International Journal of Geo-Information

MDPI

Article

Geospatial Knowledge-Base Question Answering Using Multi-Agent Systems

Jonghyeon Yang and Jiyoung Kim *

Social Eco Tech Institute, KonKuk University, Seoul 05029, Republic of Korea; yangjonghyeon@snu.ac.kr
* Correspondence: elliekim@konkuk.ac.kr

Abstract

Large language models (LLMs) have advanced geospatial artificial intelligence; however, geospatial knowledge-base question answering (GeoKBQA) remains underdeveloped. Prior systems have relied on handcrafted rules and have omitted the splitting of datasets into training, validation, and test sets, thereby hindering fair evaluation. To address these gaps, we propose a prompt-based multi-agent LLM framework (based on GPT-4o) that translates natural-language questions into executable GeoSPARQL. The architecture comprises an intent analyzer, multi-grained retrievers that ground concepts and properties in the OSM tagging schema and map geospatial relations to the GeoSPARQL/OGC operator inventory, an operator-aware intermediate representation aligned with SPARQL/GeoSPARQL 1.1, and a query generator. Our approach was evaluated on the GeoKBQA test set using 20 few-shot exemplars per agent. It achieved 85.49 EM (GPT-4o) with less supervision than fine-tuned baselines trained on 3574 instances and substantially outperformed a single-agent GPT-4o prompt. Additionally, we evaluated GPT-4o-mini, which achieved 66.74 EM in a multi-agent configuration versus 47.10 EM with a single agent. The observations showed that the multi-agent gain was higher for the larger model. Our results indicate that, beyond scale, the framework's structure is important; thus, principled agentic decomposition yields a sample-efficient, execution-faithful path beyond template-centric GeoKBQA under a fair, hold-out evaluation protocol.

Keywords: geospatial knowledge bases; knowledge-base question answering; large language models; multi-agent systems

Check for updates

Academic Editors: Wolfgang Kainz, Huayi Wu and Zhipeng Gui

Received: 2 October 2025
Revised: 19 December 2025
Accepted: 4 January 2026
Published: 8 January 2026

Copyright: © 2026 by the authors. Published by MDPI on behalf of the International Society for Photogrammetry and Remote Sensing. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

1. Introduction

The rapid evolution of large language models (LLMs) has fundamentally transformed natural-language processing (NLP), marking a paradigm shift toward more sophisticated human–computer interaction capabilities. Contemporary LLMs demonstrate exceptional performance across diverse language tasks and exhibit remarkable proficiency in language comprehension, reasoning, and text generation [1]. The emergent abilities observed in large-scale models, including complex multi-step reasoning and instruction following, have sparked unprecedented interest in applying these capabilities to specialized domains beyond general-purpose NLP [2].

In particular, in the geospatial domain, the integration of artificial intelligence (AI) within geographic information science, commonly termed geospatial AI (GeoAI), is rapidly emerging as a method to enhance spatial data analysis and address geographic problems [3,4]. This interdisciplinary domain encompasses the development of spatially explicit AI techniques aimed at addressing challenges in geographic-knowledge discovery

and spatial reasoning. Recent studies have initiated the integration of LLMs in GeoAI applications, suggesting significant potential; however, this field remains in the early exploratory phase [5,6].

One line of inquiry has examined the degree to which LLMs encode geographical knowledge. For example, Manvi et al. [7] demonstrated that LLMs embed rich spatial information about locations; however, they noted that simply prompting LLMs with coordinates is insufficient for accurate predictions. Accordingly, they proposed a GeoLLM approach that augments LLM prompts with auxiliary OpenStreetMap (OSM) data to predict variables such as population density and economic livelihoods, achieving performance on par with or exceeding that of satellite-based methods. These findings suggest that LLMs implicitly contain geospatial knowledge and can be effective for spatial tasks when they are provided with the appropriate context.

In addition to examining the geospatial knowledge inherently encoded in LLMs, recent research efforts have focused on explicitly extending LLM capabilities to geospatial applications through three primary approaches: (i) enabling LLM-driven use of geographic information system (GIS) tools [8,9], (ii) developing retrieval-augmented generation (RAG) [10] tailored to geospatial content [11], and (iii) implementing natural language interfaces to geospatial databases [12,13]. Across these approaches, studies have adopted different modeling strategies to operationalize LLMs for geospatial tasks. Some systems rely on task-specific fine-tuning of LLMs to equip them with GIS-specific knowledge and tool-use capabilities [8,9], whereas others employ multi-agent system architectures that decompose geospatial workflows into coordinated subtasks [12,13].

Despite the extensiveness of existing GeoAI research, geospatial knowledge-base (KB) QA (GeoKBQA) remains relatively underexplored. GeoKBQA, also termed factoid GeoQA, refers to the task of answering natural-language questions by querying a geographic KB, often via a formal query language such as GeoSPARQL. Traditional KBQA has been extensively studied in NLP; however, GeoKBQA poses additional challenges, such as interpreting spatial relationships and functions (e.g., “within 200 m” and “nearest to”). Early studies on GeoKBQA addressed these challenges by using rule-based semantic parsing. Punjani et al. [14] pioneered the first GeoKBQA system using handcrafted templates to map natural-language questions to GeoSPARQL queries. They introduced the GeoQuestions201 benchmark, which contains 201 geospatial questions paired with the corresponding SPARQL/GeoSPARQL queries. Although this template-based approach provides a proof of concept, it has limited flexibility and coverage; templates can handle only a narrow set of question patterns, and their effectiveness decreases with increasing question complexity. Subsequent studies have attempted to improve template robustness by incorporating neural network-based components. Hamzei et al. [15] employed deep neural networks (e.g., BERT-based encoders) to better understand question semantics; however, they continued to rely on template-driven query construction. This hybrid approach showed improved accuracy over purely manual rules but remained constrained by the underlying templates. More recently, Kefalidis et al. [16] expanded the GeoKBQA benchmark to 1089 questions (GeoQuestions1089) to cover a wider range of linguistic variations and spatial-reasoning challenges. In addition, they developed an enhanced version of their earlier GeoQA system [14], called GeoQA2. GeoQA2 demonstrated improved performance compared to the system described in [15]; however, some areas require further refinement. A key limitation of these systems, including GeoQA2, is their continued reliance on handcrafted rules, which limits their flexibility and scalability. This reliance on rule-based mechanisms highlights the need for a paradigm shift in GeoKBQA, analogous to the transition toward neural semantic parsers in general KBQA systems. In their comparative analysis, Kefalidis et al. [16] tested GPT-4 as an alternative to GeoQA2.

Although prompting-based methods using GPT-4 showed some potential, the difference in performance between the two approaches was not significant. Consequently, they opted to continue with the GeoQA2 system, leaving improvements in GPT-4 performance as a topic for future research. This finding further emphasizes the need for more advanced, adaptable solutions beyond template-based approaches, moving towards neural models capable of generalizing across diverse linguistic and spatial contexts.

A significant step in this direction was achieved by Yang et al. [17], who proposed the first fully neural GeoKBQA system. Rather than relying on templates, their system used a sequence-to-sequence model (T5) [18] to directly translate natural language into GeoSPARQL queries, incorporating a neural entity linker (ELQ) [19] to manage entities. In addition, they released a new GeoKBQA dataset comprising 4468 questions grounded in OSM data, marking a substantial departure from previous methods. This neural-based approach demonstrates the potential of leveraging deep learning over manual rule-based coding for geospatial QA. Crucially, Yang et al. [17] emphasized the importance of splitting the dataset into training, validation, and test sets for a fair evaluation, a practice that has been largely overlooked in earlier GeoKBQA approaches [14–16]. The datasets must be separated, even for rule-based systems, to ensure that the rules are derived solely from the training set rather than being tuned to overfit the entire dataset. This practice of dataset separation—which is well-established in machine learning, as articulated by Goodfellow et al. [20]—ensures that models are assessed based on their ability to generalize to unseen data.

In summary, although recent advances have extended LLM capabilities to geospatial applications—primarily through task-specific fine-tuning or multi-agent system designs—most existing GeoKBQA research still relies heavily on rule-based approaches for handling geospatial questions over geospatial knowledge bases. In this study, we propose a novel GeoKBQA approach that fills this gap by leveraging LLMs through a multi-agent system architecture, and we evaluate its performance on the test set used by Yang et al. [17] to ensure a fair comparison. Rather than fine-tuning a smaller dedicated model or relying on rigid templates, we orchestrate a team of LLM-based agents that collaborate to convert language questions into GeoSPARQL queries. Our approach was inspired by recent successes in applying multi-agent LLM frameworks to complex tasks, where decomposing a problem into specialized subtasks has significantly enhanced performance and interpretability [21,22]. Specifically, we design agents to handle different aspects of the query-translation process: one agent analyzes the user's question to decompose their intent (e.g., identifying if the query asks for a spatial relationship, count, specific attribute, etc.), whereas other agents focus on retrieving the relevant ontology concepts (concepts, properties, and spatial relations) and constructing subqueries. These agents communicate and pass partial outputs, and the final agent assembles a complete GeoSPARQL query. By decomposing the task, the system reduces the cognitive load on any single agent (or prompt) and allows for targeted prompts for each subtask. This multi-stage, multi-agent strategy for query generation draws from both the KBQA literature (e.g., the retrieval and generation framework for multi-stage query generation) and the emerging paradigm of multi-agent pipelines in GeoAI [12,13]. Our system follows a similar concept but is uniquely tailored to querying knowledge graphs via GeoSPARQL, which presents challenges such as mapping to a predefined ontology and adhering to the SPARQL syntax. To the best of our knowledge, this is the first approach that employs a multi-agent LLM framework for GeoKBQA rather than extensive fine-tuning.

The contributions of this study are as follows:

1. We propose a multi-agent system capable of converting natural language into GeoSPARQL queries.

2. We empirically demonstrate that multi-agent systems outperform single-agent systems, highlighting the necessity of multi-agent systems for GeoKBQA.
3. We design a multi-agent system pipeline based on domain knowledge in GIScience and the design principles of SPARQL/GeoSPARQL, and we conduct ablation studies to demonstrate the importance of each component.

The remainder of this paper is organized as follows: Section 2 reviews prior research related to QA over GeoKB and the transition from LLMs to multi-agent systems. Section 3 outlines the multi-agent system pipeline and provides detailed explanations of each agent in the process. Section 4 presents the experimental setup, results, and analysis of our system. Finally, Section 5 concludes the paper with a summary of our findings and suggestions for future research.

2. Related Works

2.1. LLMs for Geospatial Applications

Building on the broader trend of integrating LLMs into geospatial workflows, recent studies have explored several directions for enhancing their geospatial capabilities. These include enabling LLMs to operate GIS tools [8,9], leveraging RAG for domain-specific knowledge access [11], and developing natural-language interfaces for geospatial databases [12,13]. Several studies have fine-tuned LLMs or developed specialized models to operate GIS software tools, enabling the models to perform complex spatial analyses by invoking mapping and geoprocessing functions. This advancement addresses the key limitation of typical LLMs, which are trained on general text and lack knowledge of GIS-specific operations and data formats. For example, Wei et al. [8] introduced GeoTool-GPT, an LLaMA-2 [23] model fine-tuned with thousands of geospatial instructions, to master common GIS tasks. They curated a comprehensive instruction dataset and used instruction tuning to train the model for the use of GIS functions. The resulting GeoTool-GPT system can autonomously solve mapping problems by chaining tool operations, and its performance is comparable to that of GPT-4 [24] on expert-designed GIS benchmarks. Similarly, Zhang et al. [9] developed GTChain, a geospatial LLM trained using a self-instruct framework to generate multi-step tool-use sequences. Starting from a seed set of example tasks, they simulated diverse geospatial problems and their corresponding tool chains and subsequently fine-tuned LLaMA-2 on these data. The fine-tuned GTChain model demonstrated expert-level proficiency in tool use, effectively executing the provided GIS tools to solve tasks, achieving a success rate exceeding that of GPT-4 by more than 30% on a geospatial-task benchmark. These tool-using LLMs illustrate that with domain-specific training, LLMs can approach or even surpass general models, such as GPT-4, in specialized GIS workflows.

Another approach integrates LLMs with external text corpora and domain knowledge to improve geographic question answering (QA). Wang et al. [11] proposed GeoRAG, which couples an LLM with a geospatial-literature corpus through RAG [10]. They constructed a structured knowledge base from 3267 geography documents, including research papers and technical reports, categorized into key thematic dimensions of geographic knowledge. Upon receiving a query, GeoRAG uses a multi-step process: initially, a classifier identifies the topical dimension of the question, relevant documents are then retrieved from the corpus, and specially designed GeoPrompt templates inject the retrieved facts into the prompts of the LLM. The combination of prompt engineering and domain retrieval significantly improves the accuracy of geographic QA. Evaluations showed that GeoRAG outperformed standard LLM baselines, demonstrating more precise and context-rich answers and effectively validating the hypothesis that augmenting LLMs with domain-specific texts can overcome the knowledge gaps. The GeoRAG approach highlights the potential of retrieval

augmentation to provide LLMs with up-to-date geographic information and terminology, thereby significantly improving their performance in addressing spatially oriented queries.

The final example involves the use of LLMs to query and manipulate geospatial databases using natural languages. Researchers have combined LLM reasoning with tool use and searches to translate user questions into formal GIS database queries. For example, Peng et al. [12] developed a novel method that integrates RAG with a multi-agent LLM system to generate SQL queries for geospatial databases. Their system leverages GIS metadata and external knowledge to enrich the understanding of the LLM and employs a team-of-agents approach to break complex questions into manageable subtasks. Each subtask is handled by specialized agents; for example, one agent may identify relevant tables/attributes, another may formulate the query syntax, and the system ultimately assembles a final SQL query that reflects the user's intent. This strategy yields syntactically correct and contextually precise SQL queries, achieving over 80% accuracy in translating natural questions into SQL and outperforming single-LLM baselines on large-scale geospatial datasets. Similarly, Feng et al. [13] designed a multi-agent LLM framework for a GeoQA portal, enabling nonexperts to interact with geospatial data using everyday language. Their approach decomposes user queries into subtasks handled by different agents (e.g., locating relevant data and performing analysis) and uses a semantic search on geospatial data catalogs to retrieve information even when queries use informal terms. The system subsequently returns answers with supporting maps or descriptions while displaying the task plans of the agents for transparency. User studies have shown that this multi-agent LLM portal markedly lowers the barrier to geospatial data access, enabling users unfamiliar with GIS or SQL to obtain correct answers and visualizations for complex spatial questions.

2.2. Question Answering over Geospatial Knowledge Base

GeoKBQA has attracted growing interest in addressing place-centric questions that require nontrivial spatial-reasoning capabilities, which are not adequately supported by conventional factoid QA systems [14–17]. A foundational resource for early work was GeoQuestions201, a dataset interlinked with DBpedia, OSM, and the GADM database of global administrative areas, which provided an initial testbed for GeoKBQA. However, its scale and composition imposed clear limitations: only 201 questions were largely authored by third-year students, with scenarios skewed toward relatively simple geographic information needs. These characteristics constrained its usefulness for advancing methods to real-world heterogeneous queries and probing the ability of systems to generalize across varied linguistic forms and spatial operators. The subsequent GeoQuestions1089 dataset offered a substantial improvement in both volume and semantic/linguistic difficulty, introducing questions that required a richer natural-language understanding and explicit GeoSPARQL proficiency. However, methodological issues remained unresolved. Because both datasets relied on fully human-authored questions, they inherited restrictions in scalability, paraphrase diversity, and coverage of schema items and spatial relations (e.g., qualitative topological predicates and quantitative distance constraints). This manual curation is valuable in controlled settings. However, it limits the coverage and reduces the external validity of open-world evaluations.

To address these limitations, Yang et al. [17] introduced the GeoKBQA dataset, comprising 4468 geographically focused questions paired with gold-standard GeoSPARQL queries and entity annotations. Using GeoQuestions1089 as a seed, the authors constructed geospatial-question templates and systematically substituted entities, classes, and spatial functions to broaden the schema and operator coverage. Subsequently, they paraphrased the generated questions using ChatGPT (GPT-3.5-Turbo) to enrich the linguistic variability.

Once the templates were fixed, the pipeline became automated, yielding a substantially larger and more diverse corpus suitable for benchmarking and training neural models that benefit from scale. Equally important, unlike GeoQuestions201/1089, the dataset was explicitly partitioned into training, validation, and test sets, enabling fairer comparisons, discouraging handcrafted rule engineering on test items, and aligning GeoKBQA evaluation with standard machine learning practice. Below, we discuss the evaluation considerations pertinent to the system-level comparisons.

GeoKBQA is commonly formulated as a semantic parsing problem, where a natural-language question is translated into an executable SPARQL/GeoSPARQL query. Importantly, the nature of GeoKBQA is that a correct executable program typically requires joint grounding of multiple semantic elements that are explicitly represented in early GeoKBQA engines [14–16]: (i) instances (place/POI entities mentioned or implied by the question), (ii) concepts (feature types/classes such as restaurant, river), (iii) properties (attribute constraints and qualifiers, including numeric and comparative conditions), and (iv) geospatial relations/operators (topological predicates and distance-based constraints that must be evaluated over geometries). This joint grounding view makes GeoKBQA fundamentally a multi-component mapping problem, where errors in any one element can lead to non-executable or semantically invalid queries. Accordingly, prior systems have often adopted modular pipelines that explicitly identify these elements and then compose them into GeoSPARQL, providing a concrete basis for discussing both task requirements and methodological trade-offs.

Early work, notably [14], introduced one of the first geospatial QA engines, featuring a template-based translator that maps text to a predefined query template. Building on the Frankenstein framework [25], we establish a pipeline comprising a dependency parse/part-of-speech (POS) module, concept identifier, instance identifier, geospatial-relation identifier, SPARQL/GeoSPARQL query generator, and query executor. The dependency component produces POS tags and a dependency tree for each question. The concept identifier detects user-specified feature types and aligns them to classes in DBpedia, GADM, and the OSM ontology; for example, for the question "Which restaurants are near Big Ben in London?", restaurants were mapped to an osmo:Restaurant. The instance identifier recognizes the mentioned entities (e.g., Ireland, Dublin, or Shannon) by leveraging Stanford NER [26] and the AGDISTIS linker [27], which exploits hand-engineered features derived from the knowledge-graph structure and string similarity. The geospatial-relation identifier targets both qualitative predicates (e.g., borders) and quantitative constraints (e.g., at most 2 km from the specified location) via dictionary matching. Finally, the query generator assembles SPARQL/GeoSPARQL using handcrafted templates. Although effective for basic geospatial questions, this rule-centric design exhibits limited flexibility and coverage; it is not robust to paraphrasing and compositional variation, scales poorly to diverse schema items and spatial operators, and cannot generalize to more complex real-world queries.

Building on these foundations, Hamzei et al. [15] proposed a workflow that extracts a structured semantic encoding as the intermediate representation for compiling natural-language questions into SPARQL/GeoSPARQL. Importantly, the components that earlier systems implemented as separate identifiers (e.g., instance/concept/property/geospatial-relation identifiers) are realized in [15] as explicit fields in the extracted encodings. In particular, instances are captured via place names recognized in the question (supported by advanced NER models [28]), and concepts are captured via generic place types (i.e., type expressions) aligned with the ontology/schema. Geospatial relations are primarily captured through preposition encoding: preposition phrases that refer to place names or place types are recorded as candidate spatial relations (to be compiled into spatial predicates/operators). Properties are captured via adjective encoding as property qualities:

superlative/descriptive adjective phrases are encoded as place qualities if they refer to place names/types, and otherwise as property qualities, which provide property-related constraints for query compilation. Beyond these four core signals, Ref. [15] additionally encodes other operators needed for compositional questions—such as situation/activity distinctions from verbs (using BERT-based verb representations [29]), temporal relations anchored to dates, comparative operators, and conjunction/negation—so that the overall intent can be compiled into an executable GeoSPARQL query, although the final query construction still relies on a template-based compilation procedure.

GeoQA2 [16] explicitly follows the same GeoKBQA decomposition—jointly grounding instances, concepts, properties, and geospatial relations—and implements this structure with modules for concept identification, instance identification, geospatial-relation identification, and query generation/execution, augmented with constituency parsing and an explicit property identifier. To improve instance-level disambiguation [16], adopts TAGME [30], a feature- and rule-engineered entity linker that yields measurable gains in linking accuracy. Despite these refinements, GeoQA2 still relies heavily on handcrafted rules and templates across components, and its final SPARQL/GeoSPARQL queries are produced using predefined patterns. Consequently, the end-to-end performance remains sensitive to paraphrasing and compositional variation and scales poorly to heterogeneous schemas and complex geospatial operators.

Yang et al. [17] marked a clear departure from the above rule-centric pipelines by introducing the first fully neural GeoKBQA system: an entity retriever based on an off-the-shelf linker (ELQ) and a T5 sequence-to-sequence translator that directly maps questions to GeoSPARQL. By replacing handcrafted templates with a learned query generator, ref. [17] improves generalization when sufficient supervision is available and establishes a standardized evaluation protocol with explicit training/validation/test splits aligned with standard machine learning practice [20]. At the same time, this end-to-end translation paradigm shifts the practical bottleneck toward data availability: achieving high performance typically requires large-scale paired supervision and substantial manual effort in dataset construction (e.g., designing templates and systematically expanding coverage of entities/classes/functions before paraphrasing). In contrast, earlier systems [14–16] were carefully designed around the intrinsic GeoKBQA decomposition but depend predominantly on rule-based extraction and template-based compilation, which limits robustness and yields relatively modest end-to-end accuracy (often on the order of tens of percent, e.g., ~30–50% as reported in those works). This trade-off motivates our positioning: we retain the decomposition that reflects the nature of GeoKBQA (instances, concepts, properties, and geospatial relations) while aiming to substantially improve executable query accuracy under few-shot/low-supervision settings by leveraging modern LLM capabilities for structured grounding and reliable composition without requiring large-scale training data.

2.3. LLMS to Multi-Agent Systems

Early NLP systems were predominantly rule-based, relying on handcrafted linguistic rules that typically lacked scalability and robustness when faced with the variability of natural languages. The introduction of statistical models, such as n-gram language models [31], marked a paradigm shift by leveraging the statistical properties of languages derived from large corpora. However, these models are limited in handling long-range dependencies because of their fixed context windows, which consider only a limited number of preceding words.

Recurrent neural networks (RNNs) were introduced to surpass fixed-order models by sharing parameters over time and maintaining a recurrent state, thereby enabling variable-length context modeling in sequential data [32]. In practice, training RNNs with

backpropagation over time results in vanishing/exploding gradients, which hinder the learning of long-range dependencies [33,34]. Long short-term memory (LSTM) addresses this issue by introducing gated memory cells and a mechanism for nearly constant error flows, thereby enabling effective long-range credit assignments [35]. However, LSTMs remain inherently sequential, limiting parallelism and scaling to very long contexts; these constraints motivate attention mechanisms in RNNs [36].

A pivotal development that enabled modern LLMs is the transformer architecture [37]. Transformers replace recurrence with multi-head self-attention and position-wise feed-forward layers, eliminating the need for the sequential state updates found in LSTMs. Self-attention enables each token to consider all the others in the sequence, computes the attention weights that model pairwise dependencies, and supports long-range interactions. Attention is computed in parallel across positions with positional encodings providing order information; therefore, the architecture scales more effectively than RNNs and improves the modeling of complex sentences with nonlocal dependencies.

Building on the transformer, the field converges on pre-train-then-fine-tune paradigms at scale. BERT [29] pretrains a bidirectional encoder with masked language modeling and next-sentence-prediction objectives and attaches a lightweight task head for downstream fine-tuning. Bidirectional context modeling, conditioned on tokens to the left and right, yields significant gains across standard NLP benchmarks. In parallel, the GPT line adopts a decoder-only, left-to-right (causal language modeling) objective with a strong generative capacity. GPT-1 [38] showed that generative pre-training on a broad unlabeled corpus, followed by task-specific fine-tuning, can deliver substantial improvements. GPT-2 [39] further demonstrated that scaling the model size and data leads to robust zero-shot performance across diverse tasks (e.g., QA, machine translation, reading comprehension, and summarization) without explicit task supervision.

As decoder-only transformers scaled from GPT-1 to GPT-3, their performance improved consistently, and new behaviors emerged. Notably, GPT-3 [40], with its 175 B parameters, exhibited in-context learning, performing tasks from natural-language prompts without gradient updates. Consistent with this trend, scaling studies [41] showed that test loss follows smooth power-law relationships with respect to model size, dataset size, and computation. A computation-optimal trade-off for allocating parameters and data under a fixed budget was derived, providing a principled basis for predictable improvements from scaling.

Despite its strong generative capabilities, next-token prediction pre-training alone does not guarantee instruction following or alignment with human intent. Instruction tuning addresses this gap by fine-tuning collections of natural-language instructions paired with reference outputs. FLAN [42] fine-tunes encoder–decoder models, such as T5 [18], on a diverse mixture of instruction-formatted tasks, yielding substantial improvements in zero-shot generalization. However, instruction-only approaches depend on human-curated instruction–response pairs and continue to optimize a supervised proxy objective, which limits their coverage and scalability and does not fully capture user preferences. To better align the model behavior, InstructGPT [43] applies reinforcement learning from human feedback through a three-stage process: (i) supervised fine-tuning on the demonstration data, (ii) training of a reward model from pairwise human-preference comparisons, and (iii) policy optimization against the learned reward. This three-stage pipeline improves instruction adherence and preference alignment while reducing undesirable or off-task outputs.

GPT-4 [24] consolidates advances in scale, pre-training, and alignment, delivering strong gains in reasoning, problem solving, and instruction following across a wide range of benchmarks. Beyond raw accuracy, it exhibits improved robustness in long-form generation

and compositional tasks (e.g., mathematics, code synthesis, and multilingual QA) and has been presented as evidence of AGI-like potential, that is, broad competence emerging from a single, generalist model class rather than task-specific systems. These results provide a new reference point for the capability and reliability of LLMs and motivate methods that harness competence in complex, structured problems.

Simultaneously, a single-agent LLM remains a solitary policy conditioned on a prompt. For problems with a pronounced compositional structure, such an LLM cannot consistently plan, verify, and refine multi-step solutions. Therefore, recent studies have explored multi-agent LLM systems, in which multiple role-specialized agents coordinate to solve a task [21,22]. Common patterns include the division of labor (e.g., planner, solver, critic, and verifier), explicit decomposition of tasks into subgoals, transmission of natural-language messages over shared memory or a central controller, and iterative critique/debate followed by revision. This organization widens the search for solution strategies, reduces the cognitive load on any single policy, and enables cross-checking among complementary reasoning paths. These mechanisms are reported in surveys as beneficial for multi-step reasoning, complex QA, planning, and other structured tasks [21,22]. Although coordination overhead and evaluation remain open challenges, there is an emerging consensus that principled decomposition and role assignment can translate strong single-agent capabilities into more reliable performance in real-world, multistage problems. Our work follows this line of thought by structuring GeoKBQA into specialized agents aligned with subproblems of geospatial query generation.

3. Methodology

Our system is based on multi-agent systems, with each agent system playing a unique role in addressing the problems associated with translating natural language into GeoSPARQL queries.

As illustrated in Figure 1, the proposed framework comprises four main components: (1) an intent-analyzer agent, (2) retriever agents, (3) an operator-builder agent, and (4) a query-generator agent. The first stage involves the intent-analyzer agent, which analyzes the questions to extract the underlying intents. The second stage involves the retriever agents, each comprising a multi-grained retriever responsible for retrieving information regarding its role. Specifically, a concept retriever retrieves the concept information, a geospatial-relation retriever retrieves the information regarding geospatial relations, and a property retriever retrieves the necessary property information. The third stage involves an operator builder that follows the syntax of SPARQL and GeoSPARQL. The final stage involves a query generator that generates the target query based on the output of the previous agent. As shown in Figure 1, each agent uses the outputs of all the preceding stages and, in this context, performs its designated role.

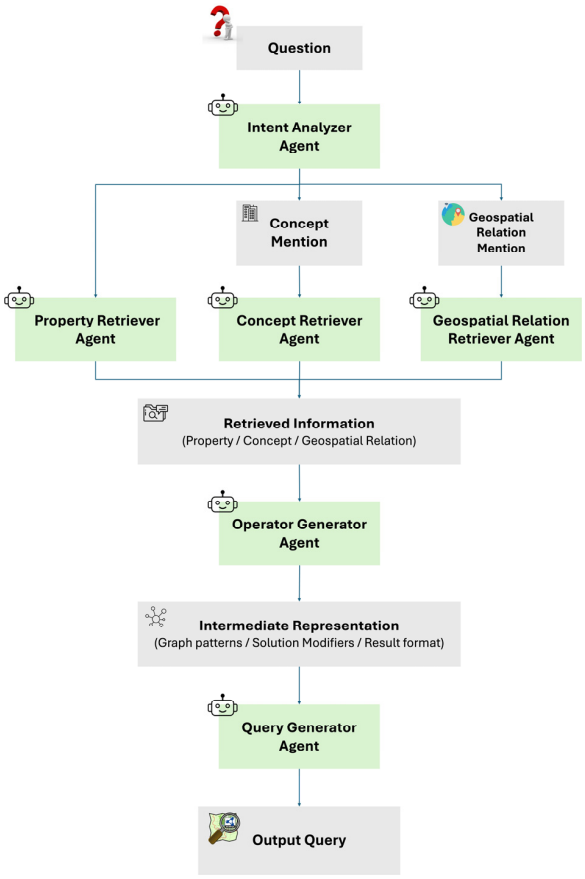


Figure 1. Overview of the proposed multi-agent GeoKBQA pipeline.

3.1. Intent-Analyzer Agent

First, the analyzer agent analyzes the input question and systematically generates the information to be used by the subsequent agents. It outputs WH-word cues, concept mentions, reference-entity mentions, and geospatial-relation phrases present in the input and question forms. WH-word cues provide an initial hypothesis about the expected answer type, thereby narrowing the downstream search space. The concept mention is the span in the question that denotes the feature class the user is querying (e.g., dam or restaurant), which in turn guides the concept retriever toward salient ontology classes and aliases. This division of labor ensures that the analyzer localizes linguistically meaningful spans, whereas the concept retriever focuses on ontology-level grounding in the concept inventory of the knowledge base. The geospatial-relation phrase captures the textual realization of spatial operators (e.g., near, within 200 m, distance between, and intersects) and provides a strong prior for the geospatial-relation retriever. Details of concept retrievers are provided in Section 3.2.1, and the geospatial-relation retriever is discussed in Section 3.2.2. Collectively, these outputs form a compact and structured specification that determines the remaining pipeline stages.

The reference-entity mention denotes the place names that serve as anchors for the spatial reasoning in each question. For example, in the query "Which dam is most distant from Bedok?", the target class is a dam, whereas the spatial anchor is Bedok, from which the distance is measured. Hence, Bedok is extracted as the reference-entity mention. In the query "What is the distance between Jurong Island and Novena?", the reference entities are Jurong Island and Novena, which are two distinct anchors because the requested value is a function of both locations. Question_form encodes the expected answer type primarily as {selection, numeric, boolean, literal, or geometry} and can be extended as required. This

attribute conditions operator synthesis and query generation; for example, it influences the choice of the query form (e.g., SELECT vs. ASK), projection variables, aggregation, and result formatting, thereby promoting type-consistent query construction.

In summary, before downstream agents execute their specialized roles, the analyzer produces clues and contexts that these agents should consider, effectively reducing ambiguity and improving retrieval precision. Figure 2 illustrates this process using the question "What is the distance between Jurong Island and Novena?" The WH-word is "what"; no concept mention is identified; the reference entities are Jurong Island and Novena; the geospatial-relation phrase is "the distance between", and the question form is numeric because the answer is a single distance value. These structured outputs flow to the retriever agents and subsequently to the operator builder and query generator, enabling faithful and well-typed GeoSPARQL generation while maintaining a clear separation of concerns across the multi-agent pipeline.

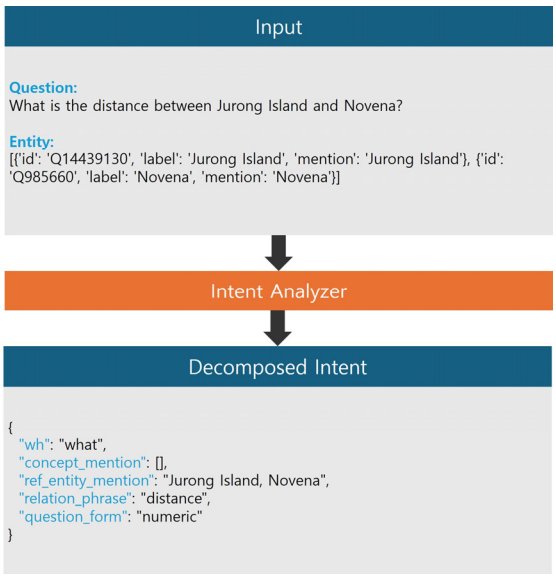


Figure 2. Illustrative example: the intent analyzer.

3.2. Multi-Grained Retriever Agents

This stage involves retrieving the task-critical symbols required downstream, namely, the concept, geospatial relation, and property, using LLM-driven prompting conditioned on the signals of the analyzer. In line with the GeoKBQA findings [14–16], an accurate GeoSPARQL synthesis depends on reliably grounding the target concept, identifying the operative spatial operator, and identifying the properties to be used for projection or filtering. We instantiate separate retrievers for each: the concept retriever grounds the target as an OSM tag key–value pair (e.g., osmkey:amenity = "restaurant"); the geospatial-relation retriever identifies the operator family (e.g., geof:distance) together with its argument mentions and any quantitative constraint stated in the question (comparator, threshold, unit); and the property retriever surfaces property IRIs referenced by attribute mentions for use as projection targets (SELECT) or filter predicates (FILTER). Figures 3–5 illustrate example outputs from the multi-grained retriever.



Figure 3. Illustrative example: the multi-grained retriever (containment query).



Figure 4. Illustrative example: the multi-grained retriever (distance-constraint "within" query).

Motivated by the evidence that LLMs encode rich geospatial knowledge [7], we obtained a small, high-recall candidate set via targeted prompting rather than pre-encoding large databases. Our approach dispenses with cross-encoder pipelines, which are common in KBQA [44,45] and require per-candidate forward passes, and bi-encoder setups [46], which precompute embeddings and maintain a vector index (vector store) for persistence and retrieval. Instead, our approach retains competitive recall through carefully scoped, analyzer-conditioned prompts. In short, retrievers provide a lightweight, ontology-aware interface: concepts arrive as (property-IRI, literal) tag assertions; geospatial relations are returned with their operator, argument mentions, and any associated constraints; and attributes are returned as property IRIs ready for projection or filtering, enabling downstream stages to assemble executable GeoSPARQL with minimal schema friction.



Figure 5. Illustrative example: the multi-grained retriever (property/attribute query).

3.2.1. Concept Retriever

The concept retriever grounds the feature type referenced in the question to the OSM tagging schema by returning an appropriate tag key–value pair rather than mapping to a separate class hierarchy. In particular, given a concept mention identified by the analyzer (e.g., park, restaurant, and hospital), the retriever produces a tuple of the form (concept_mention, property IRI, literal value), for example, ("park," osmkey:leisure, "park") or ("restaurant," osmkey:amenity, "restaurant"). This representation matches the manner in which concepts are encoded in the knowledge base (as OSM tag assertions) and can be inserted directly into basic graph patterns (BGPs) (for example, ?x osmkey:amenity "restaurant") without intermediate class mapping or schema translation.

Operationally, the retriever departs from the string-matching-based schema lookup used in prior GeoKBQA systems [14–16] and instead uses targeted prompting to align concept mentions with the OSM tagging schema. Rather than iterating through schema elements to locate a matching key–value pair, our approach relies on the LLM's internal spatial knowledge to infer the appropriate tag directly. This design is motivated by prior findings [7] indicating that LLMs implicitly encode rich geospatial information, enabling concept grounding without the need to query external resources. In our setting, the agent is guided to produce key–value tags consistent with the canonical OSM key–value inventory, and it outputs the appropriate tag for the mention (e.g., park → osmkey:leisure = "park," restaurant → osmkey:amenity = "restaurant"). This maintains the procedure faithful to the dataset—where mentions correspond directly to literal tag values—while replacing the rule-based "concept identifier" used in prior studies [14–16].

In the example shown in Figure 3, the question "What parks are included in Mutiara Rini?" yields the concept mention "park," which the concept retriever normalizes to the OSM tag osmkey:leisure = "park." Generally, each detected concept-mention retriever returns (i) the original mention span, (ii) a property IRI selected from the OSM key namespace, and (iii) the corresponding literal value from the OSM tag value inventory (with optional aliases, when helpful). This design maintains an interface that is ontology-aware yet schema-faithful to OSM, ensuring that concept grounding remains consistent with how entities are encoded in the knowledge graph.

3.2.2. Geospatial-Relation Retriever

Geospatial questions typically lexicalize either qualitative spatial relations (e.g., within, intersects, touches, and borders) or quantitative relations that invoke a metric (e.g., within 10 km of the distance between A and B). The geospatial-relation retrievers map such surface forms to their corresponding GeoSPARQL/OGC operators (e.g., geof:sfWithin, geof:sfIntersects, geof:sfTouches, geof:sfContains, and geof:distance) and return an operator specification accompanied by its argument mentions and, where explicitly expressed, a quantitative constraint consisting of a comparator, threshold, and unit. This operator-centric representation precisely provides the semantic predicate required to anchor spatial reasoning in the subsequent stages.

Methodologically, we employed targeted prompting instead of dictionary matching [14,16] or rule-based heuristics [15]. Unlike concept and property retrieval, which bind key–value tags or property IRIs, the geospatial case requires recovering the operand structure of a function, that is, determining which entities or concept-denoted sets populate the arguments of the operator. The agent is constrained to a canonical inventory of GeoSPARQL operators and units and extracts only what is overtly licensed by the question, thereby remaining faithful to the dataset while avoiding overinterpretation. This yields a compact type-aware specification that the operator builder can compile into valid functional applications within SPARQL/GeoSPARQL.

In the example illustrated in Figure 3, the input asks for parks included in Mutiara Rini; the phrase included in the input is mapped to the qualitative operator geof:sfContains with arguments instantiated by the grounded concept (park → osmkey:leisure "park") and the reference entity (Mutiara Rini), and no quantitative constraint is emitted. By contrast, in the example illustrated in Figure 4, the input seeks hospitals within 50 km of the Southern Islands; the retriever selects geof:distance, identifies arguments (hospitals, Southern Islands), and records the constraint as "<50,000 uom:metre." Importantly, selector semantics such as "closest to" or "farthest from" are not resolved at this stage; they are realized downstream via BIND to materialize a distance variable and ORDER BY/LIMIT within the solution modifiers, consistent with the separation of concerns of the pipeline.

3.2.3. Property Retriever

The property retriever locates the attribute referenced in the question and grounds it in the corresponding OSM property IRI, yielding the predicate–variable binding required for downstream projection (SELECT) or, when applicable, filtering (FILTER). Given an attribute mention (e.g., population, time zone, and name), the module returns a compact tuple (property_mention, property IRI, and value variable). By design, it does not bind the subject of the triple. It specifies only the predicate and object-side variables, allowing later stages to pair this binding with a subject introduced by concept grounding or entity resolution.

Methodologically, we replaced the pattern/dictionary matching commonly used in prior GeoKBQA studies [16] with the targeted prompting of an LLM constrained to the canonical OSM key inventory. The agent extracts the attribute phrase licensed by the question and aligns it to a single-property IRI, for example, by mapping the phrase corresponding to the population to osmkey:population. The retriever deliberately refrains from inferring solution modifiers or introducing implicit operators. Its role is strictly limited to supplying the predicate–object–variable pair that the subsequent components compile in SPARQL/GeoSPARQL.

Figure 5 illustrates the Malaysian population. The attribute mention "how many individuals live in" is aligned to the property IRI osmkey:population, and the retriever emits the tuple ("how many individuals live in," osmkey:population, ?population) such that the query generator can project a scalar value (for example, SELECT ?population). This

behavior is representative of the scope of the module; it surfaces property IRIs referenced by attribute mentions together with an object-side value variable while leaving the subject bound elsewhere in the pipeline, thereby providing a clean, narrowly scoped interface for downstream query construction.

3.3. Operator-Builder Agent

The operator builder is the pipeline's intermediate representation (IR) layer, which consolidates the outputs from the analyzer and retrievers into a domain-specific language (DSL) whose syntax and evaluation follow SPARQL 1.1 [47] and GeoSPARQL 1.1 [48]. Assuming a fixed default dataset, the DSL explicitly encodes graph patterns, solution modifiers, and the result form and further refines graph patterns into two strata aligned with the standard execution model. The first stratum, pattern.basic, contains BGPs realized as SPO triples for entity anchoring and concept constraints, such as OSM tag assertions, property assertions, and geometry linking (e.g., via geo:hasGeometry). The second, pattern.advanced, comprises the remaining WHERE clause constructs: GeoSPARQL-extension functions are used in expressions (e.g., geof:distance), BIND materializes the derived variables, and FILTER imposes thresholds. Solution modifiers are recorded separately and applied in the canonical order (ORDER BY, DISTINCT/REDUCED, OFFSET, LIMIT), whereas the result form captures the query type and projection consistently with those modifiers. This organization yields an IR that is faithful to the SPARQL/GeoSPARQL evaluation and, crucially for geospatial semantics, respects the division between the GeoSPARQL vocabulary used as triples and the GeoSPARQL functions invoked in expressions.

Methodologically, the DSL follows the established KBQA practice of compiling natural language into a structured IR prior to query synthesis. Prior work has shown that S-expressions and graph-style logical forms provide a compact, executable abstraction for KBQA over general KBs [44,45] and that first-order logic (FOL) forms serve as a robust intermediate in GeoKBQA settings [15]. We adopted this IR principle but specialized it to SPARQL/GeoSPARQL: instead of a domain-agnostic logical form, the operator builder produces a typed, operator-centric program that maps one-for-one to BGPs, group-pattern expressions, solution modifiers, and result forms. Concretely, the multi-grained retrieval is reified as minimal operators: E (entity), C (concept via OSM key–value), P (property for projection or filtering), and R (geospatial relation as a function call). The builder assembles these into pattern.basic and pattern.advanced while enforcing variable hygiene (introducing and reusing geometry variables), scoping for BIND/FILTER, and deferring selector semantics such as "closest/farthest" to the solution-modifier stage, where they are realized by ordering over a bound measure with an optional limit. The result is an interpretable IR—each operator has a direct semantic counterpart in SPARQL/GeoSPARQL and an executable one because the downstream Query Generator can deterministically linearize it into a standards-compliant query without ad hoc rules.

In the running example illustrated in Figure 6, for the question "Which dam is the most distant from Bedok?," the operator builder receives the reference entity with its geometry, concept tag osmkey:waterway = "dam," and geospatial relation family "distance" without an explicit threshold. These are compiled into pattern.basic triples for entity and concept grounding with geometry bindings, an R call to geof:distance with a BIND that materializes a distance variable in pattern.advanced, and a selector encoded as ORDER BY DESC with LIMIT 1 in solution_modifiers. The result_format records a SELECT form with the intended projection and distinctness. In effect, the operator builder condenses heterogeneous natural-language signals into a compact, standard-aligned IR similar to S-expression/FOL intermediates, now specialized in GeoSPARQL, thereby providing the query generator with a lossless, verifiable blueprint for producing executable geospatial queries.

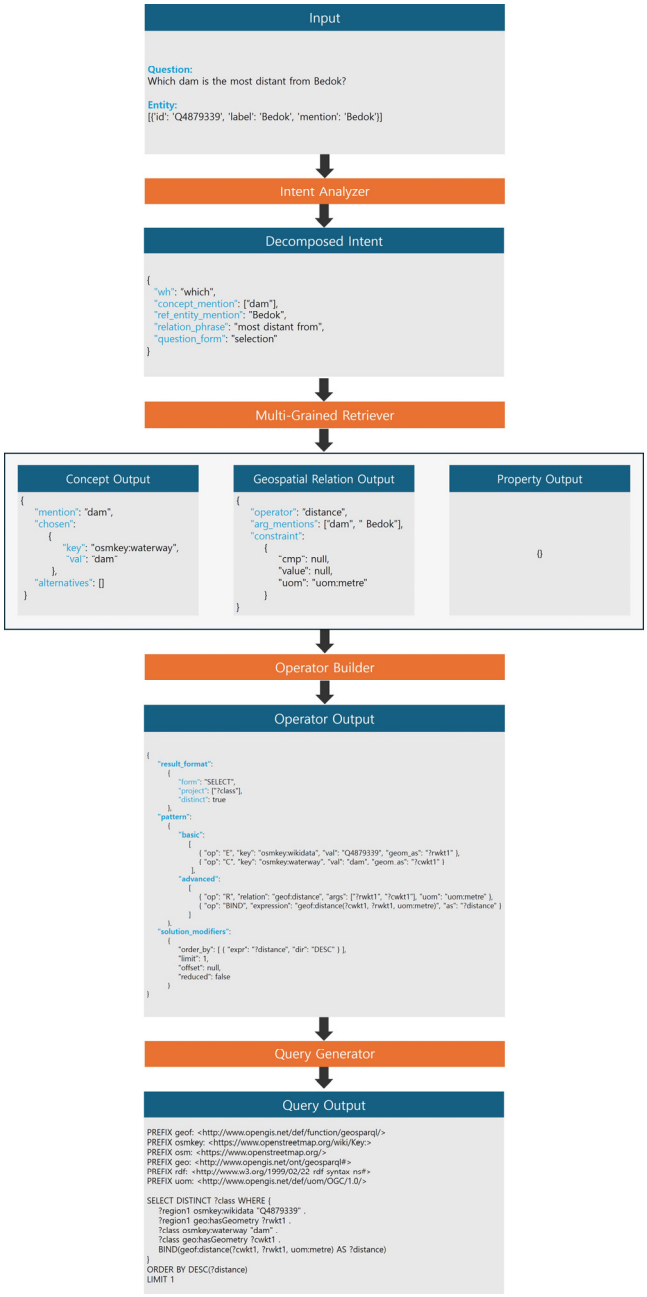


Figure 6. Running example of our multi-agent systems.

3.4. Query-Generator Agent

The query generator operates at the terminal stage of the pipeline. It consumes the structured representation produced by the operator builder and converts it into a fully specified GeoSPARQL query. At this point, all upstream ambiguities have been resolved; concepts have been grounded in OSM tag assertions, geospatial relations have been mapped to functions with argument bindings, properties have surfaced for projection or filtering, and operator composition has been organized into a well-typed DSL. Therefore, the role of the generator is not to reinterpret the input but to faithfully linearize it into standard GeoSPARQL syntax, ensuring compliance with the SPARQL 1.1 evaluation rules and GeoSPARQL 1.1 extensions. To this end, the generator systematically declares the required prefixes, instantiates the WHERE clause with the pattern.basic and pattern.advanced components, and applies the modifiers and projections as prescribed by the DSL. This guarantees that the queries are syntactically valid, semantically consistent with the outputs

of the retrievers, and executable on off-the-shelf GeoSPARQL-compliant endpoints without further adjustment.

As the operator builder already enforces a division between the BGPs, advanced expressions, and solution modifiers, the query generator primarily functions as a code emitter. It traverses the DSL in canonical order—prefix declarations, result format, WHERE block, and solution modifiers—and instantiates the variables and function calls exactly as designated. More importantly, the generator introduces no additional inference; its contribution lies in ensuring determinism and fidelity to the standards. Therefore, every operator in the DSL has a one-to-one correspondence with a clause or expression in the final query. This design choice maximizes interpretability; reviewers can verify the correctness of a query by comparing the DSL to the linearized form, and downstream systems can reuse the same DSL for alternative backends (e.g., query optimization or explanation interfaces) without any loss of information.

As illustrated in Figure 6, for the question "Which dam is the most distant from Bedok?", the query generator receives a DSL, where the result_format specifies a SELECT DISTINCT projection of ?class, pattern.basic encodes entity and concept bindings (Bedok via osmkey:wikidata, dams via osmkey:waterway), pattern.advanced introduces the distance function and binds its output distance, and solution_modifiers requests a descending order and top-1 selection. The generator converts this faithfully into the GeoSPARQL query shown in Figure 6: prefix declarations, a WHERE clause with entity and concept triples plus distance binding and an ORDER BY/LIMIT clause. This example demonstrates how the generator consolidates the cumulative reasoning of prior agents into an executable artifact, thereby closing the loop from natural-language questions to standard-compliant queries that can be executed against geospatial knowledge bases. In this manner, the query generator serves as the final bridge between the multi-agent interpretation and verifiable system output.

4. Experimental Results and Analysis

4.1. Setup

We conducted experiments on the GeoKBQA dataset presented in [17], which, unlike earlier resources [14,16], provides explicit training, validation, and test splits, enabling a fair out-of-sample evaluation. In line with prior work [14–17], we focus on well-formed geospatial questions whose intent is sufficiently specified to be translated into executable GeoSPARQL. Ill-defined or underspecified queries are outside the scope of this study. All the agents operate under a few-shot prompting regime (prompts are provided in Appendix A). Each agent receives 20 in-context examples randomly sampled from the training split. We assumed gold reference entities to isolate the contribution of our multi-agent pipeline from entity linking. The agent workflow was implemented using LangChain. For GPT-4o and GPT-4o-mini, we set the temperature to 0 to ensure reproducibility. Following a previous study [17], we report the exact match (EM) as the primary metric.

4.2. Overall Performance

The results are presented in Table 1. Our multi-agent pipeline achieved 85.49 and 66.74 EM with GPT-4o and GPT-4o-mini, respectively. Fine-tuned baselines from [17] demonstrated competitive or superior performance under full-data supervision: T5-Base achieved 71.65 EM, T5-Large achieved 79.02 EM, and CodeT5+ (770M) achieved 94.20 EM, with the latter benefiting from pretraining targeted at code-like generation. Crucially, these fine-tuned models were trained on 3574 curated GeoKBQA training examples, whereas our multi-agent setup used only 20-shot in-context exemplars per agent, that is, <1% of the supervision used in [17]. Therefore, the gap reflects not only the model choice but also

the supervision scale and inductive bias: CodeT5+ excels at supervised, syntax-sensitive sequence mapping, whereas our approach emphasizes sample efficiency and generalization with minimal guidance. Despite the stark differences in the labeled data, GPT-4o augmented with our agentic pipeline remains competitive, indicating that structured decomposition plus retrieval can recover much of the performance of fully fine-tuned systems under stringent EM evaluations.

Table 1. Results on the GeoKBQA test set.

	EM
Fine-tuning based methods (Fine-tuned on all training sets–3574, results taken from [17])	
T5-base	71.65
T5-Large	79.02
Code-T5+ (770M)	94.20
Prompting-based methods (Used 20 examples)	
Multi-agent System (GPT-4o-mini)	66.74
Multi-agent System (GPT-4o)	85.49

4.3. Multi-Agent System vs. Single-Agent System

Table 2 compares the single- and multi-agent configurations. For the single-agent system, we adopt the approaches of [16] for GeoKBQA. Using GPT-4o, the multi-agent pipeline reached 85.49 EM compared to 55.36 EM for a single agent (absolute increase of 30.13). Using GPT-4o-mini, the multi-agent system attained 66.74 EM compared with 47.10 EM for a single agent (absolute increase of 19.64). These gains indicate that decomposition into specialized agents paired with explicit operator construction substantially improves execution-faithful GeoSPARQL synthesis compared to treating the task as a single-shot prompting problem. The single-agent results are consistent with prior observations [18] that general-purpose LLMs underperform on geospatial QA when using single-agent prompting. This reinforces the idea that agentic factorization is more effective than relying exclusively on the raw model's scale.

Table 2. Comparison of the performance of multi- and single-agent systems.

	EM
Single-agent system (GPT-4o-mini)	47.10
Single-agent system (GPT-4o)	55.36
Multi-agent system (GPT-4o-mini)	66.74
Multi-agent system (GPT-4o)	85.49

The model size remains important; however, its effect is amplified by an agentic pipeline. Transitioning from GPT-4o-mini to GPT-4o improved the single-agent baseline by approximately +8.26 EM (from 47.10 to 55.36 EM), whereas the same increase in capacity within the multi-agent system yielded an improvement of approximately +18.75 EM (from 66.74 to 85.49). Thus, scaling the backbone proves beneficial. However, scaling the architecture using role-specialized analyzers, retrievers, and an operator-aware builder consumed by a deterministic generator delivers disproportionate returns. This pattern suggests that for GeoKBQA, structured decomposition and retrieval are first-order drivers of exact-match accuracy, with the model size providing secondary, compounding benefits.

4.4. Multi-Agent System Ablation

Table 3 presents the ablation results for the multi-agent pipeline. Removing the operator builder yielded only a marginal EM drop of approximately 0.5%, indicating that

the query generator can often translate natural language directly from the analyzer and multi-grained retrievers into valid GeoSPARQL. This indicates that the operator layer is not the primary driver of raw EM under the conditions of our benchmark. Its value lies in its structural and explanatory roles: it imposes a clean IR that improves interpretability, facilitates the localization of failure modes, and—by explicitly distinguishing basic and advanced graph patterns—offers potential for compositional queries where scoped BIND/FILTER and multi-step operator assembly are more critical. We expect its impact to grow with query complexity, a direction we intend to explore in future stress tests on more challenging GeoKBQA regimes.

Table 3. Ablation results for the multi-agent system.

	EM
MAS (GPT-4o)	85.49
w/o Operator	85.04
w/o Retriever	59.15
w/o Retriever and Operator	67.63
Single Agent System (GPT-4o)	55.36

In contrast, ablating the multi-grained retrievers yields a large degradation to 59.15 EM, representing an increase of only approximately 4 EM over the single-agent GPT-4o baseline (55.36). This establishes the retrievers as the dominant contributors to exact-match accuracy, as reliably grounding concepts, properties, and geospatial relations is essential for generating executable, schema-faithful queries. Interestingly, among the configurations that lack a retriever, the intent-analyzer–only system achieves 67.63 EM, outperforming the 59.15 EM obtained when the operator is added. In other words, when no retrieved context is available, introducing an operator stage worsens performance rather than improving it. A plausible explanation is that the operator builder, when not grounded by high-quality retrieved information, can amplify upstream errors and propagate them through later stages, whereas directly compiling queries from the analyzer can sometimes avoid additional points of failure. Notably, the 67.63-EM configuration also exceeds the single-agent baseline (55.36), indicating that even without retrievers, the intent analyzer provides valuable structural guidance that meaningfully improves GeoSPARQL synthesis over a single agent.

Taken together, these results indicate that (i) retrieval quality is the primary bottleneck, (ii) the operator builder contributes mainly to interpretability rather than raw EM, and (iii) effective multi-agent systems require careful architectural design rather than the indiscriminate addition of agents.

4.5. Case Study

Table 4 presents a case study (the prefixes are omitted for brevity). The single-agent system produces a syntactically valid and executable query; however, it is semantically imprecise: instead of constraining candidates to building = “church”, it uses amenity = “place_of_worship”, thereby broadening the result set to include churches, mosques, temples, and other worship sites. This mismatch violates the user intent—“Which church is the farthest from Perling?”—because the ranking over distance is applied to an over-inclusive candidate pool. In contrast, our multi-agent pipeline correctly grounds the concept to osmkey:building “church”, preserving the intended class constraint and yielding the faithful top-1 result under the “farthest” selector.

Table 4. Case study of concept grounding—multi-agent versus single-agent.

Item	Description
Input	Which Church Is the Farthest from Perling? [{“Id”: “Q13548361”, “Label”: “Perling”, “Mention”: “Perling”}]

Ground-Truth	SELECT DISTINCT ?class WHERE {
	?region1 osmkey:wikidata “Q13548361”.
	?region1 geo:hasGeometry ?rwkt1.
	?class osmkey:building “church”.
	?class geo:hasGeometry ?cwkt1.
	BIND(geof:distance(?cwkt1, ?rwkt1, uom:metre) AS ?distance)
	}
	ORDER BY DESC(?distance)
	LIMIT 1

Single-Agent System Output	SELECT DISTINCT ?class WHERE {
	?region1 osmkey:wikidata “Q13548361”.
	?region1 geo:hasGeometry ?rwkt1.
	?class osmkey:amenity “place_of_worship”.
	?class geo:hasGeometry ?cwkt1.
	BIND(geof:distance(?cwkt1, ?rwkt1, uom:metre) AS ?distance)
	}
	ORDER BY DESC(?distance)
	LIMIT 1

Multi-Agent System Output	SELECT DISTINCT ?class WHERE {
	?region1 osmkey:wikidata “Q13548361”.
	?region1 geo:hasGeometry ?rwkt1.
	?class osmkey:building “church”.
	?class geo:hasGeometry ?cwkt1.
	BIND(geof:distance(?cwkt1, ?rwkt1, uom:metre) AS ?distance)
	}
	ORDER BY DESC(?distance)
	LIMIT 1

Intent Output	{
	“wh”: “which”,
	“concept_mention”: [“church”],
	“ref_entity_mention”: “Perling”,
	“relation_phrase”: “farthest from”,
	“question_form”: “selection”
	}

Concept Output	{
	“concepts”: [
	{
	“mention”: “church”,
	“chosen”: {
	“key”: “osmkey:building”,
	“val”: “church”
	},
	“alternatives”: []
	}
	]
	}

Table 4. *Cont.*

Item	Description
Geospatial-Relation Output	{ "geospatial_relations": [{ "operator": "distance", "arg_mentions": ["church", "Perling"], "constraint": { "cmp": null, "value": null, "uom": "uom:metre" } }] }
Property Output	{ "properties": [] }
Operator Output	{ "result_format": { "form": "SELECT", "project": ["?class"], "distinct": true }, "pattern": { "basic": [{ "op": "E", "key": "osmkey:wikidata", "val": "Q13548361", "geom_as": "?rwkt1" }, { "op": "C", "key": "osmkey:building", "val": "church", "geom_as": "?cwkt1" }], "advanced": [{ "op": "R", "relation": "geof:distance", "args": ["?cwkt1", "?rwkt1"], "uom": "uom:metre" }, { "op": "BIND", "expression": "geof:distance(?cwkt1, ?rwkt1, uom:metre)", "as": "?distance" }] }, "solution_modifiers": { "order_by": [{ "expr": "?distance", "dir": "DESC" }], "limit": 1, "offset": null, "reduced": false } }

Table 5 highlights an error in the selection of the geospatial operator (prefixes omitted). The single-agent system maps the phrase "shares borders with" to geof:sfIntersects, which is excessively permissive; the operator "intersects" applies when two geometries overlap or merely touch, potentially admitting rivers that cross into or overlap the interior of Teluk Danga rather than only those that meet it along a boundary. In contrast, the multi-agent pipeline correctly selects geof:sfTouches, whose topological semantics capture boundary contact without interior overlap—precisely aligning with the intended meaning of "shares borders with." Because both systems correctly restrict candidates to osmkey:waterway "river" and bind geometries, the difference in answers is entirely attributable to the selected

geospatial operator. Consequently, multi-agent mapping yields a semantically faithful FILTER and a higher-precision candidate set that aligns with the query.

Table 5. Case study of geospatial relation selection—multi-agent versus single-agent.

Item	Description
Input	Can You Tell Me Which River Shares Borders with Teluk Danga? [{“Id”: “Q5215956”, “Label”: “Teluk Danga”, “Mention”: “Teluk Danga”}]

Ground-Truth	SELECT DISTINCT ?class WHERE {
	?region1 osmkey:wikidata “Q5215956”.
	?region1 geo:hasGeometry ?rwkt1.
	?class osmkey:waterway “river”.
	?class geo:hasGeometry ?cwkt1.
	FILTER(geof:sfTouches(?cwkt1, ?rwkt1))
	}

Single-Agent System Output	SELECT DISTINCT ?class WHERE {
	?region1 osmkey:wikidata “Q5215956”.
	?region1 geo:hasGeometry ?rwkt1.
	?class osmkey:waterway “river”.
	?class geo:hasGeometry ?cwkt1.
	FILTER(geof:sfIntersects(?cwkt1, ?rwkt1))
	}

Multi-Agent System Output	SELECT DISTINCT ?class WHERE {
	?region1 osmkey:wikidata “Q5215956”.
	?region1 geo:hasGeometry ?rwkt1.
	?class osmkey:waterway “river”.
	?class geo:hasGeometry ?cwkt1.
	FILTER(geof:sfTouches(?cwkt1, ?rwkt1))
	}

Intent Output	{
	“wh”: “which”,
	“concept_mention”: [“river”],
	“ref_entity_mention”: “Teluk Danga”,
	“relation_phrase”: “shares borders with”,
	“question_form”: “selection”
	}

Concept Output	{
	“concepts”: [
	{
	“mention”: “river”,
	“chosen”: {
	“key”: “osmkey:waterway”,
	“val”: “river”
	},
	“alternatives”: []
	}
	]
	}

Table 5. *Cont.*

Item	Description
Geospatial-Relation Output	{ "geospatial_relations": [{ "operator": "sfTouches", "arg_mentions": ["river", "Teluk Danga"], "constraint": { "cmp": null, "value": null, "uom": null } }] }
Property Output	{ "properties": [] }
Operator Output	{ "result_format": { "form": "SELECT", "project": ["?class"], "distinct": true }, "pattern": { "basic": [{ "op": "E", "key": "osmkey:wikidata", "val": "Q5215956", "geom_as": "?rwkt1" }, { "op": "C", "key": "osmkey:waterway", "val": "river", "geom_as": "?cwkt1" }], "advanced": [{ "op": "R", "relation": "geof:sfTouches", "args": ["?cwkt1", "?rwkt1"] }, { "op": "FILTER", "condition": "geof:sfTouches(?cwkt1, ?rwkt1)" }] }, "solution_modifiers": { "order_by": [], "limit": null, "offset": null, "reduced": false } }

Tables 4 and 5 show that single-agent LLMs internalize nontrivial geospatial regularities, often yielding syntactically valid, executable queries and broadly reasonable answers; however, they frequently fail to capture fine-grained intent. In both cases, the single agent chose a plausible but semantically misaligned grounding (amenity = "place_of_worship" instead of building = "church" and geof:sfIntersects instead of geof:sfTouches), which expanded or skewed the candidate set and, therefore, the final result. The multi-agent pipeline corrects these errors by isolating concept grounding, spatial-operator selection, and selector semantics into dedicated steps, subsequently compiling them into a standards-faithful IR. This decomposition reliably preserves the intended constraint and relation of the user, producing answers that are not only executable but also intent-consistent.

5. Conclusions

We presented a GeoKBQA system that operationalizes language models as a coordinated team of role-specialized agents—an analyzer, a set of multi-grained retrievers, an operator builder, and a query generator—to translate language questions into executable GeoSPARQL. The core design choices are principled and standards-aligned: concepts are grounded to OSM key–value tags rather than abstract classes; geospatial relations are mapped to GeoSPARQL/OGC functions with an explicit argument structure and optional numeric constraints; and properties are surfaced as projection/filter predicates. These elements are compiled into a compact domain-specific IR that mirrors SPARQL 1.1/GeoSPARQL 1.1 evaluation (basic versus advanced graph patterns, result form, and solution modifiers). This decomposition yields an interpretable, schema-faithful pipeline in which the upstream ambiguity is resolved before code emission, allowing the query generator to linearize the IR into queries that are syntactically and semantically compliant with off-the-shelf geo-triplestores.

Empirically, on the GeoKBQA benchmark of [17], the multi-agent pipeline achieved 85.49 EM with GPT-4o using only 20-shot in-context exemplars per agent (<1% of the 3574 samples used to fine-tune baselines). Our results showed that although fully supervised models, particularly CodeT5+ (770M), remain strong in the abundant-label regime (up to 94.20 EM), structured factorization and retrieval can recover much of their performance with minimal supervision. Because our method uses only standard prompting strategies, future work could explore more advanced techniques—such as chain-of-thought prompting—which may further enhance the effectiveness of our multi-agent system. Comparisons against single-agent prompting highlighted the architectural gains: +30.13 EM with GPT-4o and +19.64 EM with GPT-4o-mini, indicating that role specialization and operator-aware IR are more decisive than the raw model scale for execution-faithful GeoSPARQL synthesis. Ablation studies further clarified the sources of accuracy: removing the operator builder changed the EM by only approximately 0.5%, underscoring its primary value for interpretability and compositional potential, whereas removing the retrievers caused a large decrease to 59.15 EM, establishing concept/property/geospatial relation grounding as a critical path. Note that these results assume gold entity annotations and use string-based Exact Match (EM) on the generated GeoSPARQL queries; thus, the reported EM reflects query synthesis under gold grounding, not fully end-to-end GeoKBQA including entity linking.

Our study argues that progress in GeoKBQA depends not only on larger models or more labels but also on a better structure. A small set of well-chosen agents, a retrieval layer that speaks the ontology language, and an IR that respects SPARQL/GeoSPARQL semantics collectively yield an interpretable, sample-efficient, and competitively accurate pipeline. This result carries several important implications. Although our approach is not as strong as fully fine-tuned models, it operates with only 20 in-context examples—compared to the 3574 training instances required by supervised baselines—making it particularly attractive in low-label settings. Moreover, as demonstrated in our ablations, the operator-aware design offers interpretability benefits that fine-tuned models generally lack. Finally, because the system requires only a small number of exemplars rather than task-specific training, it is well suited for rapid deployment in new domains. We anticipate that the released prompts will stimulate further research on more complex geospatial reasoning by enabling the development of more advanced multi-agent systems for reliable GeoKBQA at scale. Moreover, given our assumption of gold entity information, an important next step involves the seamless integration of LLM-based entity linking into the multi-agent architecture.

Author Contributions: Conceptualization, Jonghyeon Yang and Jiyoung Kim; methodology, Jonghyeon Yang and Jiyoung Kim; software, Jonghyeon Yang; validation, Jonghyeon Yang and Jiyoung Kim; formal analysis, Jonghyeon Yang and Jiyoung Kim; investigation, Jonghyeon Yang and Jiyoung Kim; resources, Jiyoung Kim; data curation, Jonghyeon Yang; writing—original draft preparation, Jonghyeon Yang; writing—review and editing, Jonghyeon Yang and Jiyoung Kim; visualization, Jonghyeon Yang; supervision, Jiyoung Kim; project administration, Jiyoung Kim; funding acquisition, Jiyoung Kim. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the Public Demand-Oriented Customized Living Safety R&D Program (Phase II) funded by the Ministry of the Interior and Safety (MOIS, Republic of Korea), grant number RS-2023-00241703.

Data Availability Statement: The original contributions presented in this study are included in the article. Further inquiries can be directed to the corresponding author.

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

Abbreviations

The following abbreviations are used in this manuscript:

LLM	Large language model
AI	Artificial intelligence
NLP	Natural-language processing
GeoAI	Geospatial artificial intelligence
GIS	Geographic information system
RAG	Retrieval-augmented generation
QA	Question answering
SQL	Structured Query Language
KB	Knowledge-base
GeoKBQA	Geospatial Knowledge-base Question Answering
ELQ	Entity Linking for Questions
SLM	Small language model
OSM	OpenStreetMap
POS	Parse/part-of-speech
RNN	Recurrent neural network
LSTM	Long short-term memory
IR	Intermediate representation
DSL	Domain-specific language
BGP	Basic graph pattern
FOL	First-order logic

Appendix A

Prompt Template for Multi-Agent System

We demonstrate the prompt templates for our multi-agent system implemented in LangChain. Each figure shows the role of an agent, the structure of its output, and its dependencies on the outputs of upstream agents. Throughout the prompts, double asterisks (**) denote emphasized/important text.

Figure A1 presents the prompt template for the intent analyzer, which extracts WH cues, concept mentions, reference-entity mentions, geospatial-relation phrases, and the question form (selection, numeric, boolean, literal, or geometry). Its structured output conditions all downstream agents.

Figure A2 presents the prompt template for the concept retriever, which is conditioned on the concept mentions of the intent analyzer and aligns them with the OSM tagging scheme by returning a tuple of the form (mention, property IRI, literal value).

Prompt Template for Intent Analyzer

```
You are the **Intent Decomposer** in a multi-agent NL→GeoSPARQL pipeline.
INPUTS:
- A natural-language question.
- A list of resolved entities (mention=id). Treat them as hints.

OUTPUT:
Return a STRICT JSON object with EXACTLY these keys (in this order):
- "wh": lowercased wh-token appearing in the question (e.g., "what", "which", "how many", "is", "are", "can").
- "concept_mention": array of EXACT substrings (spans) from the question that denote conceptual types to be mapped by the concept retriever (use exact spans; no normalization).
- "ref_entity_mention": a single string containing the reference entity mention(s) AS THEY APPEAR in the question. If multiple, join in question order with ", " (use exact spans).
- "relation_phrase": the EXACT substring (span) in the question that expresses the geospatial relation (e.g., "most distant from", "closest to", "distance between", "less than 50km away from", "intersects with", "included in", "near", "close to"); preserve exactly as written.
- "question_form": one of {"selection","numeric","boolean","literal","geometry","route","others"}):
    * selection — return one/many items (e.g., "which...", "closest...", "most distant...").
    * numeric   — return a number/value (e.g., distance, "how many...").
    * boolean   — yes/no existence or proximity (e.g., "is X close to Y?").
    * literal   — non-numeric attribute value (e.g., capital, name).
    * geometry  — return the geometry/shape/location (e.g., WKT).
    * route     — network path/cost/time (e.g., shortest path, travel time).
    * others    — none of the above.

ADDITIONAL GUIDELINES:
- If a field truly has no content, use [] for arrays and "" for strings.

FEW-SHOT EXAMPLES (do not quote or refer in output):
{few_shot_examples}

CURRENT INPUT:
Question:\n{question}
Entities:\n{entities_str}
Output(strict JSON):
```

Figure A1. Prompt template: the intent analyzer.

Figure A3 presents the prompt template for the geospatial-relation retriever, which depends on the relation phrases identified by the intent analyzer and maps them to the GeoSPARQL/OGC operators with argument roles and optional numeric constraints.

Prompt Template for Concept Retriever

```
You are the **Concept Retriever** in a multi-agent NL→GeoSPARQL pipeline.
Your job is to extract the conceptual types from the decomposed intent.
Given the question, entities, decomposed intent, identify the conceptual types being queried.
Do NOT generate SPARQL. Do NOT invent facts.


INPUTS:
- A natural-language question.
- A list of resolved entities (mention=id). Treat them as hints.
- The decomposed intent from the Intent Decomposer.

OUTPUT:
RETURN A STRICT JSON OBJECT with EXACTLY this key
- "concepts": a list of zero or more json objects, each with:
    - "mention": the EXACT substring (span) from the question that denotes a conceptual type
    - "chosen": an object with:
        - "key": the normalized OSM key (e.g., "osmkey:amenity")
        - "val": the normalized OSM value (e.g., "restaurant")
    - "alternatives": a list of zero or more objects, each with:
        - "key": the normalized OSM key (e.g., "osmkey:amenity")
        - "val": the normalized OSM value (e.g., "cafe")

ADDITIONAL GUIDELINES:
- Use exact spans from the question; no normalization.
- If no concepts are found, return an empty array for "concepts".
- If there are no plausible alternatives, return an empty array for "alternatives".
- concepts are OSM key-value pairs
    - osmkey:place has values {(city, town, village, island, suburb, neighbourhood, ...)}
    - osmkey:amenity has values {(restaurant, cafe, school, hospital, prison, library, college, ...)}
    - osmkey:tourism has values {(museum, hotel, attraction, viewpoint, ...)}
    - osmkey:leisure has values {(park, stadium, playground, sports_centre, ...)}
    - osmkey:waterway has values {(river, stream, canal, ditch, ...)}
    - osmkey:building has values {(church, mosque, ...)}

FEW-SHOT EXAMPLES (do not quote or refer in output):
{few_shot_examples}

CURRENT INPUT:
Question:\n{question}
Entities:\n{entities_str}
Intent Decomposed Output:\n{intent_decomposed_output}
Output (strict JSON):
```

Figure A2. Prompt template: the concept retriever.

Prompt Template for Geospatial Relation Retriever

```
You are the **Geospatial Relation Retriever** in a multi-agent NL→GeoSPARQL pipeline.
Your job is to extract geospatial operations from the decomposed intent. Do NOT generate SPARQL or invent facts.

INPUTS:
- Natural-language question.
- Resolved entities (mention=id) — hints only.
- Intent JSON from the Intent Decomposer.

OUTPUT:
Return a STRICT JSON object with EXACTLY this key:
- "geospatial_relations": a list (possibly empty) of objects, each with:
    - "operator": one of {(distance, sfIntersects, sfWithin, sfContains, ...)} derived from the given inputs.
    - "arg_mentions": [span1, span2] — EXACT substrings from the question for the two arguments.
      If a concept and a reference entity exist, use [<concept span>, <reference entity span>].
      If two entities are referenced, keep their question order [ent1, ent2].
    - "constraint": {( "cmp": "<|<=|>|>=|=|null", "value": number or null, "uom": "uom:metre|null" )}.
      cmp and value represent a numeric threshold which is optional, and maybe used for distance operations.
      When a numeric threshold is explicitly stated, set cmp/value and convert km →metres; otherwise set cmp/value=null.
      For distance, set uom="uom:metre" even if cmp/value are null; for topological ops set uom=null.

ADDITIONAL GUIDELINES:
- Use exact spans from the question; no normalization.
- If no geospatial relation is found, return an empty array for "geospatial_relations".
- If multiple relations are found, return all in the array.
- Produce exactly two items in "arg_mentions" (binary operators only).
- Output formatting guidelines
    1. Compare
    • ≤: "within X km" (convert km → metres).
    • <: "less than X km (away)", "closer than X km".
    • < 5000 m: non-numeric proximity cues — "near", "nearby", "close to", "in the vicinity of", "(within a) short distance from".
    • > 5000 m: distance-far cues — "distant from", "a long distance away from", "far away from", "(located) far from".
    • **for Negation**, invert the default; e.g., "not near/not close to" ⇒ ≥ 5000 m.
    2. Touches
    • Map "borders", "adjacent to", and "next to" → sfTouches.
    3. Within vs. Contains
    • Map "in", "of", "within" → sfWithin (e.g., "park in Tuas", "castle of Thomson").
    • Use sfContains only when explicit: "contain(s)", "include(s)", "encompass(es)". Otherwise, default to sfWithin.
    4. Intersects
    • Use sfIntersects for phrase like "intersects with", "overlaps", "shares a space with".

FEW-SHOT EXAMPLES (do not quote or refer in output):
{few_shot_examples}

CURRENT INPUT:
Question:
{question}
Entities:
{entities_str}
Intent Decomposed Output:
{intent_decomposed_output}

Output (strict JSON):
```

Figure A3. Prompt template: the geospatial-relation retriever.

Figure A4 presents the prompt template for the property retriever, which extracts property mentions and grounds them to OSM property IRIs for projection or filtering.

Prompt Template for Property Retriever

```
You are the **Property Retriever** in a multi-agent NL→GeoSPARQL pipeline.
Your job is to extract non-spatial properties from the decomposed intent. Do NOT generate SPARQL or
invent facts.

INPUTS:
- A natural-language question.
- A list of resolved entities (mention=id). Treat them as hints.
- The decomposed intent from the Intent Decomposer.

OUTPUT:
Return a STRICT JSON object with EXACTLY this key:
- "properties": a list (possibly empty) of objects, each with:
    - "mention": the EXACT substring (span) from the question that denotes the attribute (e.g., "timezone",
"how many individuals live in").
    - "key": the canonical OSM property key as "osmkey:<key>" (e.g., "osmkey:timezone",
"osmkey:population").
    - "val": a SPARQL variable name starting with "?" (e.g., "?population", "?o", "?value").

ADDITIONAL GUIDELINES:
- Use exact spans only; no normalization.
- If no attribute property is requested (e.g., the question is purely geospatial), return {{"properties": []}}.
- Choose a concise variable name; if unclear, default to "?value".

FEW-SHOT EXAMPLES (do not quote or refer in output):
{few_shot_examples}

CURRENT INPUT:
Question:₩n{question}
Entities:₩n{entities_str}
Intent Decomposed Output:₩n{intent_decomposed_output}
Output (strict JSON):
```

Figure A4. Prompt template: the property retriever.

Figure A5 presents the prompt template for the operator builder, which integrates outputs from the multi-grained retrievers (concept, geospatial relation, property) with the intent analyzer, producing an IR aligned with SPARQL/GeoSPARQL evaluation.

Prompt Template for Operator Builder

```
You are the **Operator/DSL Builder** in a multi-agent NL→GeoSPARQL pipeline.
Output ONLY the DSL JSON (STRICT JSON). Do NOT generate SPARQL.

### DSL SHAPE (keys fixed, many fields optional)
{{
  "result_format": {{
    "form": "SELECT | ASK", (check if we really have the result format of SELECT or ASK in SPARQL?)
    "project": [ "?var", "(expr AS ?v2)" ],   // optional, omit for ASK
    "distinct": true | false                  // optional, omit for ASK
  }},
  "pattern": {{
    "basic": [
      // Prefer E/C/P shorthands; SPO form is also allowed as fallback.
      // use "as" to bind to a variable if needed later. if not needed, omit "as".
      // use "geom_as" to bind the geometry to a variable if needed.
      {{ "op": "E", "key": "osmkey:wikidata", "val": "QID", "as": "?resVar", "geom_as": "?rwktN" }},
      {{ "op": "C", "key": "osmkey:<key>",    "val": "<classVal>", "as": "?classVar", "geom_as": "?cwktN" }},
      {{ "op": "P", "key": "osmkey:<prop>",   "val": "?outVar|<lit|iri>" }},

      // Fallback (rare): raw SPO triples if needed
      {{ "s": "?s", "p": "prefix:pred", "o": "?o|W'litW"|<iri>" }}
    ],
    "advanced": [
      // Geospatial / topological op (no obligation to BIND)
      // use "as" to bind to a variable if needed later. if not needed, omit "as".
      {{ "op": "R", "relation": "geof:distance|geo:sfIntersects|geo:sfWithin|...", "args": ["?g1","?g2"], "uom": "uom:metre|null", "as":
"?var(optional)" }},

      // Optional helpers
      {{ "op": "BIND",   "expression": "<SPARQL_expr>", "as": "?var" }},
      {{ "op": "FILTER", "condition": "<SPARQL_boolean_expr>" }},

      // Control/structural constructs (use only if needed)
      {{ "op": "OPTIONAL", "pattern": {{ "basic": [...], "advanced": [...] }} }},
      {{ "op": "UNION",    "patterns": [ {{ "basic":[...], "advanced":[...] }}, {{ ... }} ] }},
      {{ "op": "VALUES",   "bindings": {{ "?var": ["<iri|lit|UNDEF>", ...] }} }},
      {{ "op": "GROUP_BY", "vars": ["?x", "..."] }},
      {{ "op": "HAVING",   "condition": "<boolean_agg_expr>" }},
      {{ "op": "SUBQUERY", "query": {{ "result_format": {{...}}, "pattern": {{...}}, "solution_modifiers": {{...}} }} }}
    ]
  }},
  "solution_modifiers": {{
    "order_by": [ {{ "expr": "?var|<expr>", "dir": "ASC|DESC" }} ],
    "limit": <int|null>,
    "offset": <int|null>,
    "reduced": true|false
  }}
}}

INPUTS:
- Natural-language question.
- Resolved entities (mention=id) — hints only.
- Intent JSON from the Intent Decomposer.
- Concept JSON from the Concept Retriever.
- Property JSON from the Property Retriever.
- Geospatial Relation JSON from the Geospatial Relation Retriever.
- Use ONLY the provided inputs; do NOT invent facts or relations.

OUTPUT:
Return a STRICT JSON object following the DSL SHAPE above.

FEW-SHOT EXAMPLES (do not quote or refer in output):
{few_shot_examples}

CURRENT INPUT:
Question:₩n{question}
Entities:₩n{entities_str}
Intent Decomposed Output:₩n{intent_decomposed_output}
Concept Output:₩n{concepts_output}
Property Output:₩n{properties_output}
Geospatial Relation Output:₩n{relations_output}
Output (strict JSON):
```

Figure A5. Prompt template: the operator builder.

Figure A6 presents the prompt template for the query generator, which depends on the operator builder, as well as the retrievers and intent analyzer, linearizing the IR into a complete GeoSPARQL query with prefixes, WHERE clauses, and solution modifiers.

Prompt Template for Query Generator

```
You are the **Query Synthesizer** in a multi-agent NL→GeoSPARQL pipeline.
Your job is to generate the final SPARQL query based on the outputs from previous agents.
Do NOT invent facts or relations.

INPUTS:
- A natural-language question.
- A list of resolved entities (mention=id) — hints only.
- The decomposed intent from the Intent Decomposer.
- The concept output from the Concept Retriever.
- The property output from the Property Retriever.
- The geospatial relation output from the Geospatial Relation Retriever.
- The operator output from the Operator Retriever.

OUTPUT:
Return the final SPARQL query as a string.

Additional Guidelines:
When generating queries, you may refer to the few-shot examples; they are illustrative only. You can view
them in two ways:
1. Few shot examples of **NL question + Entities** to **SPARQL query**
2. Few shot examples of **NL question + Entities** + **Multiple Agent Outputs** (Intent Decomposed
Output, Concept Output, Property Output, Geospatial Relation Output, Operator Output) to **SPARQL
query**

FEW-SHOT EXAMPLES (do not quote or refer in output):
{few_shot_examples}

CURRENT INPUT:
Question:₩n{question}
Entities:₩n{entities_str}
Intent Decomposed Output:₩n{intent_decomposed_output}
Concept Output:₩n{concept_output}
Property Output:₩n{property_output}
Geospatial Relation Output:₩n{geospatial_relation_output}
Operator Output:₩n{operator_output}
SPARQL Query:
```

Figure A6. Prompt template: the query generator.

References

1. Wei, J.; Tay, Y.; Bommasani, R.; Raffel, C.; Zoph, B.; Borgeaud, S.; Yogatama, D.; Bosma, M.; Zhou, D.; Metzler, D.; et al. Emergent Abilities of Large Language Models. *arXiv* **2022**, arXiv:2206.07682. [CrossRef]
2. Wei, J.; Wang, X.; Schuurmans, D.; Bosma, M.; Xia, F.; Chi, E.; Le, Q.; Zhou, D. Chain-of-thought prompting elicits reasoning in large language models. *Adv. Neural Inf. Process. Syst.* **2022**, *35*, 24824–24837.
3. Liu, P.; Biljecki, F. A review of spatially explicit GeoAI applications in urban geography. *Int. J. Appl. Earth Obs. Geoinf.* **2022**, *112*, 102936. [CrossRef]
4. Wang, S.; Huang, X.; Liu, P.; Zhang, M.; Biljecki, F.; Hu, T.; Fu, X.; Liu, L.; Liu, X.; Wang, R.; et al. Mapping the landscape and roadmap of geospatial artificial intelligence (GeoAI) in quantitative human geography: An extensive systematic review. *Int. J. Appl. Earth Obs. Geoinf.* **2024**, *128*, 103734. [CrossRef]
5. Mai, G.; Huang, W.; Sun, J.; Song, S.; Mishra, D.; Liu, N.; Gao, S.; Liu, T.; Cong, G.; Hu, Y.; et al. On the opportunities and challenges of foundation models for geospatial artificial intelligence. *arXiv* **2023**, arXiv:2304.06798. [CrossRef]
6. Tan, C.; Cao, Q.; Li, Y.; Zhang, J.; Yang, X.; Zhao, H.; Wu, Z.; Liu, Z.; Yang, H.; Wu, N.; et al. On the Promises and Challenges of Multimodal Foundation Models for Geographical, Environmental, Agricultural, and Urban Planning Applications. *arXiv* **2023**, arXiv:2312.17016. [CrossRef]
7. Manvi, R.; Khanna, S.; Mai, G.; Burke, M.; Lobell, D.B.; Ermon, S. GeoLLM: Extracting Geospatial Knowledge from Large Language Models. 2024. Available online: https://rohinmanvi.github.io/GeoLLM/ (accessed on 1 October 2025).
8. Wei, C.; Zhang, Y.; Zhao, X.; Zeng, Z.; Wang, Z.; Lin, J.; Guan, Q.; Yu, W. GeoTool-GPT: A trainable method for facilitating Large Language Models to master GIS tools. *Int. J. Geogr. Inf. Sci.* **2025**, *39*, 707–731. [CrossRef]
9. Zhang, Y.; Li, J.; Wang, Z.; He, Z.; Guan, Q.; Lin, J.; Yu, W. Geospatial large language model trained with a simulated environment for generating tool-use chains autonomously. *Int. J. Appl. Earth Obs. Geoinf.* **2025**, *136*, 104312. [CrossRef]
10. 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.
11. Wang, J.; Zhao, Z.; Wang, Z.J.; Da Cheng, B.; Nie, L.; Luo, W.; Yu, Z.Y.; Yuan, L.W. GeoRAG: A Question-Answering Approach from a Geographical Perspective. *arXiv* **2025**, arXiv:2504.01458.
12. Peng, Z.; Kuai, X.; Ke, S.; Dong, X.; Guo, R. Enhancing geodatabases operability: Advanced human-computer interaction through RAG and Multi-Agent Systems. *Big Earth Data* **2025**, *9*, 217–242. [CrossRef]
13. Feng, Y.; Zhang, P.; Xiao, G.; Ding, L.; Meng, L. Towards a Barrier-free GeoQA Portal: Natural Language Interaction with Geospatial Data Using Multi-Agent LLMs and Semantic Search. *arXiv* **2025**, arXiv:2503.14251. [CrossRef]

14. Punjani, D.; Singh, K.; Both, A.; Koubarakis, M.; Angelidis, I.; Bereta, K.; Angelidis, I.; Bereta, K.; Beris, T.; Bilidas, D.; et al. Template-based question answering over linked geospatial data. In Proceedings of the 12th Workshop on Geographic Information Retrieval, Seattle, WA, USA, 6 November 2018; pp. 1–10.
15. Hamzei, E.; Tomko, M.; Winter, S. Translating place-related questions to GeoSPARQL queries. In Proceedings of the ACM Web Conference 2022, Lyon, France, 25–29 April 2022; pp. 902–911.
16. Kefalidis, S.A.; Punjani, D.; Tsalapati, E.; Plas, K.; Pollali, M.A.; Maret, P.; Koubarakis, M. The question answering system GeoQA2 and a new benchmark for its evaluation. *Int. J. Appl. Earth Obs. Geoinf.* **2024**, *134*, 104203. [CrossRef]
17. Yang, J.; Jang, H.; Yu, K. Geographic Knowledge Base Question Answering over OpenStreetMap. *ISPRS Int. J. Geo-Inf.* **2023**, *13*, 10. [CrossRef]
18. Raffel, C.; Shazeer, N.; Roberts, A.; Lee, K.; Narang, S.; Matena, M.; Zhou, Y.; Li, W.; Liu, P.J. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.* **2020**, *21*, 1–67.
19. Li, B.Z.; Min, S.; Iyer, S.; Mehdad, Y.; Yih, W.T. Efficient One-Pass End-to-End Entity Linking for Questions. In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), Online, 16–20 November 2020; pp. 6433–6441.
20. Goodfellow, I.; Bengio, Y.; Courville, A.; Bengio, Y. *Deep Learning (Vol. 1, No. 2)*; MIT Press: Cambridge, MA, USA, 2016.
21. Talebirad, Y.; Nadiri, A. Multi-agent collaboration: Harnessing the power of intelligent LLM agents. *arXiv* **2023**, arXiv:2306.03314. [CrossRef]
22. Wang, Y.; Wu, Z.; Yao, J.; Su, J. Tdag: A multi-agent framework based on dynamic task decomposition and agent generation. *Neural Netw.* **2025**, *185*, 107200. [CrossRef]
23. 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. [CrossRef]
24. Achiam, J.; Adler, S.; Agarwal, S.; Ahmad, L.; Akkaya, I.; Aleman, F.L.; Almeida, D.; Altenschmidt, J.; Altman, S.; Anadkat, S.; et al. Gpt-4 technical report. *arXiv* **2023**, arXiv:2303.08774. [CrossRef]
25. Singh, K.; Radhakrishna, A.S.; Both, A.; Shekarpour, S.; Lytra, I.; Usbeck, R.; Vyas, A.; Khikmatullaev, A.; Punjani, D.; Lange, C.; et al. Why reinvent the wheel: Let's build question answering systems together. In Proceedings of the 2018 World Wide Web Conference, Lyon, France, 23–27 April 2018; pp. 1247–1256.
26. Finkel, J.R.; Grenager, T.; Manning, C.D. Incorporating non-local information into information extraction systems by gibbs sampling. In Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05), Ann Arbor, MI, USA, 25–30 June 2005; pp. 363–370.
27. Usbeck, R.; Ngonga Ngomo, A.C.; Röder, M.; Gerber, D.; Athaide Coelho, S.; Auer, S.; Both, A. AGDISTIS–agnostic disambiguation of named entities using linked open data. In *ECAI 2014*; IOS Press: Amsterdam, The Netherlands, 2014; pp. 1113–1114.
28. Lample, G.; Ballesteros, M.; Subramanian, S.; Kawakami, K.; Dyer, C. Neural Architectures for Named Entity Recognition. In Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, San Diego, CA, USA, 12–17 June 2016; Association for Computational Linguistics: East Stroudsburg, PA, USA, 2016.
29. Devlin, J.; Chang, M.W.; Lee, K.; Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Minneapolis, MN, USA, 2–7 June 2019; Volume 1, pp. 4171–4186.
30. Ferragina, P.; Scaiella, U. Tagme: On-the-fly annotation of short text fragments (by wikipedia entities). In Proceedings of the 19th ACM International Conference on Information and Knowledge Management, Toronto, ON, Canada, 26–30 October 2010; pp. 1625–1628.
31. Brown, P.F.; Della Pietra, V.J.; Desouza, P.V.; Lai, J.C.; Mercer, R.L. Class-based n-gram models of natural language. *Comput. Linguist.* **1992**, *18*, 467–480.
32. Elman, J.L. Finding structure in time. *Cogn. Sci.* **1990**, *14*, 179–211. [CrossRef]
33. Bengio, Y.; Simard, P.; Frasconi, P. Learning long-term dependencies with gradient descent is difficult. *IEEE Trans. Neural Netw.* **1994**, *5*, 157–166. [CrossRef]
34. Pascanu, R.; Mikolov, T.; Bengio, Y. On the difficulty of training recurrent neural networks. In Proceedings of the International Conference on Machine Learning, Atlanta, GA, USA, 16–21 June 2013; pp. 1310–1318.
35. Hochreiter, S.; Schmidhuber, J. Long short-term memory. *Neural Comput.* **1997**, *9*, 1735–1780. [CrossRef]
36. Bahdanau, D.; Cho, K.; Bengio, Y. Neural machine translation by jointly learning to align and translate. In Proceedings of the International Conference on Learning Representations (ICLR), San Diego, CA, USA, 7–9 May 2015.
37. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, L.; Polosukhin, I. Attention is all you need. In Proceedings of the Advances in Neural Information Processing Systems, Long Beach, CA, USA, 4–9 December 2017; Volume 30.
38. Radford, A.; Narasimhan, K.; Salimans, T.; Sutskever, I. *Improving Language Understanding by Generative Pre-Training*; OpenAI: San Francisco, CA, USA, 2018; Available online: https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf (accessed on 1 October 2025).

39. Radford, A.; Wu, J.; Child, R.; Luan, D.; Amodei, D.; Sutskever, I. Language models are unsupervised multitask learners. *OpenAI Blog* **2019**, *1*, 9.
40. 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.
41. Kaplan, J.; McCandlish, S.; Henighan, T.; Brown, T.B.; Chess, B.; Child, R.; Gray, S.; Radford, A.; Wu, J.; Amodei, D. Scaling laws for neural language models. *arXiv* **2020**, arXiv:2001.08361. [CrossRef]
42. Wei, J.; Bosma, M.; Zhao, V.; Guu, K.; Yu, A.W.; Lester, B.; Du, N.; Dai, A.M.; Le, Q.V. Finetuned Language Models are Zero-Shot Learners. In Proceedings of the International Conference on Learning Representations, Online, 25–29 April 2022.
43. Ouyang, L.; Wu, J.; Jiang, X.; Almeida, D.; Wainwright, C.; Mishkin, P.; Zhang, C.; Agarwal, S.; Slama, K.; Ray, A.; et al. Training language models to follow instructions with human feedback. *Adv. Neural Inf. Process. Syst.* **2022**, *35*, 27730–27744.
44. Ye, X.; Yavuz, S.; Hashimoto, K.; Zhou, Y.; Xiong, C. RNG-KBQA: Generation Augmented Iterative Ranking for Knowledge Base Question Answering. In Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Dublin, Ireland, 22–27 May 2022; pp. 6032–6043.
45. Shu, Y.; Yu, Z.; Li, Y.; Karlsson, B.; Ma, T.; Qu, Y.; Lin, C.Y. TIARA: Multi-grained Retrieval for Robust Question Answering over Large Knowledge Base. In Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, Abu Dhabi, United Arab Emirates, 7–11 December 2022; pp. 8108–8121.
46. Wu, L.; Petroni, F.; Josifoski, M.; Riedel, S.; Zettlemoyer, L. Scalable Zero-shot Entity Linking with Dense Entity Retrieval. In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), Online, 16–20 November 2020; pp. 6397–6407.
47. Harris, S.; Seaborne, A.; Prud'hommeaux, E. SPARQL 1.1 Query Language. W3C Recommendation. 2013. Available online: https://www.w3.org/TR/sparql11-query/ (accessed on 1 October 2025).
48. Perry, M.; Herring, J.; Musen, P. GeoSPARQL 1.1: A Geographic Query Language for RDF Data. OGC Implementation Standard. Open Geospatial Consortium. 2022. Available online: https://www.ogc.org/standards/geosparql (accessed on 1 October 2025).

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.