the driving route from Eiffel Tower to Louvre Museum.

Manual Method

- Used Google Maps[5] web interface
- Copied data to spreadsheets with exact formatting
- Repeated 5 times per task per participant, with the median time recorded to mitigate outliers.

Automated Method

- Executed via MapQaTor 's Web Interface
- Used identical search parameters

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.

3.2 The MapEval Benchmark

To evaluate the annotation quality, we introduce MapEval (Dihan et al., 2025), a benchmark designed to evaluate LLMs on geospatial reasoning tasks. One of its evaluation settings, MapEval-Textual[6], assesses model performance by prompting LLMs with context and a question, then comparing their responses to the annotated ground truth. This evaluation used 300 MCQs annotated using MapQaTor to benchmark 19 LLMs (e.g., Claude-3.5-Sonnet, GPT-4o, Gemini-1.5-Pro). Preliminary results (Table 3) reveal significant gaps in model performance on complex spatial tasks, demonstrating the value of MapQaTor in generating high-quality datasets for benchmarking.

MapQaTor's caching mechanism was key in annotating the dataset within the Google Map API's free tier limit, while the visualization feature improved annotation accuracy and human evaluation. In MapEval-Textual, two human evaluators, who were not involved in the annotation process, answered the same 300 MCQs, achieving an average accuracy of 86.67%—more than 20% higher than the top-performing models (Table 3). This disparity is attributed to MapQaTor's context visualization feature (Section 2.1.4). While LLMs only had access to textual context, lacking visualization capabilities, humans were able to leverage the embedded map to interpret the spatial context.

Model	Accuracy (%)
Claude-3.5-Sonnet	66.33
Gemini-1.5-Pro	66.33
GPT-4o	63.33
Human (with MapQaTor)	86.67

Table 3: MapEval-Textual Performances

In MapEval-Textual, LLMs were prompted with `Formatted Context` (Section 2.3). Statistics for the 300 MCQs reveal that the average length of Structured Context is 17,534 characters, while the Formatted Context is just 2,536 characters—an 85.54% reduction. This not only demonstrates MapQaTor's space efficiency but also significantly lowers evaluation costs, as the cost is based on the number of tokens processed.

4 Related Works

Recent research has highlighted the potential of map data in mimicking real-world planning tasks through various tools (Xie et al., 2024; Zheng et al., 2024). Additionally, studies emphasize the significance of caching API call results to establish a stable database for evaluation purposes (Guo et al., 2024; Xie et al., 2024). 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 (Choimeun et al., 2010; Cai and Hovy, 2010; Zheng et al., 2014).

While tool-calling datasets like ToolBench (Qin et al.) and APIBank (Li et al., 2023) 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

[5] https://www.google.com/maps

[6] https://huggingface.co/datasets/MapEval/MapEval-Textual

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.

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, deprecations, or invalid API keys can 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.

References

Congxing Cai and Eduard Hovy. 2010. Summarizing textual information about locations in a geo-spatial information display system. In *Proceedings of the NAACL HLT 2010 Demonstration Session*, pages 5–8.

S Choimeun, N Phumejaya, S Pomnakchim, and Chantana Chantrapornchai. 2010. 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*, pages 107–113. Springer.

Google DeepMind. 2025. Gemini 2.0 flash. Accessed: 2025-03-13.

Mahir Labib Dihan, Md Tanvir Hassan, Md Tanvir Parvez, Md Hasebul Hasan, Md Almash Alam, Muhammad Aamir Cheema, Mohammed Eunus Ali, and Md Rizwan Parvez. 2025. Mapeval: A map-based evaluation of geo-spatial reasoning in foundation models. In *Forty-second International Conference on Machine Learning*.

Zhicheng Guo, Sijie Cheng, Hao Wang, Shihao Liang, Yujia Qin, Peng Li, Zhiyuan Liu, Maosong Sun, and Yang Liu. 2024. Stabletoolbench: Towards stable large-scale benchmarking on tool learning of large language models. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 11143–11156.

Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Russ Salakhutdinov, and Daniel Fried. 2024. Visualwebarena: Evaluating multimodal agents on realistic visual web tasks. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 881–905.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. 2023. Api-bank: A comprehensive benchmark for tool-augmented llms. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 3102–3116.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. 2024. Agentbench: Evaluating llms as agents. In *ICLR*.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. Toolllm: Facilitating large language models to master 16000+ real-world apis. In *The Twelfth International Conference on Learning Representations*.

Jian Xie, Kai Zhang, Jiangjie Chen, Tinghui Zhu, Renze Lou, Yuandong Tian, Yanghua Xiao, and Yu Su. 2024.

Travelplanner: A benchmark for real-world planning with language agents. In *International Conference on Machine Learning*, pages 54590–54613. PMLR.

Huaixiu Steven Zheng, Swaroop Mishra, Hugh Zhang, Xinyun Chen, Minmin Chen, Azade Nova, Le Hou, Heng-Tze Cheng, Quoc V Le, Ed H Chi, et al. 2024. Natural plan: Benchmarking llms on natural language planning. *CoRR*.

Yuxin Zheng, Zhifeng Bao, Lidan Shou, and Anthony KH Tung. 2014. Mesa: A map service to support fuzzy type-ahead search over geo-textual data. *Proceedings of the VLDB Endowment*, 7(13):1545–1548.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for building autonomous agents. In *The Twelfth International Conference on Learning Representations*.

A Data Collection Tools

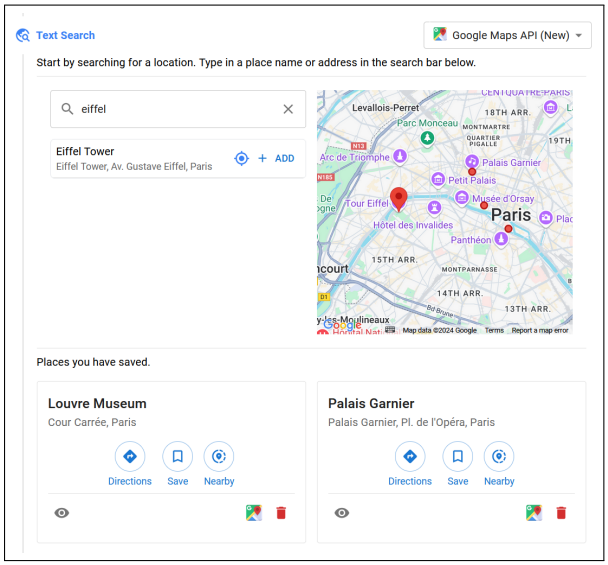

Figure 5: Search for a place

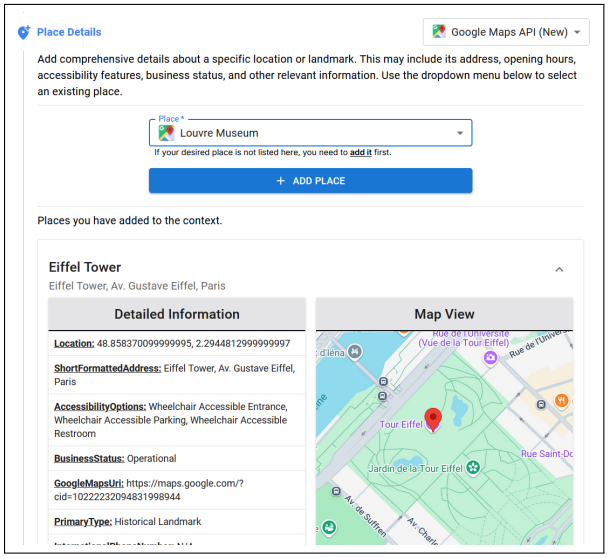

Figure 6: Fetch full details of a place

B Prompt Design Interface

The prompt design interface enables users to generate prompts for LLM evaluation by selecting a structured or formatted context. It displays the generated prompt, ground truth answers, and Gemini's response for comparison. Figure 10 illustrates this process.

C Exclusion of Temporal Variations in Routing APIs

To ensure reproducibility, MapQaTor removes temporal variations in routing by:

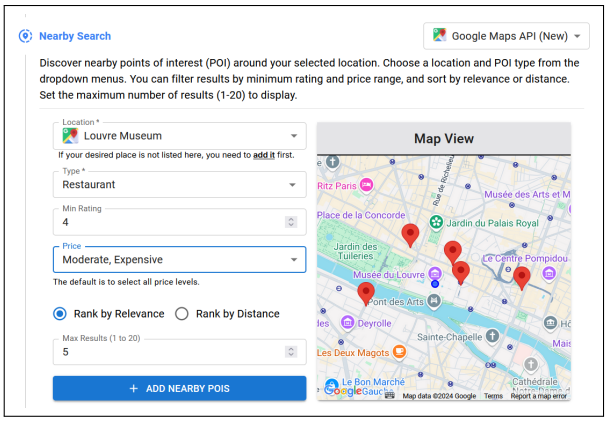

Figure 7: Search Nearby Places

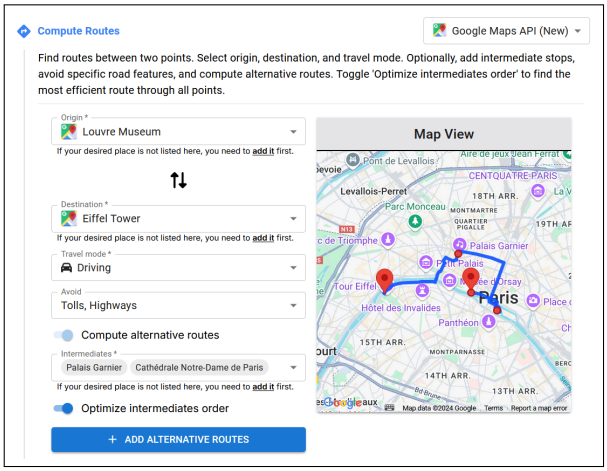

Figure 8: Find routes between places

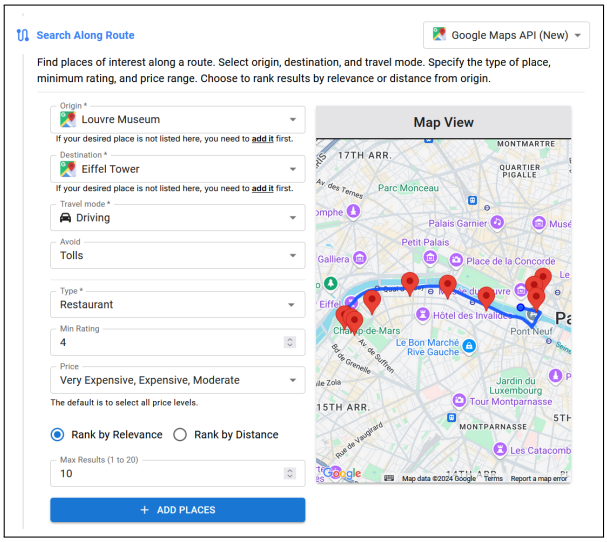

Figure 9: Search places along a route

Traffic Awareness Setting: Routing APIs are set to "TRAFFIC_UNAWARE," ensuring consistent travel times by ignoring real-time traffic.

Exclusion of Transit Mode: The "TRANSIT" mode is excluded to prevent variability from sched-

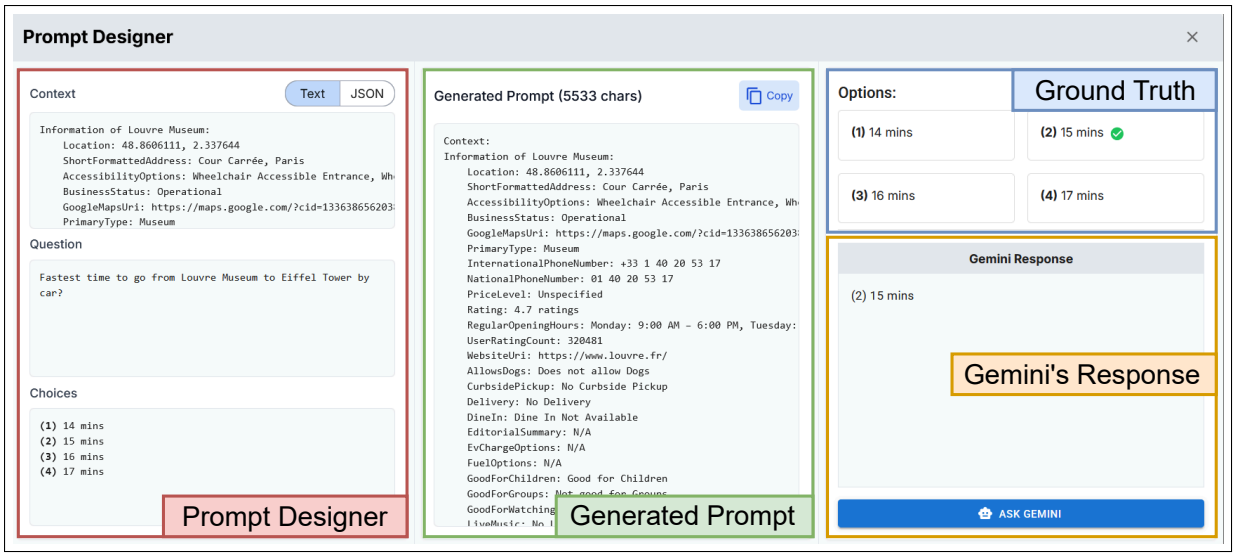

Figure 10: The figure illustrates prompt creation, ground truth comparison, and Gemini's response assessment.

ule changes.

Benefits:

- Ensures consistent responses for identical queries.
- Focuses evaluations on spatial reasoning, not real-time changes.
- Provides a stable baseline for model benchmarking.

These measures enable reliable and reproducible geospatial evaluations in MapQaTor .

D API Extension Mechanism

Figure 12 demonstrates how new map services are integrated by extending MapQaTor 's core tools:

E Place Name Suggestion

Using the TextSearch tool, annotators can retrieve place names. While writing a question or answer, pressing '@' suggests available place names, ensuring consistency between context and QA pairs.

Figure 11: Suggesting available places from the context.

```
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 12: Extending Text Search for TomTom API