DOI: https://doi.org/10.36910/6775-2524-0560-2026-62-11
UDC 004:02
Moroz Borys, D.Sc., Professor
https://orcid.org/0000-0002-5625-0864
Shyshatskyi Oleksandr, PhD student
https://orcid.org/0009-0008-6008-7079
National Technical University «Dnipro Polytechnic», Dnipro, Ukraine

NOVEL ARCHITECTURAL APPROACH TO AI LLM AGENTS FOR GEOSPATIAL DATABASES

Shyshatskyi O., Moroz B. A novel approach to AI LLM Agents for interaction with Geospatial Databases. The integration of Large Language Models (LLMs) with geospatial databases is complicated by several practical limitations. First, LLMs operate on textual data and are not designed to perform precise computations over points, lines, and polygons. Second, the model's context window is too small to accommodate query results that may contain tens of thousands of records or more. This paper proposes an architecture that addresses these challenges by separating responsibilities between system components. The LLM is responsible for interpreting the user's query and planning the sequence of actions required to solve the task, while a specialized backend executes the geospatial operations. To store intermediate results, an external memory mechanism is employed: the model sends geospatial queries to the backend and receives only short descriptors (handles) in response. These descriptors serve as references to query results stored in external memory and can be reused in subsequent operations. This approach allows the LLM to construct complex chains of geospatial operations without exceeding its context limitations.
Keywords: large language models, geospatial databases, external memory, function calling, spatial queries, LLM agents.

Шишацький О.О., Мороз Б.І. Новий архітектурний підхід до AI LLM-агентів для взаємодії з геопросторовими базами даних. Інтеграція великих мовних моделей (Large Language Models, LLM) з геопросторовими базами даних ускладнена низкою фундаментальних обмежень. По-перше, LLM орієнтовані на обробку тексту і не забезпечують коректного виконання точних геометричних обчислень над просторовими об'єктами (точками, лініями, полігонами). По-друге, обмежений розмір контекстного вікна робить неможливим безпосереднє опрацювання результатів просторових запитів, які можуть містити десятки тисяч і більше об'єктів. У статті запропоновано архітектуру, яка вирішує цю проблемі шляхом розділення ролей: LLM відповідає за інтерпретацію користувацького запиту та планування дій для вирішення проблеми, а спеціалізований бекенд - за виконання геопросторових операцій. Для збереження проміжних результатів використано "зовнішню пам'ять": модель надсилає до бекенду геопросторові запити, але у відповідь отримує лише короткі "дескриптори" (handles). Ці дескриптори є посиланнями на результати запиту у "зовнішній пам'яті", що можуть бути перевикористані в подальших запитах. Таким чином LLM може будувати ланцюжки операцій без переповнення контексту.
Ключові слова: великі мовні моделі, геопросторові бази даних, зовнішня пам'ять, виклик функцій, просторові запити, LLM-агенти.

Introduction. Large Language Models (LLMs) effectively interpret user queries in natural language, can break down tasks into steps, and form structured responses [1]. This makes LLMs candidates for the role of "interface" to complex subject domains, particularly database systems, where specialized operator training is usually required. At the same time, geospatial data is becoming increasingly abundant and is used in a wide variety of fields. Such datasets can contain millions of objects with geometry and attributes. Working with them usually requires GIS tools and skills, which creates a barrier for users.

At the intersection of these trends, a practical question arises: how to build a natural language interface to geospatial databases so that the LLM assists the user but does not perform those operations for which it is poorly suited (precise geometric computations, processing large samples in context, preserving intermediate state between steps)?

Problem Statement. The integration of LLMs with geospatial databases has several key limitations that complicate geospatial data processing and may introduce risks in certain use cases:

1. **Modality mismatch**: LLMs are designed for text, while geospatial data is geometry (points, lines, polygons), coordinate systems, and spatial relationships. Technically, one can pass coordinates to the model and ask it to calculate area or intersection, but this yields unreliable results.
2. **Context window limitations**: Modern LLMs have large but still limited context windows, meaning the amount of information they can process at once. Depending on the model, this limit can be up to 200,000 tokens [15]. Geospatial datasets often contain tens or hundreds of thousands of objects, each with attributes and geometry. Processing such data entirely within the context window is impractical: a GeoJSON file with 100,000 objects can easily expand to millions of tokens.

Hence the problem: an architecture is needed that combines the strengths of LLMs in understanding user prompts with reliable execution of spatial operations.

Analysis of Recent Research and Publications. A significant number of existing studies are devoted to converting natural language queries into formal database queries (so-called Text-to-SQL). Early results demonstrated the practical feasibility of such a solution [14]. Subsequently, standard test task sets and benchmarks Spider [13] and BIRD [5] were created for them. It was also determined that query synthesis quality can be improved through model training and the use of self-correction mechanisms [8]. At the same time, for practical applications, the question of safe use of such a solution remains open. The most vulnerable are architectures in which queries or program code generated by LLMs are directly executed. For research prototypes, this may be acceptable, but in production systems, such practice creates predictable risks: injections, unauthorized data access, and resource-intensive tasks leading to denial of service (Figure 1).

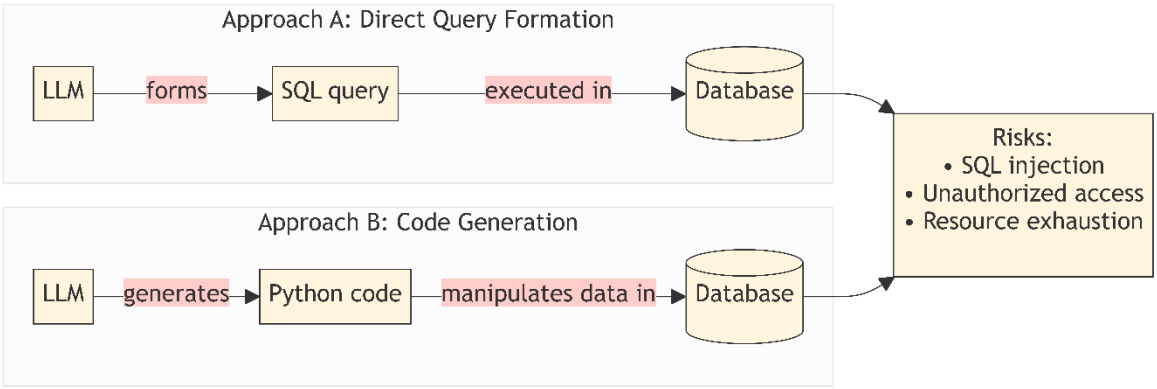

Fig. 1 – Existing approaches to LLM interaction with databases.
Both direct SQL generation and code generation allow LLM unrestricted access to underlying systems, creating security vulnerabilities.

Databases are not the only domain of AI application that does not allow direct data processing, so there are known solutions. One of them is the function calling mechanism [2][6]. The LLM receives additional meta-information in the prompt in the form of a list of available functions and their parameters. Instead of a response, the LLM can send a request to execute such a function to the control program, and the control program executes it and returns the result. The mechanism in which the model performs planning and interacts with an API has been formalized in the Toolformer [9] and ReAct [12] frameworks, on the basis of which agent frameworks (LangChain [3], AutoGPT [11]) operate. At the same time, the specific API provided for function calling is limited only by the imagination and needs of the program author, potentially allowing integration of AI with an unlimited number of domains.

To overcome context window limitations, Retrieval-Augmented Generation (RAG) is widely used, i.e., an approach in which the model receives additional relevant fragments from an index [4]. Individual works propose more controlled memory mechanisms: MemGPT [7] formalizes methods for transferring information between "short" and "long" memory, and Reflexion [10] demonstrates improvement in results from generalization and analysis of previous attempts. Unfortunately, most of the approaches focus on textual artifacts or dialogue history. In geospatial tasks, a different problem is key: efficient storage and reuse of intermediate results of structured operations (often of large volume) between task execution steps.

Despite significant progress in query synthesis and dialogue memory mechanisms, in the task of using LLMs as a secure interface to large geospatial data, there are no agreed-upon architectural solutions that simultaneously account for context window limitations, computational correctness requirements, and data access policies. Therefore, the **goal of this work** is to develop an architecture for integrating LLMs with geospatial databases that ensures correct execution of complex multi-step spatial queries by separating the functions of interpretation, planning, and execution of geographic computations.

Presentation of the main material. During query execution, the LLM operates within a closed computational environment and operates exclusively on input text sequences and parameters formed during pre-training. All computations occur on the input token sequence, which has a strict upper limit – the context

window. This limit includes not only the user's prompt but also system instructions, dialogue history, descriptions of available tools, and (if necessary) results that we return to the model through these tools. In practice, "token consumption" consists of two parts:

1. Input tokens: everything the model reads (prompt, history, tool schemas, call results).
2. Output tokens: what the model generates (explanations, plan, function calls, or final answer).

Let us consider the limitations that arise for the developer using the example of a typical geospatial query: "*given GeoJSON, it is necessary to select objects with an area greater than 1 hectare*". Making such a filter directly "inside" the LLM is unacceptable for two reasons: context window limitations and lack of guarantees of computational correctness/reproducibility:

1. GeoJSON with thousands of polygons can correspond to hundreds of thousands or millions of tokens. As a result, the input data may not fit in the model's context window and will be "truncated," making the filtering results incomplete.
2. Filter by area requires correct mathematical processing of data, which text models are not capable of.

The LLM can formulate instructions and an execution plan, but it is not designed to perform precise geometric computations over large data arrays and does not provide guarantees of correctness for such results. Thus, in an applied architecture, it is advisable to clearly separate the responsibilities of components: the LLM is used for interpreting queries in natural language and planning the sequence of operations, while the execution of computationally complex and accuracy-critical geospatial operations is delegated to a specialized backend.

The function calling mechanism [12] allows formalizing the role separation described above: the LLM interprets the query and plans the sequence of actions, while the execution of computationally complex or accuracy-critical operations is delegated to an external service. Each operation is defined by a contract (name, purpose, parameters, result format), and the tool's response is returned in a structured form. An example of such a contract is shown in Figure 2.

```
load_layer:
  input:  source (string)
  output: GeoJSON

filter_by_area:
  input:  input (GeoJSON),
          threshold_ha (number)
  output: GeoJSON
```

Fig. 2 – Example of a contract for two external functions:
load_layer for loading geodata from a source and filter_by_area for filtering results by area.

By executing contractual external functions, AI can perform sequential actions on data, forming a controlled multi-step pipeline, as shown in Figure 3 using the example of a user prompt "find polygons with an area greater than 1 hectare.".

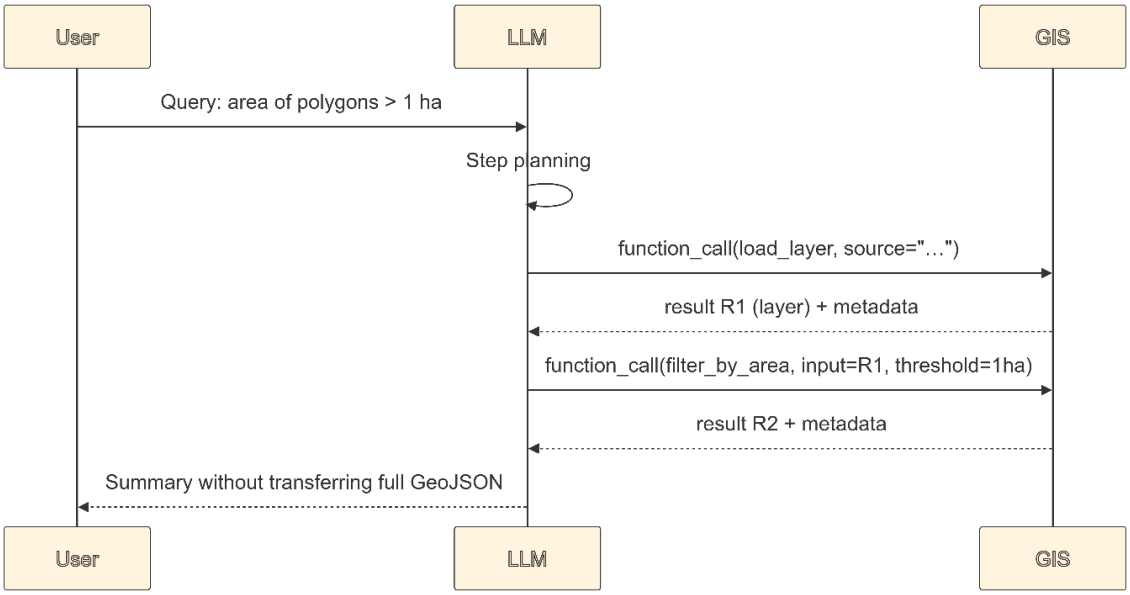

Fig. 3 – Pipeline «plan → tool call → structured result»

However, this pipeline implementation has an obvious problem: external functions return the result of each contractual operation (e.g., filtering) directly to the LLM in textual form. This approach is only viable for small data volumes. If results have significant size (thousands or tens of thousands of objects), their transmission as full GeoJSON or a table to the LLM quickly exhausts the context window. Therefore, in our proposed architecture, we also separate "where data is stored" and "what the LLM operates on." External functions return to the LLM not the full result in textual form, but only a descriptor (handle) - a compact identifier that uniquely references the corresponding result stored in external memory. The model builds a multi-step chain, passing descriptors to subsequent calls without transferring the entire set of geometries into text. The updated contract for function calling is shown in Figure 4, and Figure 5 shows the call diagram for the same query as in the previous example.

```
load_layer:
  input:
    source: string
  output:
    Handle(type = layer, id = <unique_hash>)

filter_by_area:
  input:
    handle_id: Handle
    threshold_ha: number
  output:
    Handle(type = attribute_filter, id = <unique_hash>)

describe_handle:
  input:
    handle_id: Handle
  output:
    string
```

Fig. 4 – Example of an improved contract where only short references to data are returned instead of the data itself for their subsequent use.

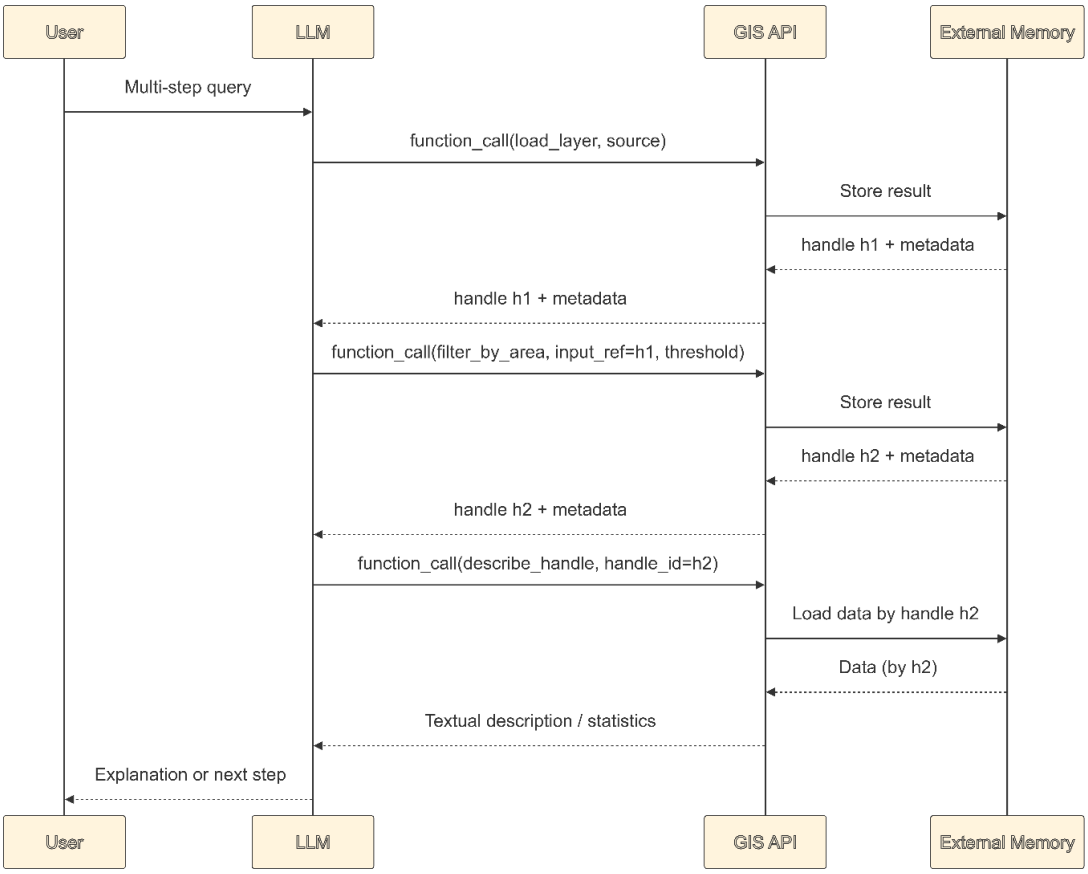

Fig. 5 – Use of external memory: large intermediate results are stored outside the context, and the LLM operates with short descriptors (handles).

To demonstrate the effectiveness of the architecture, a demonstration program was created, the GIS API component of which was implemented based on the geopandas library, the LLM model – GPT 4.1, and basic external function calls were also implemented as shown in Table 1.

Table 1.

Category	Function	Purpose / Result
Database Inspection	list_layers	Returns a list of available layers.
Database Inspection	describe_layer	Returns the schema (fields and types), service metadata, and usage notes (e.g., how to search for boundaries and which fields to apply in filters).
Loading	load_layer	Load a layer (with preliminary filters if needed), output data: descriptor to a set of objects
Filter	filter_attributes	Filter a set by conditions (AND/OR), output data: descriptor to a filtered set.
Filter	spatial_filter	Filter by spatial relationships (within/intersects/contains, etc.), output data: descriptor to result
Sorting	sort_attributes	Order by fields, output data: descriptor to a sorted set.

Analytics	get_unique_values, get_value_counts	Get aggregated statistics by field, output data: compact list/table of values.
Inspection	preview_layer	Get a small sample of objects for verification, output data: compact output.
Export	export_geojson	Export result to GeoJSON, output data: file/string (according to access policies).

The demonstration program is available in the form of the open repository [16], and below are execution logs for several practical tasks, obtained by running the program with corresponding task specifications. To demonstrate the operation of the proposed architecture, step-by-step interaction of system components is shown in the form of a chain "planning, tool call, descriptor return." This interaction demonstrates how the system works with large intermediate results stored outside the LLM context. As a data source, an OpenStreetMap dataset located in the data/osm.geojson directory was used. The volume comprises 177,217 objects: buildings, POIs, administrative boundaries, etc.

Experimental task #1. The first task assigned to the program is to find schools within the Dnipro city community, leaving only gymnasiums and lyceums.

Task Formulation:
"Find all schools within the Dnipro city council. Leave only gymnasiums and lyceums. Provide their names and coordinates.
The steps of external function calls as they performed by the LLM agent are shown in Table 2.

Table 2.

№	Function call	Reason for Call	Data (in → out)
1	`list_layers()`	Determining available layers.	–
2	`describe_layer("osm")`	Identifying available attributes.	–
3	`load_layer("osm", [("eq","amenity","school")]) → layer_schools`	Forming a set of candidate objects.	177 217 → 27
4	`load_layer("osm", [("eq","boundary","administrative"),("eq","admin_level","7"),("like","name","%Дніпровська міська громада%")]) → layer_boundary`	Obtaining the boundary polygon.	177 217 → 1
5	`spatial_filter(layer_schools,"within",layer_boundary)`	Removing objects outside boundaries.	27 → 25
6	`filter_attributes(layer_filtered,[("like","name","%гімназія%"),("like","name","%ліцей%")],"or")`	Filtering by name.	25 → 4
7	`preview_layer(layer_result)`	Returning result.	4 → 4

Results.
Found 4 educational institutions within the community whose names contain "gymnasium" or "lyceum" (original language preserved):

- Ліцей-інтернат фізико-математичного профілю — (48.4373025, 35.0337536)
- Ліцей №100 — (48.4648004, 35.0560842)
- Гімназія №1 — (48.46528, 34.9836802)
- Гімназія №83 — (48.410033, 35.048007)

Experimental task #2. The second task assigned to the program is to find schools with unknown addresses in the dataset.

Task Formulation:

"Find all schools where address information is missing. Provide their names and coordinates."

The steps of external function calls by the LLM agent are shown in Table 3.

Table 3.

№	Function call	Reason for Call	Data (in → out)
1	`list_layers()`	Determining available layers.	–
2	`describe_layer("osm")`	Identifying available attributes.	–
3	`load_layer("osm", [("eq","amenity","school")]) → layer#530fdfb1`	Forming a set of candidate objects.	177 217 → 27
4	`filter_attributes(layer#530fdfb1, [("is_null","addr_street"), ("eq","addr_street","")], logical_operator="or") → attribute_filter#8f935550`	Identifying schools without addresses.	27 → 13
5	`preview_layer(attribute_filter#8f935550, limit=50)`	Returning result.	13 → 13

Result: found 13 schools with unknown addresses (original language preserved):

- Школа №32 — (48.4677806, 35.0089433)
- Спеціалізована дитячо-юнацька спортивна школа олімпійського резерву №1 — (48.4630062, 35.0330474)
- Friend's club english school — (48.4521571, 35.0627121)
- Школа №75 — (48.4390938, 35.0126021)
- №25 — (48.3981063, 34.9802923)
- Школа № 51 — (48.4691747, 34.9991566)
- Початкова школа ліцею №100 — (48.4636336, 35.0561209)
- Гімназія №1 — (48.46528, 34.9836802)
- Гімназія №83 — (48.410033, 35.048007)
- Школа № 58 — (48.474792, 35.0220994)
- Дитяча художня школа №1 — (48.4620809, 35.0282672)
- Школа №22 — (48.4414913, 35.0130805)
- Школа без назви — (48.4673215, 35.025446)

Practical examples show that the demonstration program correctly forms a sequence of atomic geospatial operations without user intervention and can perform everyday tasks on large datasets.

Potential Directions for Further Research. The current implementation has several limitations that open opportunities for future research:

1. Vendor dependency: the system relies on commercial LLMs (OpenAI in our case). Adaptation to open-source models (Llama, Mistral) will require potentially different strategies.
2. Write operations: the current implementation works only for reading. Support for write operations (creation, modification, deletion of objects) will require additional mechanisms for object creation and transaction management.

Conclusions. This work proposes a new architectural approach to integrating large language models with geospatial databases, which overcomes fundamental limitations of LLMs in working with spatial data - particularly context window limitations and the guarantees for correctness of geometric computations. The main idea of the proposed approach is clear separation of responsibilities between system components: the LLM is used exclusively for interpreting queries in natural language and planning

the sequence of actions, while the execution of geospatial operations is delegated to a specialized computational backend.

The scientific novelty of the work lies in the use of an external memory mechanism based on descriptors for storing and reusing intermediate results of spatial queries. This approach allows building multi-step analytical pipelines without transferring large volumes of structured data into the LLM context. The presence of an intermediate layer of external functions also allows building data access control policies, which is an important aspect for multi-user systems.

The demonstrational implementation of the proposed architecture and experiments on the OpenStreetMap dataset confirm its effectiveness for typical geoanalytical tasks, particularly attribute and spatial filtering, aggregation, and data inspection. It is shown that users can formulate complex queries in natural language without knowledge of GIS tools, while the system ensures execution of data access operations.

The proposed approach can be used as a basis for building secure LLM-oriented interfaces to large geospatial repositories in public and corporate information systems. Further research should be directed toward scaling the external memory mechanism, supporting write operations, using open language models, and diverse sources of geospatial data.

References

1. Evaluation of the efficiency of large language models for extracting entities from unstructured documents / O. Shyshatskyi et al. Technology audit and production reserves. 2025. Vol. 6, no. 2(86). P. 57–67. URL: https://doi.org/10.15587/2706-5448.2025.341926 (date of access: 20.01.2026).
2. Tool use with Claude. Claude API Docs. URL: https://docs.anthropic.com/claude/docs/tool-use (date of access: 06.01.2026).
3. langchain: the platform for reliable agents. URL: https://github.com/langchain-ai/langchain (date of access: 23.12.2025).
4. P. Lewis, E. Perez, A. Piktus., Retrieval-augmented generation for knowledge-intensive NLP tasks. Advances in neural information processing systems. 33rd ed. 2020. P. 9459–9474.
5. J. Li, B. Hui, Ge Qu. Can LLM already serve as a database interface? A big bench for large-scale database grounded text-to-SQLs. NIPS '23: proceedings of the 37th international conference on neural information processing systems, New Orleans LA, 10 December 2023. 2023. P. 42330–42357.
6. Function calling and other API updates. OpenAI. URL: https://openai.com/blog/function-calling-and-other-api-updates (date of access: 23.12.2025).
7. C. Packer, S. Wooders, K. Lin, "MemGPT: Towards LLMs as operating systems", 2023. (Preprint 2310.08560). DOI: https://doi.org/10.48550/arXiv.2310.08560 (date of access: 20.02.2026).
8. M. Pourreza and D. Rafiei, DIN-SQL: Decomposed In-Context Learning of Text-to-SQL with Self-Correction, 2023. (Preprint. 2304.11015). DOI: https://doi.org/10.48550/arXiv.2304.11015 (date of access: 20.02.2026).
9. Schick T., Dwivedi-Yu J., Dessì R. Toolformer: Language models can teach themselves to use tools. 2023. (Preprint. 2302.04761). URL: https://doi.org/10.48550/arXiv.2302.04761 (date of access: 20.02.2026).
10. Shinn N., Berman E., Cassano F. Reflexion: Language Agents with Verbal Reinforcement Learning. 2023. 19 p. (Preprint. 2303.11366). DOI: https://doi.org/10.48550/arXiv.2303.11366 (date of access: 20.02.2026).
11. GitHub - Significant-Gravitas/AutoGPT: AutoGPT is the vision of accessible AI for everyone, to use and to build on. Our mission is to provide the tools, so that you can focus on what matters. GitHub. URL: https://github.com/Significant-Gravitas/AutoGPT (date of access: 13.12.2025).
12. S. Yao, J. Zhao, D. Yu., ReAct: Synergizing Reasoning and Acting in Language Models. 2023. 33 p. (Preprint. 2210.03629). DOI: https://doi.org/10.48550/arXiv.2210.03629 (date of access: 20.02.2026).
13. T. Yu, R. Zhang, K. Yang, Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, 1 October 2018. 2018. P. 3911–3921. DOI: https://doi.org/10.18653/v1/d18-1425 (date of access: 20.02.2026).
14. Zhong V., Xiong C., Socher R. Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning. 2017. 12 p. (Preprint. 1709.00103). DOI: https://doi.org/10.48550/arXiv.1709.00103 (date of access: 20.02.2026).
15. Google. Our next-generation model: gemini 1.5. Innovation & AI. URL: https://blog.google/innovation-and-ai/products/google-gemini-next-generation-model-february-2024/#gemini-15 (date of access: 11.11.2025).
GEO LLM demonstration, URL: https://github.com/oshyshatskyi-phd/geo-llm-demo (date of access: 26.02.2026).

Історія статті:
Отримано: 26.01.2026 Доопрацьовано: 21.02.2026 Прийнято до друку: 23.03.2026 Опубліковано: 29.03.2026