International Journal of Geo-Information

MDPI

*Article*

# Large Language Model-Driven Structured Output: A Comprehensive Benchmark and Spatial Data Generation Framework

**Diya Li [1,2], Yue Zhao [1], Zhifang Wang [1], Calvin Jung [1] and Zhe Zhang [2,*]**

[1] Survey123, Esri, Redlands, CA 92374, USA; diya.li@tamu.edu (D.L.)
[2] Department of Geography, Texas A&M University, College Station, TX 77840, USA
* Correspondence: zhezhang@tamu.edu

**Abstract:** Large language models (LLMs) have demonstrated remarkable capabilities in document processing, data analysis, and code generation. However, the generation of spatial information in a structured and unified format remains a challenge, limiting their integration into production environments. In this paper, we introduce a benchmark for generating structured and formatted spatial outputs from LLMs with a focus on enhancing spatial information generation. We present a multi-step workflow designed to improve the accuracy and efficiency of spatial data generation. The steps include generating spatial data (e.g., GeoJSON) and implementing a novel method for indexing R-tree structures. In addition, we explore and compare a series of methods commonly used by developers and researchers to enable LLMs to produce structured outputs, including fine-tuning, prompt engineering, and retrieval-augmented generation (RAG). We propose new metrics and datasets along with a new method for evaluating the quality and consistency of these outputs. Our findings offer valuable insights into the strengths and limitations of each approach, guiding practitioners in selecting the most suitable method for their specific use cases. This work advances the field of LLM-based structured spatial data output generation and supports the seamless integration of LLMs into real-world applications.

**Keywords:** spaital R-tree; large language model; generative pretrained transformer; GeoJSON; structured data



**Citation:** Li, D.; Zhao, Y.; Wang, Z.; Jung, C.; Zhang, Z. Large Language Model-Driven Structured Output: A Comprehensive Benchmark and Spatial Data Generation Framework. *ISPRS Int. J. Geo-Inf.* **2024**, *13*, 405. https://doi.org/10.3390/ijgi13110405

Academic Editors: Wolfgang Kainz, Hartwig H. Hochmair, Levente Juhász and Hao Li

Received: 1 September 2024
Revised: 5 November 2024
Accepted: 7 November 2024
Published: 10 November 2024



## 1. Introduction

Recent advancements in artificial intelligence and natural language processing (NLP) have paved the way for the development of large language models (LLMs). Models such as the generative pretrained transformer (GPT) series [1] have demonstrated remarkable performance across a wide range of tasks and domains, including medical assistance [2], financial analysis [3], and geospatial tasks [4–6], often achieving near-human performance. As research into LLMs progresses, LLM agents are gaining popularity, particularly in industrial applications such as LangChain [7] and Autogen [8]. An LLM agent is an autonomous system powered by a large language model that can perform complex tasks, interact with APIs, and make decisions based on user prompts or real-time data [8]. However, it is important to recognize the limitations of these agents and tackle the challenges associated with enriching machine-readable format data with free-form language output to enhance knowledge discovery, such as through GeoJSON. Addressing these challenges will enable developers and researchers to seamlessly integrate LLM techniques into their existing systems, thereby unlocking the full potential of these powerful models. The motivation for using LLMs in this context stems from their ability to simplify spatial data handling, enabling non-specialists to engage with geospatial data formats such as GeoJSON without needing advanced knowledge of professional tools. This presents an opportunity to democratize access to spatial data generation and manipulation through natural language interfaces.

Structured output has been widely used across many fields, including HTTP responses, algorithm generation, and even spatial query via SQL [9]. Spatial structured output such as GeoJSON is of great significance in the field of Geographic Information Systems (GIS). GeoJSON is a widely used format for encoding geographic data structures that enables efficient storage, sharing, and analysis of spatial information. For example, consider an emergency response application that needs to provide accurate location-based information during a natural disaster such as a flood. Generating precise GeoJSON data allows emergency services to map affected areas, identify critical infrastructure at risk, and communicate this information effectively to stakeholders in real time. In such scenarios, generating spatially structured geospatial outputs can significantly improve response times and decision-making processes. However, LLMs typically do not generate structured formats such as JSON or XML without explicit instructions. Current solutions for guiding LLMs towards generating structured JSON can be categorized into three main approaches: fine-tuning [10], retrieval-augmented generation (RAG) [11], and prompt engineering [12]. RAG and prompt engineering are two distinct approaches used to optimize large language models (LLMs) for generating structured outputs. Fine-tuning a large model such as GPT-4 is a complex and resource-intensive process. For more specific task-oriented LLMs [13], smaller-sized models such as TinyLLM [14] are often employed due to their efficiency. However, there is currently a lack of comprehensive comparison between these methods. RAG integrates external knowledge retrieval into the generation process, which has demonstrated strong results in producing more accurate and context-aware responses; however, this approach demands additional computational resources and can introduce irrelevant information, potentially lowering the overall quality of the output [11,15]. In contrast, prompt engineering involves crafting precise input prompts to steer the model towards generating the desired format or outcome. While this method can be effective, it requires significant manual effort and may not always guarantee consistent results [12].

To address these limitations and further improve the generation of structured outputs, particularly in the context of geospatial data, we propose a novel hybrid approach that leverages the strengths of existing methods while incorporating spatial indexing techniques. This new method, which we term the "R-tree enhanced LLM" (REL) approach, combines fine-tuning, RAG, and spatial indexing using R-trees to significantly improve the generation of GeoJSON outputs. The REL method builds upon the foundation of fine-tuned LLMs, which are adapted to understand and generate geospatial data structures. It then incorporates an RAG component to retrieve relevant geospatial information and context, thereby enhancing the model's ability to generate accurate and contextually appropriate GeoJSON outputs. The key innovation of our approach lies in the integration of R-tree spatial indexing [16], a widely used technique in GIS for efficient spatial querying and data organization.

By leveraging R-trees, our method can efficiently index and query spatial data, allowing the LLM to better understand and utilize the hierarchical and spatial relationships inherent in geospatial information. This spatial awareness enables the model to generate more coherent and spatially consistent GeoJSON structures, improving the overall quality and usefulness of the output for geospatial applications.

In this paper, we make several contributions to the understanding of using LLMs to enrich structured format data in order to facilitate knowledge discovery. These contributions are as follows:

- We investigate the impact of retrieval techniques and fine-tuning on the performance of LLMs for the structured spatial data output generation task.
- We conduct an extensive evaluation of using LLMs to generate structured spatial data output content by creating a new metric and a benchmark with new datasets.
- Develop a new REL method incorporating spatial R-tree search techniques to enhance the accuracy and efficiency of spatial data generation by LLMs.

## 2. Background

The integration of LLMs has the potential to revolutionize traditional software development workflows. LLMs can be leveraged to automate various aspects of the development process, including code generation [17], database schema design [18], and API documentation creation [19]. This automation not only streamlines the development process but also significantly reduces the need for manual coding, potentially increasing efficiency and productivity. Recent research [20] has demonstrated the use of structured data to generate descriptions and summaries, while other studies [9] have employed structured metadata for open question answering using LLMs.

Structured output generation refers to the process of producing content in a format that follows specific rules or templates, resulting in well-organized data. A notable example of this is the generation of GeoJSON [21]. With the development of LLM techniques, structured output generation tasks in LLMs have been increasingly adopted in both industry [22] and academic projects [23]. However, generating well-formatted and consistent JSON outputs remains a significant challenge, especially when handling complex nested structures and a mix of free-form and structured content [24]. Several approaches have been proposed to leverage LLMs for structured output generation. For instance, Escarda-Fernández et al. [25] introduced a text-to-JSON method that generates JSON through supervised fine-tuning, while Beurer-Kellner et al. [26] explored a noninvasive constrained generation method for free-form JSON text generation. Another recent study on structured output generation focused on optimizing schema discovery [27]. Moreover, the generation of spatial data through LLMs remains an underexplored area in geospatial research, with only a few studies addressing this topic directly [28]. While researchers often leverage LLM agents to search and visualize GeoJSON data from spatial databases [29], the process of creating or manipulating GeoJSON files typically requires specialized geospatial tools such as GeoPandas [30] or ArcGIS [31]. While powerful, these tools demand a certain level of expertise and may not be accessible to non-experts. Although recent research has indicated the potential of LLMs to assist in retrieving spatial data and completing formats such as GeoJSON [32], there remains a gap in the literature when it comes to fully generating and manipulating GeoJSON data in a way that is directly compatible with geospatial standards.

Despite these advancements, research specifically focusing on spatial structured output generation and the evaluation of structured schemes remains limited.

Another recent work by Musumeci et al. [33] explored the use of semantic templates to generate semi-structured documents with multiple agents in the public administration domain. Similarly, Beurer-Kellner et al. [26] proposed a constrained generation approach that incorporates regex-based text generation and parses the result into JSON format. This constrained generation method ensures that the generated output adheres to a predefined structure and can be easily converted into a machine-readable format.

Benchmarking and evaluating LLM-generated structured data is another important area worth discussing. Recent studies have reevaluated the significance of structured data generation and recognized its crucial role in various tasks [34]. In [35], the authors introduced a jump operator to accelerate structured output generation in order to provide detailed evaluation methods suitable for production environments. This addresses the need for LLMs to be integrated into low-latency industry-level applications. Another study on benchmarking retriever-augmented generation used ability-oriented methods [34]. However, their research did not thoroughly address the accuracy of generated spatial data, leaving room for further investigation.

While these recent studies highlight the growing interest in developing techniques for generating accurate and well-formatted structured outputs using LLMs, several research gaps remain. First, there is a need for methods that can generate structured spatial data outputs quickly while maintaining high levels of accuracy. Second, many real-world applications require the generation of deeply nested structured outputs. Developing LLM-based techniques that can handle such complexity without compromising coherence or

accuracy is an ongoing challenge [36]. Third, developing robust methods for validating the generated structured outputs and automatically correcting errors or inconsistencies remains an important area for further research [37].

## 3. Dataset

Our research methodology started with meticulous dataset preparation, particularly emphasizing structured output generation to ensure accurate JSON data validation. The preparation process included the following steps:

- Data Collection: We utilized the web scraping tool Beautiful Soup in conjunction with Python's native `urllib` library to send HTTP requests and retrieve HTML content from various websites. Beautiful Soup was then employed to parse, format, and extract the relevant data fields. The sources for the dataset are documented in the General Structured Dataset Section.
- Data Sampling: After the raw data was collected, we performed a random sampling process to ensure that both the training and testing datasets were representative of the overall data distribution. This was accomplished by splitting the dataset into an 80% training set and a 20% testing set. Random sampling was conducted using a reproducible seed to maintain consistency in future experiments.
- Data Encoding: To maintain compatibility across different systems and prevent potential encoding errors, we standardized the encoding of all dataset files to UTF-8. Ensuring uniform UTF-8 encoding also helped in avoiding issues related to non-standard or legacy character encodings.
- Data Cleaning: During the dataset collection process, it was observed that some sources contained uncommon characters, symbols, or formatting anomalies. We systematically cleaned the data by removing these irrelevant elements.
- Data Validation: Following data cleaning, we conducted a series of validation checks to ensure the integrity and correctness of the collected data. These checks included verifying proper JSON formatting and ensuring that all required fields were present and well-formed.

### *3.1. General Structured Dataset*

We collected and curated three datasets from various sources, which were preprocessed and reorganized to suit the task at hand. The following datasets were used in our experiments:

- Schema Dataset: This dataset is similar to a schema parsing task, but mainly focuses on converting free-form web language to a JSON-LD format that better suits web semantic construction. The original dataset was manually created by Schema.org [38] to provide examples that demonstrate the correct way to format web data. We further processed and cleaned the dataset to align with our task requirements.
- Nous Dataset: This dataset involves JSON schema parsing; given a formatted JSON schema, the task is to format a piece of text to fill in the schema. We collected and cleaned the data from the Nous JSON dataset [39] to create a refined version suitable for our experiments.
- Paraloq Dataset: This dataset focuses on regular JSON information extraction, asking the model to extract key information and output a JSON based on specific requirements. We re-organized and collected data from the original Paraloq JSON Eval dataset [40] to build a dataset tailored to our needs.

The datasets were collected using web scrapers. An overview of all the datasets is shown in Table 1.

**Table 1.** Overview of the datasets used for structured output generation

| Dataset | Num Examples | Num Tokens [1] | Max Token | Complexity [2] |
|---|---|---|---|---|
| Schema Dataset | 485 | 129,827 | 2008 | High |
| Nous Dataset | 484 | 567,383 | 2946 | Low |
| Paraloq Dataset | 100 | 30,570 | 670 | Medium |
| Coleridge Initiative | 195 | 1,823,894 | 11,626 | High |

[1] Max Token refers to the maximum number of tokens per structured output (e.g., JSON or GeoJSON). In LLMs, a token is a unit of text, such as a word or part of a word. [2] Complexity refers to the dataset's structural diversity, indicating whether it contains nested structures or is uniform.

### *3.2. Spatial Structured Dataset*

The spatial structured dataset was primarily built using GeoJSON, with the study area mainly covering the United States. This dataset consists of two parts:

- Extracted Spatial Dataset Descriptions: These were derived from the Coleridge Initiative Dataset (https://www.kaggle.com/competitions/coleridgeinitiative-show-us-the-data/overview, accessed on 6 November 2024), which comprises 14,300 publications across various fields of study. Using LLMs, we extracted 195 spatial-related dataset descriptions. An example prompt and response can be found in Appendix A Table A5.
- GeoJSON Mapping: Each of the extracted descriptions was matched with a GeoJSON representation at the US county level.

## 4. Methodology

The methodology proposed in this paper revolves around a pipeline designed to generate and evaluate question–answer pairs for general structured output generation and spatial data generation. In this section, we introduce the general methods used for structured output generation, which include fine-tuning, RAG, and prompt engineering. The overall workflow is illustrated in Figure 1. In this end-to-end workflow, we first collect and preprocess the dataset. Using this dataset, we experiment with different methods, applying the same metrics to evaluate the results.

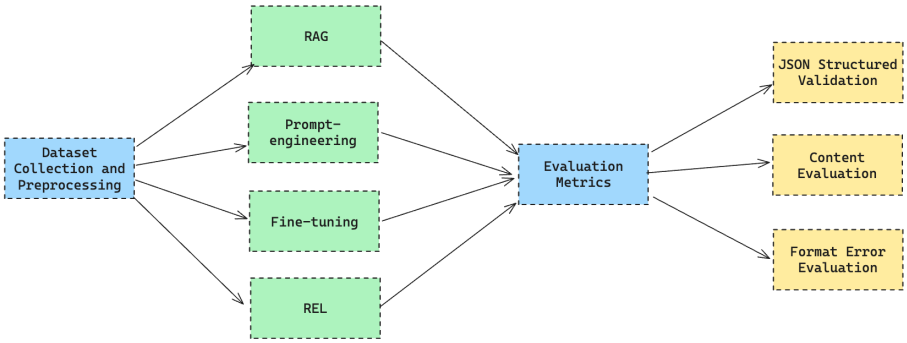


**Figure 1.** The overall workflow of our proposed methods.

### *4.1. Fine-Tuning*

Fine-tuning is a crucial step in adapting LLMs to specific tasks. By further training a pretrained model on a task-specific dataset, the model learns to capture the nuances and patterns relevant to the target task. However, fine-tuning large models can be computationally expensive and memory-intensive. To address these challenges, we employ the low-rank adaptation (LoRA) technique [10] in combination with 4-bit quantization using the GPTQ Algorithms [41]. LoRA is an efficient fine-tuning method that adds low-rank matrices to the model's weights, significantly reducing the number of trainable parameters. By training only the low-rank matrices and keeping the pretrained weights frozen, LoRA

enables faster and more memory-efficient fine-tuning. Specifically, the transformer-based models such as Llama use a self-attention mechanism that allows the model to weight the influence of different tokens in the input sequence when generating each part of the output by three key matrices: query matrix ($Q$), key matrix ($K$), and value matrix ($V$). Instead of updating $W_Q$ and $W_V$ directly, we introduce low-rank matrices $A$ and $B$ to represent the weight updates:

For $W_Q$:

$$W'_Q = W_Q + \Delta W_Q, \quad \Delta W_Q = A_Q B_Q \tag{1}$$

For $W_V$:

$$W'_V = W_V + \Delta W_V, \quad \Delta W_V = A_V B_V \tag{2}$$

where $A_Q$ and $A_V$ are of shape $(d_{\text{model}}, r)$, $B_Q$ and $B_V$ are of shape $(r, d_k)$, and $r$ is the rank (a small integer with $r \ll d_{\text{model}}$). To ensure that the model generates outputs in JSON format, a task-specific loss function measures the discrepancy between the model's output and the target JSON structure. We use the cross-entropy loss over the tokenized JSON outputs:

$$\mathcal{L}_{\text{JSON}} = -\sum_{t=1}^{T} \log p_\theta(y_t | y_{<t}, x) \tag{3}$$

where $T$ is the length of the target JSON output, $y_t$ is the target token at position $t$, and $p_\theta(y_t|y_{<t}, x)$ is the probability of the token $y_t$ given the previous tokens $y_{<t}$ and input $x$, parameterized by the model parameters $\theta$ (which include the LoRA adaptations).

Furthermore, quantization techniques compress the model's weights and activation into a lower-precision representation, such as four bits per parameter, while maintaining model performance. This quantization allows us to fine-tune larger models with limited computational resources.

All of our models use a supervised fine-tuning method with instruction data preprocessing. Supervised fine-tuning allows us to leverage labeled data to guide the model's learning process and ensure that it captures the desired task-specific patterns. By providing the model with input–output pairs subjected to instructional preprocessing, we can directly optimize the model's performance on the target task. We unified all training processes using the [42] supervised fine-tuning trainer for better reimplementation and adaptation.

The models we included in these steps were gemma-2b [43], gemma-7b [43], phi-2 [44], codellama2-7b [45], mistral-7b [46], and llama3-7b [47].

### *4.2. Retrieval-Augmented Generation*

Another technique commonly used for structured GeoJSON output is retrieval-augmented generation (RAG), which enhances the performance of language models by incorporating external knowledge during the generation process. RAG typically consists of a retriever component that searches for relevant information from a large corpus and a generator component that uses the retrieved information to guide the output generation. We use the widely-used FAISS [48] as a vector store to store the example pairs (default: 10) and allow the retriever to access this information for reference, thereby enhancing the overall generation quality.

### *4.3. Prompt Engineering*

Prompt engineering (PE) is the process of designing and optimizing input prompts to guide language models towards generating desired outputs. In the context of structured output generation, prompt engineering plays a crucial role in influencing the LLM to produce well-formatted and consistent JSON or XML outputs. To engineer effective prompts, we complete several steps: (1) explicit instructions clearly define the task and

expected output format; (2) representative examples of well-formatted structured outputs are provided to serve as templates that the model learns from and emulates; (3) JSON or GeoJSON schemas are incorporated to guide the model towards generating outputs that conform to the specified structure and constraints; (4) clear separators or delimiters are used to differentiate between the input text and the expected output format. An example showing how prompts are constructed is provided in Appendix A Table A6.

### *4.4. R-Tree Enhanced Large Language Model (Rel)*

R-trees are a type of spatial indexing structure used to efficiently store and query multidimensional data such as geographical coordinates or geometric shapes. An R-tree organizes data into a hierarchical tree-like structure in which each node represents a bounding box that encloses its child nodes or data points. These bounding boxes may overlap, allowing R-trees to handle complex and irregularly shaped spatial data. When querying, the R-tree quickly narrows down the search space by traversing nodes that intersect the query region, making it highly efficient for spatial queries such as range searches and nearest neighbor searches. In this study, we integrate R-tree spatial indexing with the RAG method to enhance the efficiency and relevance of information retrieval in LLMs. The R-tree bounding boxes allow for rapid narrowing of the search space during spatial queries. By linking spatial data in the R-tree to contextual embeddings generated by the LLM, we create a hybrid retrieval system that first filters data based on spatial proximity, then refines the results using semantic similarity. This approach enables the generation of contextually enriched responses that are both spatially aware and contextually relevant, thereby optimizing the performance of LLMs in location-based or spatially sensitive applications.

Figure 2 demonstrates the concise framework of LLM generation based on R-trees. Using the collected GeoJSON county-level data, which include the data description, we use the administration ID as the key to create a central point set. To break this down further, let $O = \{o_1, o_2, ..., o_n\}$ be a set of spatial objects in a d-dimensional space (e.g., US counties GeoJSON). Each object is identified with AID. For each object $o_i$, its minimum bounding rectangle (MBR) is defined as $MBR(o_i) = [l_1, u_1] \times [l_2, u_2] \times ... \times [l_d, u_d]$, where $l_j$ and $u_j$ are the lower and upper bounds in the j-th dimension. The MBR is calculated as the central point sets using AID for the RAG indexes of the LLM. Then, each node $N$ in the R-tree contains entries of the form $(I, ptr)$, where:

- $I$ is the MBR containing all objects in the subtree rooted at that node.
- $ptr$ is a pointer that indicates the child node.

After setting this up, we can let $M$ be the maximum number of entries in a node and $m \leq \lfloor M/2 \rfloor$ be the minimum number of entries in a node. Thus, for any non-leaf node with $k$ entries, we have $m \leq k \leq M$, while for the root we have $2 \leq k \leq M$ unless it is a leaf. Therefore, the area of the MBR is calculated as

$$I = [l_1, u_1] \times [l_2, u_2] \times ... \times [l_d, u_d], \tag{4}$$

$$Area(I) = \prod_{i=1}^{d} (u_i - l_i). \tag{5}$$

To calculate the overlap between two MBRs $I_1$ and $I_2$, we can use

$$Overlap(I_1, I_2) = \prod_{i=1}^{d} \max(0, \min(u_{1i}, u_{2i}) - \max(l_{1i}, l_{2i})). \tag{6}$$

This is set as a tool agent in the LLM framework. When a node overflows, it is split into two nodes. The goal is to minimize the total area $\min(\sum_{i=1}^{2} Area(I_i))$, where $I_1$ and $I_2$ are the MBRs of the two new nodes. In the end, for a query rectangle $Q$, we traverse the tree and perform a check at each node to determine whether $Overlap(I, Q) > 0$.

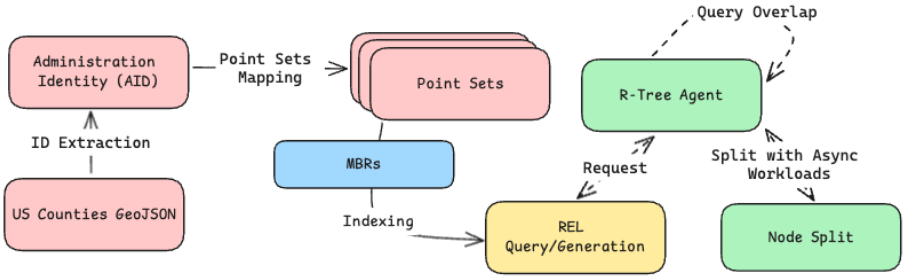


**Figure 2.** Framework of enhanced LLM generation based on R-trees.

## 5. Evaluation Metrics

Defining the metrics for evaluating the quality of structured generation using LLMs is a challenging task due to the inherent complexity of nested structures and the mixed free-and-structured format. It is sometimes not possible to validate the generated JSON format cannot because an unclosed symbol is missed in the middle of the JSON string; however, LLM and human evaluation often ignores this. To overcome these challenges, researchers often use a combination of evaluation methods, including BLEU scores [49] and ROUGE scores [50]. Moreover, there exists an open and complex problem in the field of free-form language-to-structured JSON generation concerning the challenge of defining a comprehensive metric. In this section, we introduce a combination of metric scores that includes a JSON structure validation score that can handle the JSON formatting schema and the content similarity between the correct JSON result and the generated result. The series of metrics involves using an LLM to substitute for human evaluation during the experimentation period and also considers the key criteria for the generated content, including completeness, accuracy, granularity, and consistency.

### *5.1. JSON Structured Validation Method*

The JSON structured validation method, also called structural validity [24], is a technique for comparing and measuring the similarity between two JSON objects. It takes into account both the structural similarity of the JSON objects and the textual similarity of their key–value pairs. The method calculates an overall similarity score between the two JSON objects, providing a quantitative measure of their resemblance. The key steps involved in the JSON structured validation method are as follows:

1. Key–Value Pair Extraction: The first step is to extract all the key–value pairs from each JSON object. This is accomplished by recursively traversing the JSON objects and flattening any nested dictionaries. The keys are concatenated using a temporary (e.g., dot ".") separator to preserve the hierarchical structure of the JSON objects. The extracted key–value pairs are stored as a set of tuples, where each tuple represents a unique key–value combination.
2. Common Pair Identification: The next step is to identify the common key–value pairs between the two JSON/sub-JSON objects. The common pairs are key–value pairs that are identical (key, value, and level) in both JSON objects. This is achieved by taking the intersection of the sets of extracted key–value pairs from both JSON objects. The common pairs represent the exact matches between the two JSON objects and are assigned a initial similarity score of (default 1.0) for both the key and value.
3. Similarity Scoring for Non-Common Pairs: For each non-common key–value pair in which two JSON/sub-JSON objects are not exact matches, the method finds the most similar pair between the first and second JSON objects. Similarity is determined using the Levenshtein distance, which measures the minimum number of single-character edits required to transform one string into another. The Levenshtein distance [51] is calculated separately for both the keys and values of the non-common pairs. The maximum similarity score is kept for each non-common pair, considering both the key

similarity and value similarity. This process is then repeated for each non-common pair in the second JSON object to find the most similar pair in the first JSON object.

4. Similarity Score Calculation: After obtaining the similarity scores for all pairs (common and non-common), the method calculates the average key similarity score and average value similarity score. The average scores are computed by summing up the individual similarity scores and dividing by the total number of pairs. The overall similarity score between the two JSON objects is then calculated as a weighted sum of the average key similarity score and average value similarity score. The weights for the key and value similarity scores can be adjusted based on the specific requirements of the validation task.

The JSON structured validation method draws inspiration from the Jaccard index [52], adapting its principles to develop a novel approach for validating structured data. The Jaccard index, which is a commonly used measure of similarity between sets, calculates the similarity between two sets by dividing the size of their intersection by the size of their union. In the context of JSON structured validation, the method considers not only the exact matches (intersection) but also the similarity of non-common pairs based on the Levenshtein distance. The calculation equation that summarizes the overall steps is as follows:

$$JScore(json1, json2) = w_k \cdot \frac{\sum_{p \in P} \max_{q \in Q} sim_k(p,q)}{|P| + |Q| - |P \cap Q|} + w_v \cdot \frac{\sum_{p \in P} \max_{q \in Q} sim_v(p,q)}{|P| + |Q| - |P \cap Q|} \quad (7)$$

where:

- $P$ and $Q$ are the sets of key–value pairs extracted from $json1$ and $json2$, respectively.
- $|P|$ and $|Q|$ denote the cardinalities (sizes) of the sets $P$ and $Q$.
- $|P \cap Q|$ represents the number of common key–value pairs between $P$ and $Q$.
- $sim_k(p,q)$ and $sim_v(p,q)$ are the similarity score between the keys and values of pairs $p$ and $q$, calculated using the Levenshtein distance.
- $w_k$ and $w_v$ are the weights assigned to the key and value similarity scores, respectively, with $w_k + w_v = 1$.

By incorporating both structural and textual similarity, the $JScore$ provides a comprehensive evaluation of the similarity between JSON objects. It takes into account the presence of common key–value pairs as well as the similarity of non-common pairs, allowing for a more nuanced comparison.

When validating GeoJSON objects, which are specialized JSON structures used to represent geographical data, the JSON structured validation method incorporates spatial key consideration to enhance the accuracy of similarity scoring. The variant of the Jscore equation used for validating GeoJSON is as follows:

$$JScore_g = \frac{\sum\limits_{(k_p,v_p) \in P} \max\limits_{(k_q,v_q) \in Q} \left[ w_k \cdot sim_k(k_p,k_q) + w_v \cdot sim_v(v_p,v_q) + w_s \cdot sim_s(v_p,v_q) \right]}{|P| + |Q| - |P \cap Q|} \quad (8)$$

where:

- $P$ and $Q$ are the sets of key–value pairs extracted from $geojson1$ and $geojson2$, respectively, with keys representing the full hierarchical and nested path.
- $|P|$ and $|Q|$ are the sizes of $P$ and $Q$.
- $|P \cap Q|$ represents the number of common key–value pairs.
- $w_k$, $w_v$, and $w_s$ are weights assigned to key similarity, value similarity, and spatial similarity, respectively ($w_k + w_v + w_s = 1$).

By considering the longest common prefix of the key paths, we account for the structural similarity in the nested hierarchy. When values are nested structures, the similarity is determined by recursively computing $JScore$ on these sub-objects in order to fully capture the nested similarities.

In addition to the standard key–value pair extraction and comparison processes, the method specifically identifies and evaluates spatial keys, including "coordinates", "geometry", "properties", "type", and "features". These spatial keys often contain complex nested arrays representing latitude and longitude coordinates. For these keys, the validation method extends the similarity scoring by incorporating spatial similarity measures, including calculating the Euclidean distance between coordinate sets or using other geospatial metrics to assess the proximity of locations.

$$\text{Distance}(A, B) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{9}$$

$$sim_s = 1 - \left( \frac{\text{Distance}(A, B)}{\text{MaxDistance}} \right) \tag{10}$$

By integrating spatial analysis with the textual similarity of nonspatial keys, this approach ensures that the validation of GeoJSON objects not only considers the structural and textual aspects but also accurately reflects the geographical relationships between the spatial data points. This extension of the structured validation method makes it well suited for applications involving geospatial data, in which both content and location are critical for accurate comparison.

### *5.2. Content Evaluation Method*

Evaluating structured spatial answers generated by LLMs is challenging due to their inherent complexity, as they include nested structures, mixed free-and-structured formats, and potential formatting errors. Similar works that have adopted LLMs to substitute for human evaluation have shown excellent results for complex tasks [53]. This method was later verified and proved to be efficient, with 45 times lower costs than crowd-workers, as demonstrated by [54]. In this section, we introduce a series of prompt methods to enable LLMs (GPT-4o) to rate the content on a scale of 1 to 5, with 5 being the best quality of generation and 1 being generation that is not satisfactory. The chosen list of criteria included completeness, accuracy, granularity, and consistency, as suggested by [55,56]. We provide the following detailed methodology for content evaluation. First, the evaluation prompts were designed in order to ask GPT-4o to assess each of the generated structured outputs against the defined criteria (completeness, accuracy, granularity, and consistency). Each criterion was presented clearly, with detailed descriptions and examples, in order to ensure that the LLM had an accurate understanding of what was being evaluated.

- Scoring Methodology: For each output, GPT-4o was prompted to assign a score between 1 and 5 for each of the four criteria. The scoring process involved:
    1. Rating Scale: GPT-4o rated each criterion on a five-point Likert scale. The scale was explicitly defined in the prompt to guide the scoring:
        - 1: Very poor quality or incorrect representation of the information.
        - 2: Poor quality, with significant gaps or errors.
        - 3: Satisfactory, but with noticeable issues or missing details.
        - 4: Good quality, with minor issues or omissions.
        - 5: Excellent quality, fully meeting the criteria without any noticeable issues.
    2. Score Aggregation: The individual scores for each criterion were aggregated to compute an overall quality score for the generated content. Specifically, for each structured output, the scores across all four criteria were averaged to provide a final content quality score ranging from 1 to 5.

To ensure the reliability of the scoring process, multiple runs (by default, three) of GPT-4o were used to evaluate a subset of the generated outputs. Consistency between the scores given across different runs was analyzed to validate the robustness of the evaluation. The evaluation criteria details and prompt are listed below:

- Completeness: Assesses how well the generated JSON or GeoJSON captures all the relevant information present in the input text. This aims to check whether all of the important entities, relationships, and attributes are correctly identified and included in the output.
- Accuracy: Evaluates the correctness of the extracted information in the JSON output, verifies whether the values, data types, and structures match the information provided in the input text, and aims to check for any errors, inconsistencies, or misinterpretations.
- Granularity: Considers the level of detail captured in the JSON output and assesses whether the generated structure provides an appropriate level of granularity based on the requirements of the task. This aims to determine whether the JSON includes all the necessary fields and substructures to represent the information effectively.
- Consistency: When multiple examples or instances are provided, the consistency is used to evaluate the generated JSON across different inputs. This aims to check whether the LLM maintains a consistent structure, naming conventions, and data representation across various examples.

### *5.3. Format Error Evaluation*

In our evaluation, we assess the syntactic correctness of the generated outputs by verifying their adherence to the JSON format. A well-structured output should be a valid JSON object, enabling seamless storage in databases or further computational processing. To determine the presence of format errors, we implement the following procedure:

1. Parsing Attempt: Each generated output is subjected to a JSON parsing process.
2. Error Identification: If the parser encounters syntax errors such as missing commas, unclosed brackets, or improper nesting, then the output is flagged as containing a format error.

The format error rate $E_{\text{format}}$ is then calculated to quantify the prevalence of formatting issues:

$$E_{\text{format}} = \frac{N_{\text{error}}}{N_{\text{total}}} \tag{11}$$

where:

- $N_{\text{error}}$ is the number of outputs with format errors.
- $N_{\text{total}}$ is the total number of outputs generated.

A lower $E_{\text{format}}$ indicates a higher proficiency of the model in producing correctly formatted JSON objects.

## 6. Experiments

To evaluate the effectiveness of fine-tuning, RAG, prompt engineering, and REL for structured JSON output generation, we conducted a series of experiments using two types of collected data, namely, a general JSON dataset and a GeoJSON dataset.

### *6.1. Setup*

For fine-tuning, we used the LoRA technique combined with 4-bit quantization using the GPTQ algorithm. For LoRA, we set the alpha to 16 and dropout to 0.1, as recommended in the original paper [10]. The LoRA rank was set to 8, which reduces the number of trainable parameters and computational complexity while still allowing for effective adaptation. A rank of 8 provides a good balance between efficiency and performance. We used a learning rate of $2 \times 10^{-4}$, which is commonly used for fine-tuning large language models and allows for stable and gradual updates to the model's parameters during training. A weight decay of 0.001 was applied as a regularization technique to prevent overfitting by adding a penalty term. The max gradient norm was set to 0.3, which rescales the gradients if their norm exceeds the threshold, helping to maintain stable gradients and improve training

convergence. The fine-tuning experiments were conducted on an Amazon SageMaker g5.xlarge machine with consistent settings across all datasets.

For the RAG and prompt engineering experiments, we utilized the GPT-4o-mini API as the base model. In the RAG setup, we used FAISS as the vector store, storing ten example pairs as default for retrieval during generation. For prompt engineering, we designed prompts that included explicit instructions, representative examples, JSON schemas, and clear separators to guide the model's output. For the REL setup, we used fine-tuned llama3-7b as the base model, which aims for relatively better performance and accuracy [47].

### *6.2. Results*

#### 6.2.1. Structural Validity

All models produced lower scores on the high-complexity datasets, compared to the low-complexity datasets. Among the fine-tuned methods, the llama3-7b model outperformed and even surpassed the RAG method. The key-level JScore indicates whether the method or model can generate the correct key that is retrievable by downstream tasks. The value JScore is more flexible, as the value of the formatted content might require more free-form content such as address, name, or ID. Tables 2–5 present the JSON structural validity results using the JScore metric. An example result is also shown in Appendix A Table A7.

It is worth noting that although lightweight models such as gemma-2b and gemma-7b do not perform as well in terms of overall JScore, they still achieve good results. It can also be seen that larger parameter sizes yield better results, even when the models have the same structure. Another lightweight model, phi-2, still achieves good performance on structured output generation with fine-tuned techniques despite its limited parameters. Moreover, the codellama2-7b model, which is trained using a code base, shows strong results, indicating that the training data of the base model are an important factor in structured output generation performance.

**Table 2.** Results for structured output generation across different methods on the Schema Dataset.

| Method | JScore | Key JScore | Value JScore | Format Error | Edit Distance |
|---|---|---|---|---|---|
| PE | 0.56 | 0.61 | 0.52 | 0.04 | 0.50 |
| RAG | 0.52 | 0.57 | 0.47 | 0.14 | 0.46 |
| REL | 0.55 | 0.62 | 0.60 | 0.00 | 0.65 |
| gemma-2b | 0.25 | 0.28 | 0.21 | 0.47 | 0.32 |
| gemma-7b | 0.33 | 0.37 | 0.28 | 0.16 | 0.30 |
| codellama2-7b | 0.51 | 0.57 | 0.44 | 0.12 | 0.47 |
| phi-2 | 0.39 | 0.42 | 0.35 | 0.29 | 0.45 |
| mistral-7b | 0.54 | 0.59 | 0.50 | 0.12 | 0.62 |
| llama3-7b | 0.57 | 0.60 | 0.57 | 0.00 | 0.64 |

**Table 3.** Results for structured output generation across different methods on the Paraloq Dataset.

| Method | JScore | Key JScore | Value JScore | Format Error | Edit Distance |
|---|---|---|---|---|---|
| PE | 0.91 | 0.99 | 0.83 | 0.00 | 0.68 |
| RAG | 0.91 | 0.99 | 0.82 | 0.00 | 0.61 |
| REL | 0.90 | 0.95 | 0.82 | 0.03 | 0.80 |
| gemma-2b | 0.73 | 0.81 | 0.65 | 0.14 | 0.73 |
| gemma-7b | 0.81 | 0.88 | 0.74 | 0.10 | 0.80 |
| codellama2-7b | 0.89 | 0.96 | 0.81 | 0.04 | 0.82 |
| phi-2 | 0.72 | 0.79 | 0.66 | 0.20 | 0.73 |
| mistral-7b | 0.87 | 0.95 | 0.79 | 0.06 | 0.82 |
| llama3-7b | 0.90 | 0.96 | 0.84 | 0.01 | 0.85 |

**Table 4.** Results for structured output generation across different methods on the Nous Dataset.

| Method | JScore | Key JScore | Value JScore | Format Error | Edit Distance |
|---|---|---|---|---|---|
| PE | 0.96 | 0.93 | 0.99 | 0.00 | 0.99 |
| RAG | 1.00 | 1.00 | 1.00 | 0.00 | 1.00 |
| REL | 1.00 | 1.00 | 1.00 | 0.00 | 1.00 |
| gemma-2b | 0.88 | 0.92 | 0.83 | 0.10 | 0.88 |
| gemma-7b | 0.91 | 0.93 | 0.89 | 0.00 | 0.87 |
| codellama2-7b | 0.99 | 1.00 | 0.98 | 0.00 | 1.00 |
| phi-2 | 1.00 | 1.00 | 0.99 | 0.00 | 1.00 |
| mistral-7b | 0.98 | 1.00 | 0.96 | 0.00 | 0.99 |
| llama3-7b | 1.00 | 1.00 | 1.00 | 0.00 | 1.00 |

**Table 5.** Results for structured output generation across different methods on the GeoJSON Dataset.

| Method | JScore | Key JScore | Value JScore | Format Error | Edit Distance |
|---|---|---|---|---|---|
| PE | 0.87 | 0.89 | 0.82 | 0.12 | 0.85 |
| RAG | 0.89 | 0.90 | 0.85 | 0.09 | 0.86 |
| REL | 0.91 | 0.92 | 0.90 | 0.03 | 0.87 |
| gemma-2b | 0.75 | 0.72 | 0.61 | 0.21 | 0.78 |
| gemma-7b | 0.80 | 0.84 | 0.73 | 0.19 | 0.77 |
| codellama2-7b | 0.82 | 0.83 | 0.84 | 0.15 | 0.80 |
| phi-2 | 0.81 | 0.79 | 0.64 | 0.24 | 0.72 |
| mistral-7b | 0.88 | 0.91 | 0.86 | 0.09 | 0.90 |
| llama3-7b | 0.90 | 0.92 | 0.89 | 0.06 | 0.89 |

One notable column shows the format error, which indicates whether the generated JSON or GeoJSON is formatted correctly. For example, it checks whether the GeoJSON contains a "feature" section, which is required as an attribute for spatial objects. A correctly formatted JSON will be enclosed with two curly brackets, and all nested JSON values must also follow this structure, such as lists and strings with double quotes. Among all methods, prompt engineering and the RAG method tend to have lower format error rates, likely due to their higher number of model parameters. The error rate also decreases as the model's parameter count increases. The following experiments on training steps and format error rate further demonstrate that a lower format error rate can be achieved with more training steps, as shown in Table 6.

**Table 6.** Format error rate of mistral-7b model and llama3-7b model with increasing training steps.

| Format Error Rate | Training Steps | | | | |
|---|---|---|---|---|---|
| | 100 | 200 | 300 | 400 | 500 |
| mistral-7b | 0.29 | 0.22 | 0.18 | 0.15 | 0.14 |
| llama3-7b | 0.25 | 0.19 | 0.14 | 0.12 | 0.09 |

Another metric in this table is the edit distance. Similar to the cosine similarity score, this is an overall similarity rate that is commonly used in other benchmarks for free-form language as a reference. It shows that even if a method has a high edit distance or similarity score, the model can still generate formatted error answers and many more key errors, which is detrimental to formatting in downstream tasks.

Figure 3 shows that among all the methods, prompt engineering outperforms the others with an average JScore of 0.81. However, mistral-7B and llama3-7b also achieve close performance of 0.80, which suggests the possibility of further fine-tuning for better results with a larger training dataset. Table 7 shows the training loss for all of the fine-tuned models. It is important to note that the training loss alone cannot be the sole metric for evaluating a model's

performance, even when using the same trainer on the same dataset. Other factors, such as model architecture, hyperparameters, and evaluation metrics, should also be considered when assessing a model's overall effectiveness. For the optimization methods, it can be seen that the REL outperforms all other PE and RAG methods with the support of fine-tuning.

**Table 7.** Fine-tuning training loss for different models on the Nous, Paraloq, and Schema datasets.

| Model | Nous | Paraloq | Schema |
|---|---|---|---|
| phi-2 | 0.1435 | 0.4467 | 0.7240 |
| gemma-2b | 0.0491 | 0.2469 | 0.1752 |
| mistral-7b | 0.0177 | 0.1215 | 0.0826 |
| codellama2-7b | 0.0420 | 0.2197 | 0.1366 |
| gemma-7b | 0.1613 | 0.2657 | 0.2742 |
| llama3-7b | 0.0194 | 0.1031 | 0.0821 |

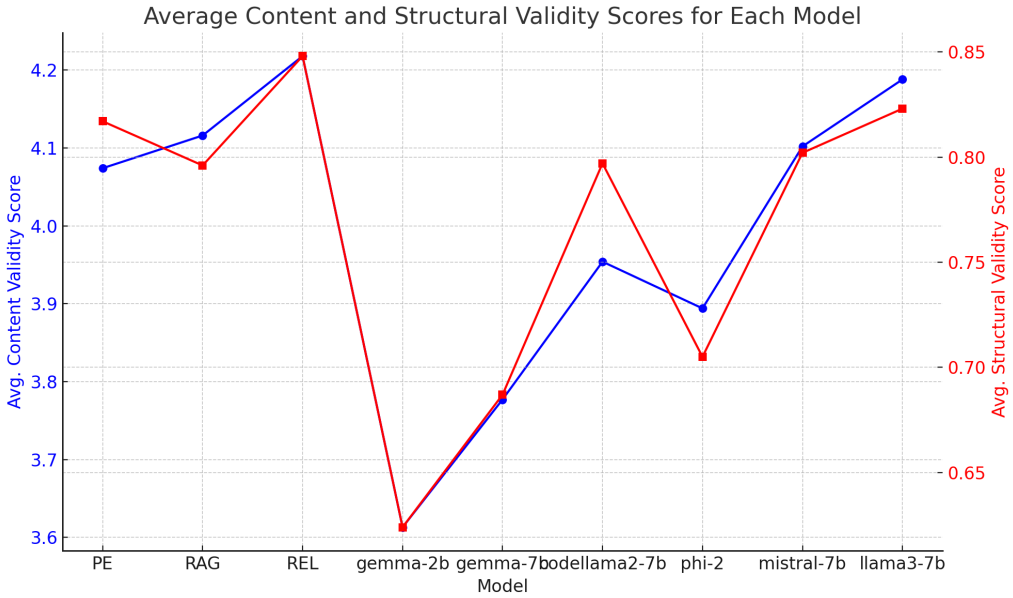


**Figure 3.** Demonstration of validity scores for each model across all datasets.

Table A4 demonstrates that the REL-enhanced optimization schema significantly outperforms all other models on more complex tasks such as GeoJSON generation, particularly in reducing format errors. The edit distance metric does not show a significant difference, potentially due to the effect of generating large numbers of tokens for the longitude and latitude values.

#### 6.2.2. Content Validity

The content validity results are presented in the Appendix A Tables A1–A4. These tables represent human-imitated evaluation methods conducted by GPT-4 using four criteria: completeness, accuracy, granularity, and consistency. The last column indicates the memory and token costs, showing that fine-tuned models are more efficient than large-size LLMs. The results show that the high-complexity dataset from Schema.org has the lowest scores, similar to the previous JScore results. However, considering the overall results of all fine-tuned models, the llama3-7b model outperforms all other models in all tasks, with an average score of 4.188 for all metrics across all datasets, as shown in Figure 3.

Surprisingly, the phi-2 model, which has only 2.79 billion parameters, outperforms the gemma-7b model, which has 7 billion parameters. This could be attributed to the fact that the phi-2 model was trained on a more diverse and programming-related dataset [44], similar to codellama2 [45], enabling it to generate more accurate and contextually appropriate

responses. Additionally, the architecture and training techniques used for the phi-2 model might have been more optimized for the specific task of structured output generation.

It is also worth noting the statistical standard deviation $\sigma$ of the JScore when calculating the average scores of all metrics. The average $\sigma$ of the JScore on the Paraloq dataset is 0.25 and the average $\sigma$ of the Schema is 0.29, which demonstrates the high complexity and lower model robustness of these results. On the other hand, the content validity across all datasets and models remains close to 1, with Schema at 1.03, Paraloq at 1.09, Nous at 0.95, and GeoJSON at 0.98. One explanation for this might be that a generated benchmark maintains a more balanced score compared with fixed metrics.

## 7. Conclusions

In this study, we have conducted a comprehensive evaluation of multiple methods for generating structured JSON outputs using LLMs. We collected three datasets of varying complexity to assess model performance, and introduce an optimization method integrating the R-tree approach for better indexing in GeoJSON generation.

Our findings conclusively demonstrate the capability of LLMs in structured output generation, such as highlighting the impact of dataset complexity, computational resource requirements, and the quality of generated outputs. Specifically, our experiments reveal that while LLMs can generate high-quality structured data, certain limitations arise with increasing dataset complexity. For example, due to the constraints of our experimental setup, the maximum token limit of 10k restricted our ability to fully represent larger datasets such as global-level GeoJSON data, which can extend beyond 100k tokens. This emphasizes the need for more robust infrastructure to effectively handle larger and more complex datasets. Additionally, our work highlights the limitations imposed by computational resources, with our current experiments capped at a 24 GB VRAM capacity. These resource limitations influenced the effectiveness of our LLM fine-tuning, suggesting that increased computational power could yield more accurate and expansive results. Thus, practitioners should consider the balance between dataset complexity and available computational resources when selecting methods for structured output generation.

Moving forward, we envision extending our framework to develop a spatial generation approach that caters to more demanding real-world applications. Future research can focus on optimizing performance and efficiency for industrial-scale spatial datasets. Additionally, adapting this framework for broader use cases presents exciting opportunities where structured output generation is essential, including applications in urban planning, public health, and disaster management. The quality and applicability of generated outputs can be significantly enhanced by leveraging models trained on diverse domain-specific datasets, paving the way for impactful research and development in these areas.

**Author Contributions:** Conceptualization, Diya Li, Zhe Zhang and Calvin Jung; methodology, Diya Li; software, Diya Li; validation, Diya Li, Zhifang Wang and Yue Zhao; formal analysis, Diya Li; investigation, Diya Li; resources, Zhe Zhang; data curation, Diya Li; writing—original draft preparation, Diya Li; writing—review and editing, Diya Li; visualization, Diya Li; supervision, Zhe Zhang; project administration, Zhe Zhang. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research was funded by (1) Collaborative Research: CyberTraining: Implementation: Small: Broadening Adoption of Cyberinfrastructure and Research Workforce Development for Disaster Management, grant number 2321069; (2) MRI: Acquisition of FASTER—Fostering Accelerated Sciences Transformation Education and Research, grant number 2019129; (3) CC* Data Storage: FASTER Data Infrastructure to Accelerate Computing, grant number 2322377; (4) CAREER: A Cyberinfrastructure-Enabled Hybrid Spatial Decision Support System for Improving Coastal Resilience to Flood Risks, grant number 2339174.

**Data Availability Statement:** All data supporting the findings of this study are available within the article. See https://github.com/dyllanwli/llm-structured-output-public (accessed on 6 November 2024).

**Acknowledgments:** We would like to express our sincere gratitude to Moxie Zhang and the ArcGIS Survey123 Team for their support of this research. Their generous contributions have been instrumental in making this work possible.

**Conflicts of Interest:** The authors declare no conflicts of interest.

## Appendix A

**Table A1.** Content scores for structured output generation across different methods on the Schema dataset.

| Method | Completeness | Accuracy | Granularity | Consistency |
|---|---|---|---|---|
| PE | 3.10 | 3.53 | 2.86 | 4.08 |
| RAG | 3.12 | 3.61 | 2.88 | 3.88 |
| REL | 3.32 | 3.56 | 3.16 | 3.87 |
| gemma-2b | 2.16 | 2.63 | 1.94 | 3.02 |
| gemma-7b | 2.27 | 2.98 | 2.20 | 3.10 |
| codellama2-7b | 2.92 | 3.49 | 2.69 | 3.55 |
| phi-2 | 2.80 | 3.33 | 2.73 | 3.39 |
| mistral-7b | 3.16 | 3.61 | 2.90 | 3.86 |
| llama3-7b | 3.43 | 3.60 | 3.22 | 4.02 |

**Table A2.** Content scores for structured output generation across different methods on the Paraloq dataset.

| Method | Completeness | Accuracy | Granularity | Consistency |
|---|---|---|---|---|
| PE | 4.00 | 4.02 | 4.14 | 4.55 |
| RAG | 3.96 | 4.08 | 4.02 | 4.53 |
| REL | 4.14 | 4.06 | 4.11 | 4.34 |
| gemma-2b | 3.57 | 3.76 | 3.49 | 3.98 |
| gemma-7b | 3.96 | 3.96 | 4.02 | 4.49 |
| codellama2-7b | 3.96 | 3.96 | 3.98 | 4.51 |
| phi-2 | 3.76 | 3.80 | 3.76 | 4.06 |
| mistral-7b | 4.12 | 3.98 | 4.14 | 4.51 |
| llama3-7b | 4.21 | 4.02 | 4.17 | 4.60 |

**Table A3.** Content scores for structured output generation across different methods on the Nous dataset.

| Method | Completeness | Accuracy | Granularity | Consistency |
|---|---|---|---|---|
| PE | 4.80 | 4.70 | 4.20 | 4.80 |
| RAG | 4.90 | 4.80 | 4.60 | 5.00 |
| REL | 4.80 | 4.60 | 4.70 | 5.00 |
| gemma-2b | 4.40 | 4.20 | 4.20 | 4.70 |
| gemma-7b | 4.30 | 4.50 | 3.60 | 4.60 |
| codellama2-7b | 4.80 | 4.60 | 4.50 | 5.00 |
| phi-2 | 4.80 | 4.90 | 4.60 | 4.80 |
| mistral-7b | 4.70 | 4.70 | 4.50 | 5.00 |
| llama3-7b | 4.90 | 4.70 | 4.70 | 5.00 |

**Table A4.** Content scores for structured output generation across different methods on theGeoJSON dataset.

| Method | Completeness | Accuracy | Granularity | Consistency |
|---|---|---|---|---|
| PE | 3.45 | 3.52 | 3.38 | 3.81 |
| RAG | 3.49 | 3.57 | 3.42 | 3.85 |
| REL | 3.64 | 3.68 | 3.53 | 4.08 |
| gemma-2b | 3.21 | 3.18 | 3.15 | 3.53 |
| gemma-7b | 3.38 | 3.45 | 3.32 | 3.74 |
| codellama2-7b | 3.52 | 3.58 | 3.45 | 3.89 |
| phi-2 | 3.35 | 3.42 | 3.28 | 3.71 |
| mistral-7b | 3.55 | 3.62 | 3.48 | 3.92 |
| llama3-7b | 3.62 | 3.68 | 3.45 | 4.01 |

**Table A5.** Example showing the use of LLMs to extract the spatial-related dataset.

**Prompt**

```
Given data products, if a data product is clearly related to
spatial data, return the UUID. If the data product is not
related to spatial data, return na.
texts: {input_texts}
Output format:
{‘‘uuid’’: [‘‘********-****-****-****-************’’, ‘‘na’’, ...]}
{format_instructions}
```

**Response Example**

```
{‘‘uuid’’: [‘‘f70051bf-a763-415b-aa66-97ae57f2efc1’’]}
```

**Table A6.** Example of prompt engineering for structured output generation.

**Prompt**

```
pre_prompt = f‘‘Given few examples of instructions and responses
from the training dataset.
The task is to generate a response for the given instruction. {task}\n\n’’

for i, example in enumerate(examples):
    pre_prompt += f"""
    Example {i+1}:\nInstruction: {example['instruction']}\n
    Response: {example['response']}\n\n
    """
```

**Response Example**

```
// Nous Dataset
{‘‘generated_responses’’: [{\"componentID\": \"RES-0001\",
\"componentName\":
\"220 Ohm Resistor\", \"specifications\": {...}}]
// Paraloq Dataset
{‘‘generated_responses’’: [{\n  \"feedback\": [\n    {\n
\"feedbackRating\": \"Somewhat positive\", ... }]}]}
// GeoJSON Dataset
{‘‘generated_responses’’: [{\"type\":\"Feature\",\"properties\":
{\"NAME\":\"Smith Art Gallery\",\"INFORMATION\":\"
..\",\"Easting\":\"414216.415\",\"Northing\":\"423086.01\"},
\"geometry\":{\"type\":\"Point\",\"coordinates\":
[-1.786128,53.704014]}]}
```

**Table A7.** Examples of different content generation methods.

**Prompt**

```
pre_prompt = """Acrodding to the publication {INDEX}, help me generate
the related GeoJSON Object.
"""
```

**REL Response Example**

```
{‘‘type’’:‘‘FeatureCollection’’,‘‘features’’:[{‘‘type’’:‘‘Feature’’,
‘‘properties’’:
{‘‘STATION’’:‘‘Brighouse’’,‘‘Easting’’:‘‘414189.9979’’,‘‘Northing’’:
‘‘423144.9948’’},‘‘geometry’’:{‘‘type’’:‘‘Point’’,‘‘coordinates’’:
[-1.785022,53.704317]}},
{‘‘type’’:‘‘Feature’’,‘‘properties’’:{‘‘STATION’’:‘‘Elland’’,
‘‘Easting’’:‘‘411045.00115’’,‘‘Northing’’:
‘‘420553.9968’’},‘‘geometry’’:{‘‘type’’:‘‘Point’’,
‘‘coordinates’’:[-1.832761,53.681102]}},
{‘‘type’’:‘‘Feature’’,‘‘properties’’:{‘‘STATION’’:’’Halifax’’,‘‘Easting’’:
‘‘407884.00205’’,‘‘Northing’’:‘‘424132.0032’’},‘‘geometry’’:
{‘‘type’’:‘‘Point’’,‘‘coordinates’’:[-1.880532,53.713322]}}
,{‘‘type’’:‘‘Feature’’,‘‘properties’’:
{‘‘STATION’’:‘‘Illingworth’’,‘‘Easting’’:‘‘407084.99685’’,
‘‘Northing’’:‘‘428741.0028’’},‘‘geometry’’:
{‘‘type’’:‘‘Point’’,‘‘coordinates’’:[-1.892534,53.754765]}}
,{‘‘type’’:‘‘Feature’’,‘‘properties’’:
{‘‘STATION’’:‘‘Mytholmroyd’’,‘‘Easting’’:‘‘401120.0011’’,
‘‘Northing’’:‘‘426149.9952’’},‘‘geometry’’:
{‘‘type’’:‘‘Point’’,‘‘coordinates’’:[-1.983021,53.731521]}}
,{‘‘type’’:’’Feature’’,‘‘properties’’:
{‘‘STATION’’:‘‘Todmorden’’,‘‘Easting’’:‘‘393790.00115’’,
    ‘‘Northing’’:‘‘424414.002’’},‘‘geometry’’:
{‘‘type’’:‘‘Point’’,‘‘coordinates’’:[-2.094107,53.71588]}]}]}
```

**llama3-7b Response Example**

```
{‘‘type’’:‘‘FeatureCollection’’,‘‘features’’:[{‘‘type’’:
‘‘Feature’’,‘‘properties’’:{‘‘STATION’’:‘‘Brighouse’’,
‘‘Easting’’:‘‘414189.9979’’,‘‘Northing’’:‘‘423144.9948’’},
‘‘geometry’’:{‘‘type’’:‘‘Point’’,‘‘coordinates’’:[-1.785022,53.704317]}},
{‘‘type’’:‘‘Feature’’,‘‘properties’’:{‘‘STATION’’:‘‘Elland’’,
‘‘Easting’’:‘‘411045.00115’’,‘‘Northing’’:‘‘420553.9968’’},
‘‘geometry’’:{‘‘type’’:‘‘Point’’,‘‘coordinates’’:[-1.832761,53.681102]}},
{‘‘type’’:‘‘Feature’’,‘‘properties’’:{‘‘STATION’’:‘‘Halifax’’,
‘‘Easting’’:‘‘407884.00205’’,‘‘Northing’’:‘‘424132.0032’’},‘‘geometry’’:
{‘‘type’’:‘‘Point’’,‘‘coordinates’’:[-1.880532,53.713322]}},{‘‘type’’:
‘‘Feature’’,‘‘properties’’:{‘‘STATION’’:‘‘Illingworth’’,‘‘Easting’’:
‘‘407084.99685’’,‘‘Northing’’:‘‘428741.0028’’},‘‘geometry’’:{‘‘type’’:
‘‘Point’’,
‘‘coordinates’’:[-1.892534,53.754765]}},{‘‘type’’:‘‘Feature’’,‘‘properties’’:
{‘‘STATION’’:‘‘Mytholmroyd’’,‘‘Easting’’:‘‘401120.0011’’,
‘‘Northing’’:‘‘426149.9952’’},‘‘geometry’’:{‘‘type’’:‘‘Point’’,
‘‘coordinates’’:[-1.983021,53.731521]}}]}
```

## References


1. Brown, T.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J.D.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; et al. Language models are few-shot learners. *Adv. Neural Inf. Process. Syst.* **2020**, *33*, 1877–1901.
2. Thirunavukarasu, A.J.; Ting, D.S.J.; Elangovan, K.; Gutierrez, L.; Tan, T.F.; Ting, D.S.W. Large language models in medicine. *Nat. Med.* **2023**, *29*, 1930–1940. [CrossRef] [PubMed]
3. Wu, S.; Irsoy, O.; Lu, S.; Dabravolski, V.; Dredze, M.; Gehrmann, S.; Kambadur, P.; Rosenberg, D.; Mann, G. Bloomberggpt: A large language model for finance. *arXiv* **2023**, arXiv:2303.17564.
4. Yin, Z.; Li, D.; Goldberg, D.W. Is ChatGPT a game changer for geocoding-a benchmark for geocoding address parsing techniques. In Proceedings of the 2nd ACM SIGSPATIAL International Workshop on Searching and Mining Large Collections of Geospatial Data, Hamburg, Germany, 13 November 2023; pp. 1–8.
5. Yin, Z.; Zhang, C.; Goldberg, D.W.; Prasad, S. An NLP-based question answering framework for spatio-temporal analysis and visualization. In Proceedings of the 2019 2nd International Conference on Geoinformatics and Data Analysis, Prague, Czech Republic, 15–17 March 2019; pp. 61–65.

6. Zhang, Z.; Li, D.; Zhang, Z.; Duffield, N. Mining Spatiotemporal Mobility Patterns Using Improved Deep Time Series Clustering. *ISPRS Int. J. -Geo-Inf.* **2024**, *13*, 374. [CrossRef]
7. Wei, J.; Wang, X.; Schuurmans, D.; Bosma, M.; Xia, F.; Chi, E.; Le, Q.V.; Zhou, D. Chain-of-thought prompting elicits reasoning in large language models. *Adv. Neural Inf. Process. Syst.* **2022**, *35*, 24824–24837.
8. Wu, Q.; Bansal, G.; Zhang, J.; Wu, Y.; Zhang, S.; Zhu, E.; Li, B.; Jiang, L.; Zhang, X.; Wang, C. Autogen: Enabling next-gen llm applications via multi-agent conversation framework. *arXiv* **2023**, arXiv:2308.08155.
9. Li, D.; Zhang, Z. MetaQA: Enhancing human-centered data search using Generative Pre-trained Transformer (GPT) language model and artificial intelligence. *PLoS ONE* **2023**, *18*, e0293034. [CrossRef]
10. Hu, E.J.; Shen, Y.; Wallis, P.; Allen-Zhu, Z.; Li, Y.; Wang, S.; Wang, L.; Chen, W. Lora: Low-rank adaptation of large language models. *arXiv* **2021**, arXiv:2106.09685.
11. Lewis, P.; Perez, E.; Piktus, A.; Petroni, F.; Karpukhin, V.; Goyal, N.; Küttler, H.; Lewis, M.; Yih, W.t.; Rocktäschel, T.; et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Adv. Neural Inf. Process. Syst.* **2020**, *33*, 9459–9474.
12. White, J.; Fu, Q.; Hays, S.; Sandborn, M.; Olea, C.; Gilbert, H.; Elnashar, A.; Spencer-Smith, J.; Schmidt, D.C. A prompt pattern catalog to enhance prompt engineering with chatgpt. *arXiv* **2023**, arXiv:2302.11382.
13. Hou, X.; Zhao, Y.; Liu, Y.; Yang, Z.; Wang, K.; Li, L.; Luo, X.; Lo, D.; Grundy, J.; Wang, H. Large language models for software engineering: A systematic literature review. *arXiv* **2023**, arXiv:2308.10620. [CrossRef]
14. Zhang, P.; Zeng, G.; Wang, T.; Lu, W. Tinyllama: An open-source small language model. *arXiv* **2024**, arXiv:2401.02385.
15. Li, D.; Zhang, Z.; Alizadeh, B.; Zhang, Z.; Duffield, N.; Meyer, M.A.; Thompson, C.M.; Gao, H.; Behzadan, A.H. A reinforcement learning-based routing algorithm for large street networks. *Int. J. Geogr. Inf. Sci.* **2024**, *38*, 183–215. [CrossRef]
16. Beckmann, N.; Kriegel, H.P.; Schneider, R.; Seeger, B. The R*-tree: An efficient and robust access method for points and rectangles. In Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data, Atlantic, NJ, USA, 23–26 May 1990; pp. 322–331.
17. Nijkamp, E.; Pang, B.; Hayashi, H.; Tu, L.; Wang, H.; Zhou, Y.; Savarese, S.; Xiong, C. Codegen: An open large language model for code with multi-turn program synthesis. *arXiv* **2022**, arXiv:2203.13474.
18. Li, J.; Hui, B.; Qu, G.; Yang, J.; Li, B.; Li, B.; Wang, B.; Qin, B.; Geng, R.; Huo, N.; et al. Can llm already serve as a database interface? A big bench for large-scale database grounded text-to-sqls. *arXiv* **2024**, arXiv:2305.03111.
19. Patil, S.G.; Zhang, T.; Wang, X.; Gonzalez, J.E. Gorilla: Large language model connected with massive apis. *arXiv* **2023**, arXiv:2305.15334.
20. Laha, A.; Jain, P.; Mishra, A.; Sankaranarayanan, K. Scalable micro-planned generation of discourse from structured data. *Comput. Linguist.* **2020**, *45*, 737–763. [CrossRef]
21. Golubev, A.; Chechetkin, I.; Parygin, D.; Sokolov, A.; Shcherbakov, M. Geospatial data generation and preprocessing tools for urban computing system development. *Procedia Comput. Sci.* **2016**, *101*, 217–226. [CrossRef]
22. LangChain. How to Return Structured Data from a Model. 2023. Available online: https://python.langchain.com/docs/how_to/structured_output/ (accessed on 14 October 2024).
23. Ko, H.; Yang, H.; Han, S.; Kim, S.; Lim, S.; Hormazabal, R. Filling in the Gaps: LLM-Based Structured Data Generation from Semi-Structured Scientific Data. In Proceedings of the ICML 2024 AI for Science Workshop, Vienna, Austria, 21–27 July 2024.
24. Pezoa, F.; Reutter, J.L.; Suarez, F.; Ugarte, M.; Vrgoč, D. Foundations of JSON schema. In Proceedings of the 25th International Conference on World Wide Web, Montreal, QC, Canada, 11–15 April 2016; pp. 263–273.
25. Escarda-Fernández, M.; López-Riobóo-Botana, I.; Barro-Tojeiro, S.; Padrón-Cousillas, L.; Gonzalez-Vázquez, S.; Carreiro-Alonso, A.; Gómez-Area, P. LLMs on the Fly: Text-to-JSON for Custom API Calling. In Proceedings of the SEPLN-CEDI 2024: VII Congreso Español de Informática, A Coruña, Spain, 19–20 June 2024.
26. Beurer-Kellner, L.; Fischer, M.; Vechev, M. Guiding LLMs The Right Way: Fast, Non-Invasive Constrained Generation. *arXiv* **2024**, arXiv:2403.06988.
27. Mior, M.J. Large Language Models for JSON Schema Discovery. *arXiv* **2024**, arXiv:2407.03286.
28. Karimzadeh, M.; Pezanowski, S.; MacEachren, A.M.; Wallgrün, J.O. GeoTxt: A scalable geoparsing system for unstructured text geolocation. *Trans. GIS* **2019**, *23*, 118–136. [CrossRef]
29. Ning, H.; Li, Z.; Akinboyewa, T.; Lessani, M.N. LLM-Find: An Autonomous GIS Agent Framework for Geospatial Data Retrieval. *arXiv* **2024**, arXiv:2407.21024.
30. Jordahl, K. GeoPandas: Python Tools for Geographic Data. 2024. Available online: https://github.com/geopandas/geopandas (accessed on 16 October 2024).
31. Esri. Artificial Intelligence (AI) and Location Intelligence. Available online: https://www.esri.com/en-us/artificial-intelligence/overview (accessed on 16 October 2024).
32. Qi, J.; Li, Z.; Tanin, E. MaaSDB: Spatial Databases in the Era of Large Language Models (Vision Paper). In Proceedings of the 31st ACM International Conference on Advances in Geographic Information Systems, Hamburg, Germany, 13–16 November 2023; pp. 1–4.
33. Musumeci, E.; Brienza, M.; Suriani, V.; Nardi, D.; Bloisi, D.D. LLM Based Multi-Agent Generation of Semi-structured Documents from Semantic Templates in the Public Administration Domain. *arXiv* **2024**, arXiv:2402.14871.
34. Chen, J.; Lin, H.; Han, X.; Sun, L. Benchmarking large language models in retrieval-augmented generation. In Proceedings of the AAAI Conference on Artificial Intelligence, Vancouver, QC, Canada, 20–27 February 2024; Volume 38, pp. 17754–17762.

35. Zheng, L.; Yin, L.; Xie, Z.; Huang, J.; Sun, C.; Yu, C.H.; Cao, S.; Kozyrakis, C.; Stoica, I.; Gonzalez, J.E.; et al. Efficiently programming large language models using sglang. *arXiv* **2023**, arXiv:2312.07104.
36. Yang, S.; Zhao, H.; Zhu, S.; Zhou, G.; Xu, H.; Jia, Y.; Zan, H. Zhongjing: Enhancing the chinese medical capabilities of large language model through expert feedback and real-world multi-turn dialogue. In Proceedings of the AAAI Conference on Artificial Intelligence, Vancouver, QC, Canada, 20–27 February 2024; Volume 38, pp. 19368–19376.
37. Zhao, Y.; Pang, T.; Du, C.; Yang, X.; Li, C.; Cheung, N.M.M.; Lin, M. On evaluating adversarial robustness of large vision-language models. *arXiv* **2024**, arXiv:2305.16934.
38. Schema.org. Schema.org Vocabulary Data. Available online: https://schema.org/ (accessed on 22 March 2024).
39. NousResearch. Json Mode Eval. Available online: https://huggingface.co/datasets/NousResearch/json-mode-eval (accessed on 22 March 2024).
40. Arrich, M. Paraloq Json Data Extraction. Available online: https://huggingface.co/datasets/paraloq/json_data_extraction (accessed on 22 March 2024).
41. Frantar, E.; Ashkboos, S.; Hoefler, T.; Alistarh, D. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv* **2022**, arXiv:2210.17323.
42. HuggingFace. Supervised Fine-Tuning Trainer. Available online: https://huggingface.co/docs/trl/en/sft_trainer (accessed on 27 March 2024).
43. Team, G.; Mesnard, T.; Hardin, C.; Dadashi, R.; Bhupatiraju, S.; Pathak, S.; Sifre, L.; Rivière, M.; Kale, M.S.; Love, J.; et al. Gemma: Open models based on gemini research and technology. *arXiv* **2024**, arXiv:2403.08295.
44. Gunasekar, S.; Zhang, Y.; Aneja, J.; Mendes, C.C.T.; Del Giorno, A.; Gopi, S.; Javaheripi, M.; Kauffmann, P.; de Rosa, G.; Saarikivi, O.; et al. Textbooks are all you need. *arXiv* **2023**, arXiv:2306.11644.
45. Touvron, H.; Martin, L.; Stone, K.; Albert, P.; Almahairi, A.; Babaei, Y.; Bashlykov, N.; Batra, S.; Bhargava, P.; Bhosale, S.; et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv* **2023**, arXiv:2307.09288.
46. Jiang, A.Q.; Sablayrolles, A.; Mensch, A.; Bamford, C.; Chaplot, D.S.; Casas, D.d.l.; Bressand, F.; Lengyel, G.; Lample, G.; Saulnier, L.; et al. Mistral 7B. *arXiv* **2023**, arXiv:2310.06825.
47. Dubey, A.; Jauhri, A.; Pandey, A.; Kadian, A.; Al-Dahle, A.; Letman, A.; Mathur, A.; Schelten, A.; Yang, A.; Fan, A.; et al. The llama 3 herd of models. *arXiv* **2024**, arXiv:2407.21783.
48. Douze, M.; Guzhva, A.; Deng, C.; Johnson, J.; Szilvasy, G.; Mazaré, P.E.; Lomeli, M.; Hosseini, L.; Jégou, H. The Faiss library. *arXiv* **2024**, arXiv:cs.LG/2401.08281.
49. Papineni, K.; Roukos, S.; Ward, T.; Zhu, W.J. BLEU: A Method for Automatic Evaluation of Machine Translation. In Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, Philadelphia, PA, USA, 6–12 July 2002; pp. 311–318.
50. Lin, C.Y. ROUGE: A Package for Automatic Evaluation of Summaries. In *Text Summarization Branches Out*; Association for Computational Linguistics: Barcelona, Spain, 2004; pp. 74–81.
51. Yujian, L.; Bo, L. A normalized Levenshtein distance metric. *IEEE Trans. Pattern Anal. Mach. Intell.* **2007**, *29*, 1091–1095. [CrossRef]
52. Fletcher, S.; Islam, M.Z. Comparing sets of patterns with the Jaccard index. *Australas. J. Inf. Syst.* **2018**, *22*. [CrossRef]
53. Chiang, C.H.; Lee, H.y. Can large language models be an alternative to human evaluations? *arXiv* **2023**, arXiv:2305.01937.
54. Dubois, Y.; Li, C.X.; Taori, R.; Zhang, T.; Gulrajani, I.; Ba, J.; Guestrin, C.; Liang, P.S.; Hashimoto, T.B. Alpacafarm: A simulation framework for methods that learn from human feedback. *Adv. Neural Inf. Process. Syst.* **2024**, *36*.
55. Balaguer, A.; Benara, V.; de Freitas Cunha, R.L.; Estevão Filho, R.d.M.; Hendry, T.; Holstein, D.; Marsman, J.; Mecklenburg, N.; Malvar, S.; Nunes, L.O.; et al. RAG vs Fine-tuning: Pipelines, Tradeoffs, and a Case Study on Agriculture. *arXiv* **2024**, arXiv:2401.08406v3.
56. Martinez-Rodriguez, J.L.; Hogan, A.; Lopez-Arevalo, I. Information extraction meets the semantic web: A survey. *Semantic Web* **2020**, *11*, 255–335. [CrossRef]