Computers, Materials & Continua

Tech Science Press

https://doi.org/10.32604/cmc.2026.080249

ARTICLE

Evaluating Ontology-Based Function Definitions for MCP Invocation Accuracy in LLM Agent-Based HPC Systems

Yejin Kwon[1], **Jeongcheol Lee**[1] **and Youngbom Park**[2,*]

[1]Analysis Platform Team, Department of Supercomputing Acceleration Research, Division of National Supercomputing, Korea Institute of Science and Technology Information, Daejeon, Republic of Korea

[2]Artificial Intelligence Information Architecture Lab, Department of Software Engineering, Dankook University, Yongin-si, Republic of Korea

*Corresponding Author: Youngbom Park. Email: ybpark@dankook.ac.kr

Received: 10 February 2026; Accepted: 17 April 2026; Published: 15 June 2026

ABSTRACT: The web-based High-Performance Computing (HPC) platform provides a simulation environment that enables users to perform computational science and engineering tasks through web services, thereby eliminating the need for complex terminal-based environments. Notwithstanding the aforementioned advantages, extant platforms frequently necessitate a considerable degree of user expertise, whilst the intricacy of simulation configuration and execution engenders limitations in terms of accessibility and usability. Furthermore, while Retrieval-Augmented Generation (RAG)-based systems are effective for information retrieval, they are insufficient for accurately constructing and invoking executable service tools. In order to address these limitations, this study proposes a user agent system integrated within a web-based HPC simulation environment, said system being based on an LLM. The proposed system enhances user understanding of available applications and execution workflows, and supports precise configuration and execution of simulations. In order to facilitate practical service tool invocation, the system integrates Model Context Protocol (MCP)-based service tools and introduces an ontology-driven approach for object normalization and relational definition. The system leverages the structured relationships among service tools, transforming LLM outputs into actionable and accurate inputs for service execution. The experimental results demonstrate that the proposed approach significantly improves the accuracy of MCP-based service tool invocation and the appropriateness of responses when compared to conventional RAG-based methods. The proposed system enhances the accessibility and usability of HPC platforms and provides a practical framework for LLM-driven service automation.

KEYWORDS: HPC platform; ontology; RAG (retrieval-augmented generation); LLM (large language model); model context protocol (MCP)

1 Introduction

The web-based High Performance Computing (HPC) platform [1], hereafter referred to as Edison-1, provides a suite of services that facilitate the execution of simulations across a range of computational science and engineering domains via the internet. This platform also enables the sharing of resulting data. Each user is able to register computational simulation applications across a range of fields and execute computations by allocating HPC resources to run the simulations [2,3]. While this platform provides a user-friendly web-based simulation environment, it is acknowledged that new users may encounter difficulties in comprehending the system in its entirety upon initial access. Moreover, given the character of computational

science and engineering, the execution of applications in fields other than one's primary specialization necessitates a breadth of specialized knowledge and an appreciation of the execution environment. In addition, a rigorous analysis and verification of input and output values is required, with particular reference to the professional domain. Consequently, to enhance user accessibility, usability, and platform comprehension, a new component was designed and built. This incorporates a user service execution component based on the widely used Large Language Model (LLM) to increase new user adoption and execution rates.

In addition to the fundamental platform functions, it is imperative to analyze user-specific trends in applications available on the user-executable simulation system. Furthermore, it is essential to identify computational science simulation information that is usable on the platform, and to provide users with an interactive simulation platform. The selection of appropriate options should be made based on the diverse data and simulation calculation application information currently provided by the platform. The utilisation of artificial intelligence (AI) algorithms, meticulously tailored to the data, enables platform-based data to recommend appropriate simulation data [4], consequently, the platform-based data must provide LLM results to enable appropriate data and application selection, task execution, and result analysis based on user prompts. Consequently, this study established a Retrieval-Augmented Generation (RAG) environment based on diverse simulation input/output data, registered datasets, and simulation applications. The construction of the system was undertaken with the objective of yielding appropriate results in accordance with the input prompts, incorporating each data type.

Existing services that utilize LLMs are susceptible to confirmation bias and inherent biases. The propagation of the problem is continuous once a misguided judgment has been made [5]. Moreover, if the system is found to be systematically misinterpreting user queries, this will inevitably lead to the perpetuation of errors, as it will generate responses to novel inquiries based on the input/output of prior user queries [6]. Consequently, users may be more prone to execute services based on erroneous information and make erroneous judgments [7]. In the event of the development of service features being confined to the handling of general queries, the likelihood of the entire service being misused due to an LLM's erroneous response would be minimal. This is due to the fact that users exclusively utilize queries based on manuals for the LLM on the platform. Consequently, even in the event of an incorrect response, no significant problems are experienced by the platform itself. However, if the service is required to execute platform functions and reflect the execution results on the platform, a service environment is established in which a single error can propagate exponentially and contaminate the entire platform [8]. Consequently, procedural stability is imperative, facilitating the utilization of platform functions with minimal errors through accurate judgment and procedures for each user query. In order to enhance this stability, the present paper has designed precise service integration by combining results based on user queries with an ontology-enhanced LLM system model. The ontology-enhanced LLM system was utilized to define the services that were to be invoked on the actual platform, in addition to the definitions of linked services. These definitions were then applied to the model.

The present paper sets out the development of an LLM agent system that is capable of handling a variety of user queries. This objective is realized through the integration of data generated by simulation functions provided by an existing high-performance computing (HPC)-based computational science and engineering simulation platform into an RAG structure via an LLM agent. This enhancement is designed to facilitate enhanced user comprehension and accessibility to the diverse range of services utilized within the simulation system. Furthermore, we periodically converted the diverse data continuously generated by the currently operational HPC platform into a vector database in order to build a RAG system incorporating this data. The RAG system has been developed to facilitate the real-time viewing and processing of diverse data and user simulation information by users. Finally, to establish a connection between the process handling diverse

user requests for executing and processing actual simulation services within the real-user query processing segment, an MCP server was defined. The establishment of an agent-based execution simulation service, linked to the existing platform service server, was undertaken. In order to define executable services on the MCP server and link them, a structured result processing process is required, rather than an LLM agent service with general user query routines. In order to process queries in a manner consistent with existing simulation execution services, the presence of structured result data for standardized user queries is essential. The processing of ontology-based simulation is imperative in order to define the structured result data, as well as the existing platform's structure and linkage process. The present paper sets out to define an ontology-based platform and standardize result data as one processing step, with a view to structuring result data for user queries.

The structure of this paper is as follows.

- Section 2 provides a detailed exposition of the structural characteristics of extant HPC platforms, LLM agent systems, and the MCP system for service execution.
- Section 3 of this text provides a comprehensive overview of the design and implementation of the process for building and executing services. This is achieved by integrating LLM agents into the established HPC platform.
- Section 4 conducts quantitative and qualitative evaluations of results for actual user prompt queries and assesses data suitability for executing real simulation services.
- Finally, Section 5 provides a balanced analysis of the advantages and disadvantages of the LLM agent service built on the actual HPC platform, while also discussing the limitations of the service and potential future research directions.

2 Backgrounds

2.1 Web Based HPC Platform

EDISON-DATA is a flexible and scalable data management platform designed to efficiently process and analyse large-scale computational science data. The software in question provides integrated support for diverse simulation results and workflows [9], The EDISON platform's integrated file management service has been demonstrated to enhance reproducibility and user convenience in computational science research. This is achieved by connecting heterogeneous computational resources and user tasks in a web-based environment, consistently providing data storage, access, and sharing [10]. Furthermore, the EDISON platform is a system that systematically analyses key functional elements affecting computational science and engineering user satisfaction based on its structure, which comprises the user interface, system reliability, service support, and computational resource accessibility [11]. The EDISON platform facilitates the integration of high-performance computational resources with the user environment through its web-based simulation execution structure. It has been developed to provide functions that enhance simulation execution efficiency and learning and research usability [12]. The present study builds upon the continuous development and refinement of the EDISON platform, a web-based high-performance computing (HPC) platform that has been systematically designed and evolved to support simulation execution for computational science and engineering applications requiring high-performance computing resources. The primary users of HPC platforms are general users, administrators and developers who create applications for HPC-based computational tasks. The following Fig. 1 illustrates the overall structure of the HPC platform that has been established thus far:

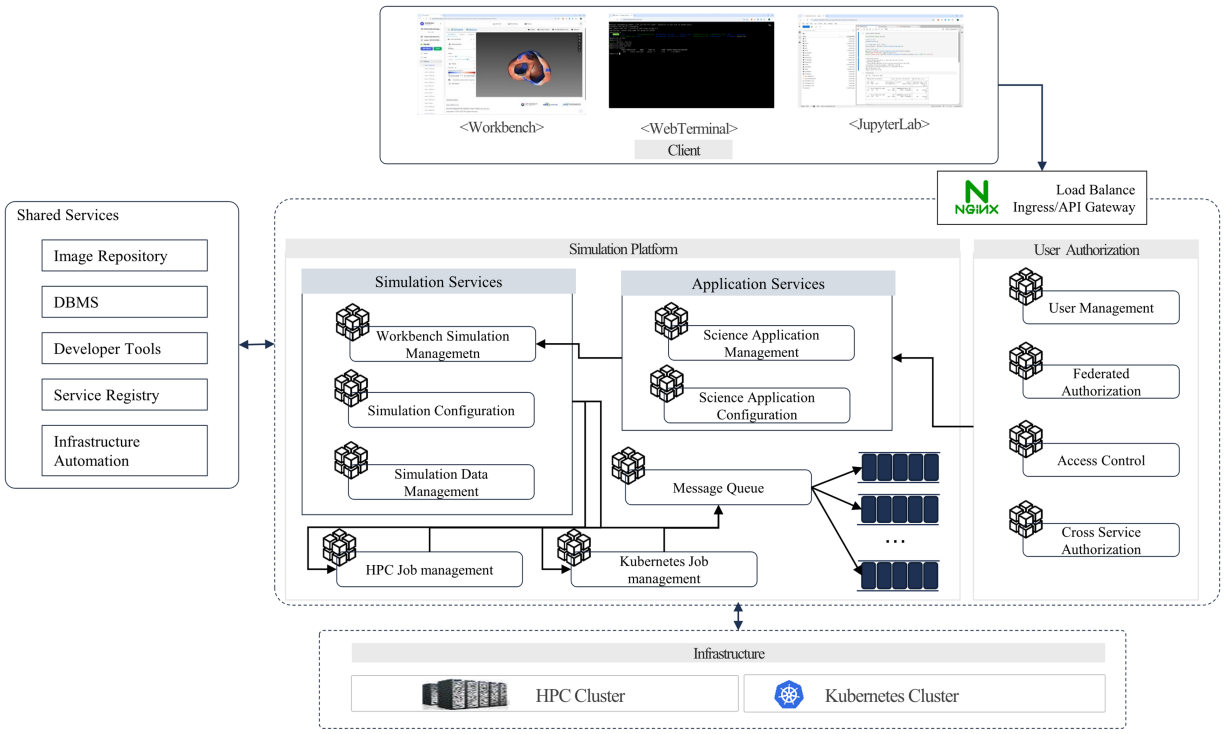

Figure 1: HPC platform architecture.

As demonstrated in Fig. 1, the simulation users have the capacity to access simulations through clients in a variety of simulation environments. The fundamental simulation environment furnished to users is equipped to facilitate HPC-based simulations, contingent upon the user's selected environment. Moreover, the simulation provides execution environments according to user selection, based on various applications registered on the platform. The system delivers appropriate system components as services, tailored to each configuration and execution environment within the infrastructure's runtime environment. The system is configured to run based on container images that define the user's settings and the various libraries or environmental elements required by the registered applications. In essence, applications registered on the HPC platform are characterised by a simulation environment that is fundamentally based on container images. The platform is constructed to incorporate services associated with these image management processes, data management, and automated build and deployment.

2.2 LLM (Large Language Model) Based Agent System

LLM agent services are having a profound impact across diverse industries and web environments [13]. In particular, all web-based services are now essentially incorporating LLM agent services as a fundamental component, and these systems are increasingly being adopted to enhance user convenience and expand service usage [14]. Moreover, the advent of the RAG architecture—which capitalises on pre-existing trained LLMs and integrates additional data with queries—has established a theoretical and empirical basis for the incorporation of versatile services. This facilitates the deployment of more specialised AI agents, which is a significant advancement beyond the mere retraining of a single LLM for service deployment. Furthermore, in the event that executable services exist that can be accessed through user queries using agents, the Model Context Protocol (MCP) can be defined. The protocol in question facilitates the execution of practical services and the subsequent return of results. It facilitates the incorporation of a more sophisticated system

that extends beyond rudimentary responses to user queries, incorporating the process of establishing connections with actual service tools.

2.2.1 RAG (Retrieval-Augmented Generation) System

Existing pre-trained LLM models necessitate substantial financial expenditure and resource consumption for training and optimisation purposes. Moreover, as LLM models themselves tend to grow increasingly large, retraining these models is inefficient. Notwithstanding optimisation for querying in conjunction with extant systems, there remains the disadvantage of constant retraining with new data to adapt to system data that is being generated continuously. Consequently, as opposed to the retraining of existing models, a system that supplies supplementary reference data to the model itself–namely, reference data for the purpose of responding to user queries–is designated RAG (Retrieval-Augmented Generation).

RAG signifies a process where an LLM retrieves relevant evidence from an external knowledge repository before generating an answer, augments that evidence into the context, and then conditionally generates the answer based on that augmented context [15]. This technique addresses the limitations of knowledge relying solely on model parameters and the typical LLM characteristic of failing to reflect the latest information in real time. The incorporation of real-time information input is intended to facilitate the delivery of the most appropriate and up-to-date responses to users.

The most recently prevalent form is the token-based RAG model. In this model, a different latent document is selected for each output token. At each decoding step, the generator marginalises over the top-K retrieved documents, thereby allowing token-level conditioning on multiple documents. The formula that is pertinent to this discussion is as follows:

$$p_{\text{RAG-Token}}(y \mid x) \approx \prod_{i=1}^{N} \sum_{z \in \text{top-}K(p(\cdot|x))} p_\eta(z \mid x)\, p_\theta(y_i \mid x, z, y_{1:i-1}). \tag{1}$$

Concretely, the top K documents are retrieved using the retriever, and then the generator produces a distribution for the next output token for each document, before marginalizing, and repeating the process with the following output token [15].

RAG is capable of responding to user queries based on the latest data without the need for retraining parameters. Existing pre-trained LLMs provide responses to queries without reflecting the latest information, leading to incorrect responses to queries based on recent events. Consequently, it is capable of updating importance based on the latest information and supporting diverse searches for time-sensitive queries [16]. Furthermore, RAG has been demonstrated to reduce hallucinations in LLMs and to appropriately find evidence, as referenced in [17]. This renders it a viable solution to address inherent model issues. Recent studies have proposed methodologies for mitigating hallucinations in LLMs through the utilisation of RAG, encompassing the iterative application of retrieval and generation processes [18,19], the delineation of workflows for the detection of hallucinations within RAG systems, the adjustment of internal probabilities to mitigate them, and the incorporation of verification modules based on evidence.

A significant benefit of utilising RAG systems is their capacity to facilitate the development of systems that can generate domain-specific responses without necessitating retraining of the LLM. Specifically, the system is employed in a variety of ways, including the construction of RAG systems based on specific company internal information or time-sensitive queries [20], and the creation of systems optimised for specialised domains such as the biomedical field [21,22]. In order to surmount the limitations of prevailing LLMs, RAG systems are being employed across a range of disciplines. The generation of user responses is

predicated on the provision of supplementary information derived from the most recent data or domain-specific data. This approach offers the advantage of reducing the resources and costs required for retraining the model itself. Consequently, RAG systems can be readily incorporated into existing platform services.

2.2.2 Model Context Protocol (MCP) System

The Model Context Protocol (MCP) is a protocol that facilitates interaction between large language models (LLMs) and external systems, domain services, or executable functions and structures. The use of MCP facilitates the connection of AI applications such as Claude or ChatGPT to a variety of data sources, including local files and databases, as well as tools such as search engines and calculators. In addition, these applications can be integrated into workflows, utilising specialized prompts to access crucial information and perform designated tasks. This capability is a significant advancement in the field of AI and has been extensively researched, as evidenced by the extensive bibliography provided in reference [23]. The MCP was initiated by Anthropic in November 2024.

The MCP server enables the MCP host and client to access external systems and execute operations, offering three core capabilities: tools, resources, and prompts. Tools in MCP enable the server to invoke external services and APIs to execute operations on behalf of AI models. When a host application (such as an AI assistant) needs to perform an operation, it first queries the MCP server through the client to obtain the list of available tools and their capabilities. Based on the task context, the host then selects an appropriate tool and issues an invocation request via the client. Resources provide access to structured and unstructured datasets that the MCP server can expose to AI models. Prompts are predefined templates and workflows that the MCP server generates and maintains to optimize AI responses and streamline repetitive tasks [24]. The structure is illustrated in the following Fig. 2:

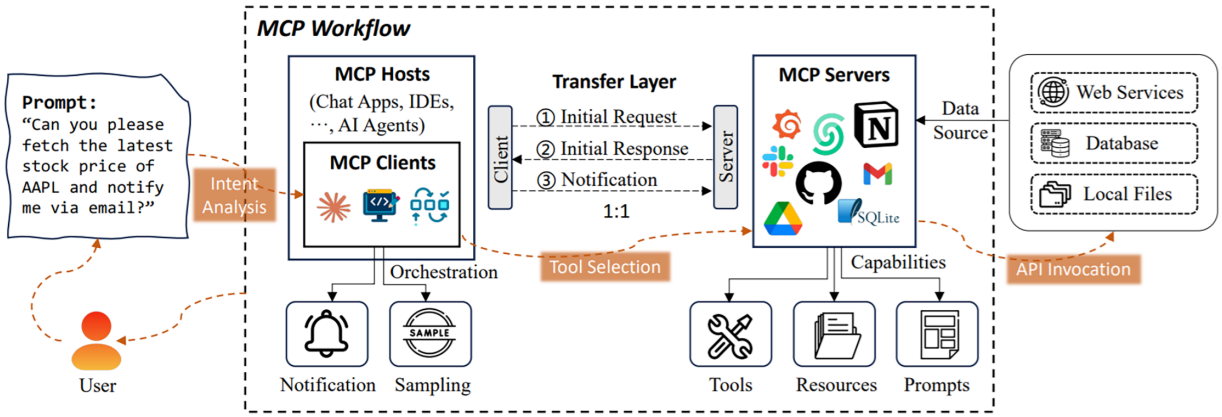

Figure 2: MCP architecture [24].

It is a common occurrence that MCPs possess a compendium of tools that can be utilised in conjunction with LLMs, which are endowed with predefined tool lists. The search for the appropriate tool is initiated in response to the user query, and the resultant list of available tools is returned. The system then selects these available tools, properly normalises the required information, calls external tools or service APIs, and returns a structured response for that call. Consequently, it can be regarded as a system defined as a protocol that invokes the relevant external service in response to the user's query and delivers the result to the user [25].

2.3 LLM Based Ontology

LLMs are inherently prone to hallucination issues in response to queries, with the potential for these hallucination problems to propagate according to the Chain-of-Thought (CoT) technique as outlined in the reference [26]. Consequently, a range of studies have sought to address this issue by employing existing, standardised prompting techniques for LLMs. Among these studies, research has been conducted to improve data normalisation and accuracy by utilising ontology-based Large Language Models and leveraging the results from each model. Recent studies in the field include research demonstrating that LLMs have the capacity to automatically learn ontologies from natural language text, thereby capturing complex linguistic patterns to structure domain knowledge as a tool [27]. In addition, research has been conducted on generating ontology drafts based on user requirements and stories using LLMs [28]. Moreover, studies have demonstrated that defining ontology quality refinement using ontologies enables LLMs to assess and supplement the logical quality at the ontology meta-level [29]. Research has also been undertaken to achieve greater accuracy than traditional engines in complex cases by identifying conceptual correspondences between different ontologies using an LLM-based matching system [30].

A plethora of endeavours have been undertaken to address issues such as hallucinations or problems arising from the chain of reasoning in extant LLMs using ontology techniques. Ontology can be regarded as a structured knowledge representation method that systematically specifies concepts, attributes, and relationships within a specific domain. In the field of information science, it is defined as a specification that defines semantic relationships between concepts, thereby enabling computers to understand and process knowledge. It is an established characteristic of such ontologies that understanding ontologies has the capacity to provide the semantic structure of knowledge. The generation of ontology-based knowledge graphs through the utilisation of LLM query results facilitates the support of high accuracy, class labeling, and logical consistency verification [29].

A plethora of ontology-based studies on LLM have been conducted. Recent studies have demonstrated that the integration of knowledge graphs with large language models (LLMs) can mitigate hallucination and enhance the precision of reasoning by utilising external structured information [31]. Furthermore, there exist studies that have proposed frameworks for integrating ontologies and LLMs into a RAG architecture to augment semantic consistency and knowledge-based reasoning capabilities [32]. Additionally, research has been conducted that proposes a biomedical code mapping method using OntologyRAG to concurrently enhance the search accuracy and processing speed of RAG by leveraging ontology-based knowledge graphs [33].

In this research, a RAG system linked to an HPC platform was built, and an AI agent system was constructed to connect user queries with simulation service tools available on the platform. In order to verify the logical consistency of each LLM's responses to user queries, an ontology-enhanced LLM system service was additionally utilized. Specifically, it structured responses to user queries in a manner that aligned with the platform's data format, thereby facilitating linkage with the platform's service tools.

3 Ontology Based RAG System

The present study has developed an AI agent based on an existing HPC-based web platform, established a process capable of merging data generated in real time, and constructed a workflow to optimise the selection and linkage of appropriate service tools in response to user queries. This objective was realised through the utilisation of an ontology-enhanced LLM system. Specifically, to provide accurate AI responses and service tool integration based on user queries, intermediate associations were established and relationships were defined/built using a knowledge graph. This approach has been demonstrated to resolve issues inherent in existing LLMs, such as hallucination and static data generation. A process for precise service tool integration

was provided, and a service was built that responds to user queries and executes HPC simulations by linking with various services required for simulation execution.

The existing High-Performance Computing (HPC) platform is responsible for the hosting of applications in the domain of computational science and engineering, the utilisation of which is registered by a variety of users. Furthermore, users are at liberty to execute simulations for each application. These simulations can be executed within an existing project or a new project. Consequently, utilising this framework, an analysis of user queries was conducted, resulting in the construction of a vector database that facilitates the integration of the HPC platform with the LLM model. The database under consideration contains three vector schemas. Utilising the extant relational database (RDB) as a foundational element, a RAG system was constructed, with the RAG system linked to the language model (LM) via three integrated schemas: The present document concerns an application, a project and a simulation job. Specifically, we employed ontoGPT [34] from the GPT family, which is capable of extracting ontology-based information from unstructured data and converting it into structured data that aligns with predefined ontology schemas and knowledge structures. This enhancement fosters seamless integration with service tools, thereby facilitating the precise invocation of designated service tools to execute simulation services.

3.1 Establishing LLM Agent Integration for HPC Platforms

The data utilised on the HPC platform is structured as a relational database, based on RDB, designed to store data such as user work history when executing new applications, projects, or simulations. At present, the platform has 249 registered applications, 336 projects, and 11,290 simulation jobs. It is important to note that this data is updated in real time. Although applications and projects do not undergo substantial growth, simulation jobs do exhibit a marked increase in data due to real-time execution by multiple users. In consideration of the characteristics of the simulation platform, the study periodically schedules tasks based on specific points in time to integrate data into the vector DB. Moreover, in the event of a simulation or project being created in response to a user query, the corresponding history is included in the user's agent history. As illustrated in Fig. 3, the architecture of the LLM agent system is linked to the HPC platform.

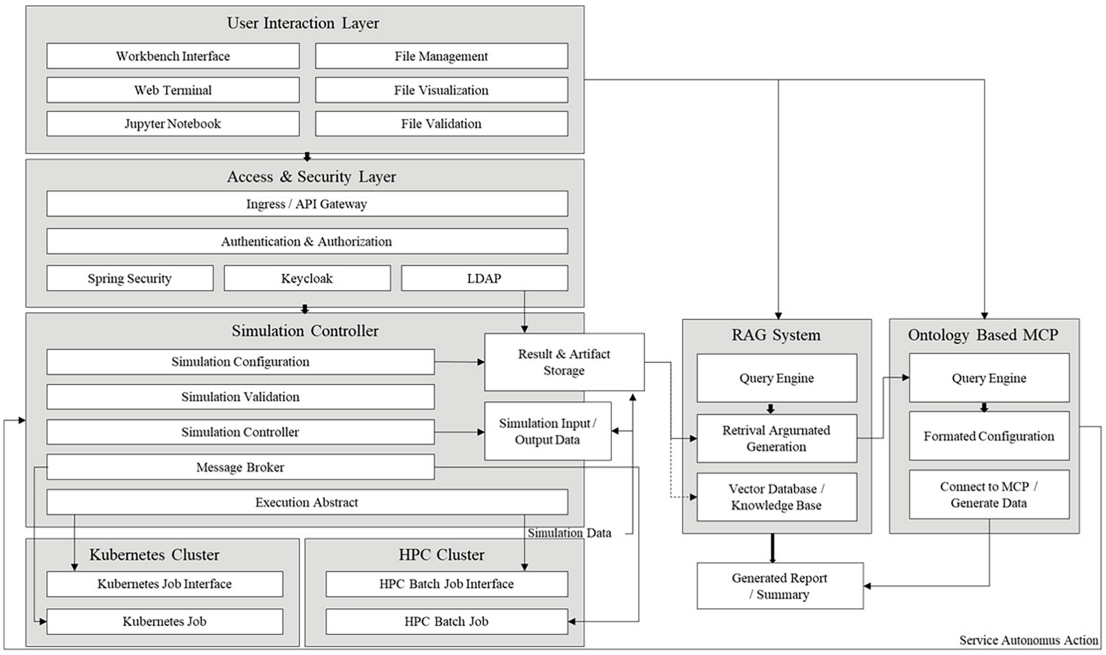

Figure 3: HPC platform architecture with AI agent.

As demonstrated in Fig. 3, a range of services exists that can execute various platform services, with the user interface acting as the starting point. It is evident that a controller for the simulation platform has been integrated with the HPC cluster, with the purpose of executing large-scale computations on behalf of the user services. The software also incorporates modules for user management and user access control for each service. The services ultimately provided by the platform include the execution of simulation tasks on the actual HPC cluster and the control of these tasks through user interface interaction. The software offers a range of features designed to assist users in the effective configuration of available resources and the establishment of simulation execution environments. Additionally, it facilitates the management of simulation data. The present study proposes a real-time data merging service as a means of integrating these services with agent services that utilise LLM. The service in question is responsible for addressing user queries regarding the platform, providing information pertaining to the service environment, and facilitating the execution of simulation services through its integration with compatible service tools.

3.2 Ontology Based RAG System

The process for handling user queries first utilizes an LLM router to perform necessary data analysis and user query analysis for the initial query. Subsequent to this analysis, the system establishes a connection to the relevant MCP tool for the requisite processing. In conclusion, the RAG system has been demonstrated to generate a suitable response to the user in accordance with query analysis. The workflow in which the AI agent responds according to the entire user flow can be represented as illustrated in the following Fig. 4.

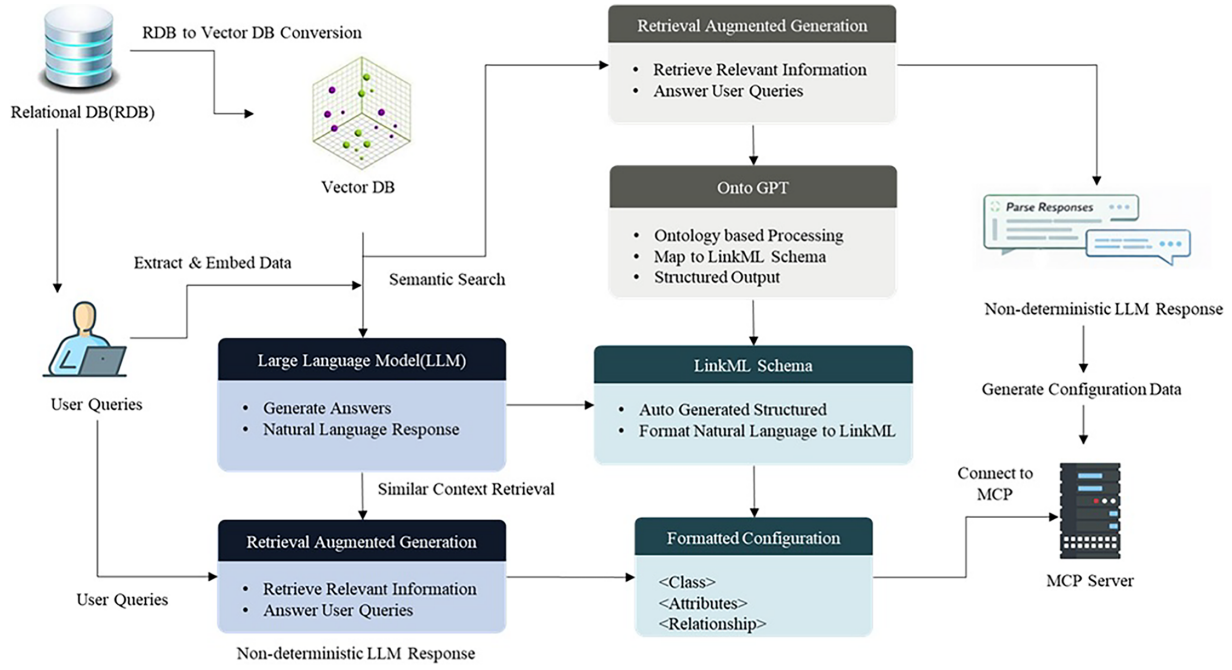

Figure 4: Ontology based MCP architecture.

As illustrated in Fig. 4, the RAG system is configured using user queries and data converted into a vector database based on various data types used in existing HPC platform services. When a user constructs a query and submits it to an agent utilising an LLM, the system constructs a prompt alongside the vector-converted data and transmits the query to the LLM. The LLM is capable of generating an appropriate response using this data, creating its response based on the retrieved data and delivering it to the user. Moreover, the present

vector DB maintains three schemas internally. The system must determine which schema to use for searching the user's query. The LLM also provides routing functionality to facilitate this decision.

Firstly, an LLM routing function has been developed to determine which vector schema should be used as the basis for providing data according to the user's query. The function of the RAG schema is to determine which schema to search for data to include. This decision is the foundation for the incorporation of search results from the designated schema into the RAG prompt. The subsequent transmission of these results and the generation of an appropriate response for the user are then facilitated.

The second process employs additional vector data for search to standardise the LLM's output data based on ontoGPT, thereby enabling the creation of a predefined ontology-based knowledge graph. This standardized data can then be linked to actual executable service tools, facilitating the selection of appropriate service tools.

3.2.1 Differences from Previous Studies

Of the existing studies, the one that is most closely related is [33], which proposed a method of mapping biomedical codes that utilises an ontology-based knowledge graph to improve the accuracy of searches and the speed of processing in RAG simultaneously. Unlike conventional RAG approaches, this study did not use a text similarity-based search. Instead, it constructed an ontology graph and used an ontology graph-based search during the retrieval phase. In other words, the study's contribution lies in supporting knowledge graph-based searches defined by ontologies rather than vector-based similarity searches for user queries, thereby enabling application to specific fields. Compared to existing RAG systems, this can be said to have improved both accuracy and processing speed.

The method proposed in this study does not simply rely on RAG; it incorporates an ontology-based process to define the relationships between RAG systems and the service tools defined within the platform. This establishes the relationships between RAG systems and the service tools that must be invoked based on the results generated by RAG.

Many existing ontology-enhanced LLM systems or knowledge graph-based RAG systems limit their use of ontologies to semantic expansion or filtering during the search phase. Furthermore, most existing studies employ a structure in which retrieval results are fed directly into the LLM with little consideration given to whether the LLM's response is relevant to the service tools. Additionally, due to the structural characteristics of HPC platforms, various data formats exist. Unlike general prompts, the available data formats on each platform are unique. Therefore, even if existing data is input into the LLM in RAG format, it is difficult to obtain responses in the desired format. Even if an accurate response is received, there is no guarantee that the associated service tools will be invoked correctly. Therefore, this study employs ontologies to address the structural issues of RAG by defining structures related to specific situations, thereby enabling a precise understanding of the platform's architecture. In contrast, this study uses ontologies as a core module for consistently normalising the representation of input data, not merely as search aids. Notably, this approach differs from existing ones in that it minimises discrepancies between retrieved information and queries by introducing an MCP-based structure to perform semantic alignment prior to model input.

Specifically, the contributions of this study are as follows:

- We have designed an integrated framework that uses the MCP structure to align the semantic content of retrieval results with queries.
- We have proposed a new pipeline to mitigate semantic mismatch issues arising from existing RAG-based approaches, as well as defining integration with service tools.

Therefore, unlike existing ontology-based RAG or knowledge graph augmentation approaches, this study aims to improve performance through structural enhancements ranging from user query processing to actual service execution that go beyond simple information augmentation.

3.2.2 Definition of the Algorithm

The detailed process is described through the Algorithms 1–3. Algorithm 1 delineates the process of analysing user queries by employing an LLM as a router function to define the necessary data schema and other elements for the user's intended question and response.

Algorithm 1: Query analysis and retrieval planning

Require: Query q, user context U, entity set $\mathcal{C} = \{\text{Job}, \text{Application}, \text{Project}\}$
Ensure: Analysis object $A(q) = (\mathcal{C}_q, K_q, F_q, M_q)$

$A(q) \leftarrow \text{LLMAnalyze}(q)$
if *A(q)* is invalid **then**
  $\mathcal{C}_q \leftarrow \text{RuleFallback}(q)$
  $K_q \leftarrow \varnothing,\ F_q \leftarrow \varnothing,\ M_q \leftarrow \varnothing$
else
  $\mathcal{C}_q \leftarrow A(q).\text{entity\_types}$
  $K_q \leftarrow A(q).\text{search\_keywords}$
  $F_q \leftarrow A(q).\text{filters}$
  $M_q \leftarrow A(q).\text{mcp\_tools}$
end if
if FirstPerson*(q)* **and** $U \neq \varnothing$ **then**
  $F_q[\text{created\_by}] \leftarrow U$
end if
if $\text{job\_status} \in \text{dom}(F_q)$ **then**
  $F_q[\text{job\_status}] \leftarrow v(F_q[\text{job\_status}])$
end if
return *A(q)*

Algorithm 1 delineates the configuration of an LLM service that functions as a router, which determines the schema to be utilised by the LLM for data analysis and user response. Specifically, it is an algorithm that determines which data – from the three types of schemas (Job, Application, Project) currently built in the Vector DB—should be retrieved and delivered to the user in response to their query. The prompt for the query must determine which existing schema to use as the basis for the response. The function of this system is to analyse which service tool should be selected for the user query, and thereby ultimately receive the final result. Specifically, the final returned value $A(q) = (\mathcal{C}_q, K_q, F_q, M_q)$ includes: The search schema is specified by C_q, the keyword is a key factor for searching that vector schema, K_q; the filter is represented by F_q; and M_q contains a list of service tools to execute based on the user query, along with the responses for those tools.

The subsequent Algorithm 2 facilitates the selection of the appropriate MCP tool in response to the user's query. It subsequently invokes the process to define and execute the precise platform behaviour, which is then appended to the response according to the user query.

Algorithm 2: OntoGPT-Augmented MCP tool selection

Require: q, $\mathcal{M}$, $\mathcal{V}$, U
Ensure: $\mathcal{M}_q$, m_{nav}
$\hat{\mathcal{A}} \leftarrow \mathcal{F}_{\text{route}}(q, \mathcal{M})$
$\mathcal{R} \leftarrow \mathcal{F}_{\text{retr}}(\mathcal{V}, q \mid \hat{\mathcal{A}}, U)$
$(\mathcal{O}, \mathcal{S}) \leftarrow \mathcal{G}_{\text{onto}}(\mathcal{R})$
$\mathcal{M}_q \leftarrow \{\, m \in \mathcal{M} \mid \phi(m, q, \hat{\mathcal{A}}, \mathcal{S}, U) = 1 \,\}$
$m_{\text{nav}} \leftarrow \arg \max_{m \in \mathcal{M}_q \cap \mathcal{M}_{\text{nav}}} s_{\text{nav}}(m, q, \hat{\mathcal{A}}, \mathcal{S}, U)$
return $\mathcal{M}_q$, m_{nav}

The prevailing MCP configuration facilitates the dissemination of suitable responses to users in accordance with LLM responses. A potential disadvantage of this configuration is the elevated probability of LLM hallucinations and the possibility of inadequate utilisation of pertinent data in structured MCP calls. Therefore, in order to address the aforementioned shortcomings and ensure accurate delivery of data required by executable service tools, a process was defined that connects data and appropriate tools based on ontology-based GPT. As with the Algorithm 2, the user query q, the available service tools M, the vector DB V, U represents the user context, and $\hat{\mathcal{A}}$ denotes the set of estimated vector database schema types derived from the user query. It is proposed that M_q can be regarded as the set of MCP service tools considered executable for the user-input query q, the semantic normalization result S, and the user context U. The set of dedicated navigation tools is denoted by m_nav. This tool is indispensable in executing the navigation tool for specific data by default, thereby reflecting the user query flow within the selected tool set M_q. For instance, to generate and execute a task based on a specific application, the M_q for task generation is selected. It is also responsible for generating the data necessary for the execution of the task, and, in essence, necessitates the utilisation of a navigation tool to access the generated task. The program was structured to incorporate these requirements, enabling an MCP tool invocation that includes two execution tools.

Algorithm 3 delineates the process of delivering a response to the user for their query, including the result data based on the user's query and the execution results of the MCP service.

Algorithm 3: Semantic RAG answer generation

Require: Query q, analysis $A(q)$, embedding model E, databases $\{\mathcal{D}_c\}$
Ensure: Streamed answer tokens $\{a_t\}$
$\mathbf{v}_q \leftarrow E(q)$
$S \leftarrow \varnothing$
for each $c \in A(q).\mathcal{C}_q$ **do**
  $W_c \leftarrow \Phi_c(A(q).F_q)$
  $S_c \leftarrow \text{ANN}_c(\mathbf{v}_q, W_c)$
  $S \leftarrow S \cup S_c$
end for
$S^\star \leftarrow \text{Unique}(S)$
for each $x \in S^\star$ **do**
  if *d(x)* exists **then**
    $s(x) \leftarrow \frac{1}{1+d(x)}$
  else
    $s(x) \leftarrow u(x)$

(Continued)

Algorithm 3 (continued)

```
    end if
end for
R ← TopK(Sort(S*, s), k)
if R = ∅ then
    R ← TextFallback(q)
end if
C ← BuildContext(R)
for each token a_t ~ LLMStream(q, C) do
    output a_t
end for
```

The selection of an appropriate vector database schema and service MCP tool via the previously defined Algorithms 1 and 2 is followed by the final definition of the process of delivering the generated answer to the user 3. Specifically, it seeks to identify the schema to be utilised as the foundation for generating the response, based on the semantically derived result $A(q)$ from analysing the user's query q. The symbol $A(q).C_q$ is used to denote the set of schema collections that have been selected based on the results of the user query that have been semantically analysed. This signifies the exploration of a semantically relevant subset of the entire database space, which is denoted by D_c. It is evident that for each collection $c \in A(q).\mathcal{C}_q$, $\Phi_c(A(q).F_q)$ is the generation of conditions is achieved through the application of appropriate filters. It is evident that the function $\mathrm{ANN}_c(\mathbf{v}_q, W_c)$ performs a conditional nearest neighbor search, returning the final result S as the search outcome. The final result, S, obtained through this search is normalized using the vector distance, $d(x)$, and the system's defined confidence and similarity metric, $u(x)$. This process forms the top-selected result set, R. It is the final search set R that is the basis for the LLM to construct a prompt conveying context. The LLM generates responses token by token, with each token being output to the user immediately, thus defining this as the process.

4 Evaluation

The purpose of the LLM-based agent developed in this study is twofold: firstly, to enhance the usability and scalability of existing platforms; and secondly, to ensure user convenience. This objective is realised through the integration of MCP service tools, thereby empowering the agent to invoke and execute pre-existing service tools based on data present across various platforms. As of the evaluation point, the size of data existing on the respective platforms is as shown in Table 1.

Table 1: Scheme-wise resource counts.

Scheme	Application	Project	Job
Count	249	336	11,279

It is improbable that applications or projects will increase or decrease exponentially; however, there is potential for jobs to grow steadily by tens to hundreds on a daily basis. Furthermore, the specifications of the service servers that are utilised for the LLM service and the HPC platform are as follows Table 2.

Table 2: Service-wise resource configuration.

Service	Nodes	GPU	CPU
Service Cluster	12	H100 × 6	2112
LLM Service	1	V100 × 2	96

The service cluster is defined as the combined numerical value of the cluster responsible for computation and the service server that hosts various HPC service tools. The utilisation of LLM can be categorised into three distinct scenarios: the utilisation of ChatGPT, the utilisation of the OpenAI API, and the utilisation of the Ollama service server responsible for LLM services. Presently, the GPU employed by the Ollama service is the V100.

In order to ensure an accurate comparison of each model, the same hyperparameters and token structure were utilised for all models. The values of the hyperparameters are shown in Table 3.

Table 3: Decoding and retrieval hyperparameters used across all models.

Parameter	Value
Model	ChatGPT/Gemma/Qwen/GPT-4.1-mini
Temperature	0.3
Max Tokens (`num_predict`)	2048
Context Window (`num_ctx`)	4096
Top-p (Nucleus Sampling)	1.0 (disabled)
Frequency Penalty	0.0
Presence Penalty	0.0
Random Seed	Not specified
Retrieval Top-k	15

In order to ensure a fair comparison, all models (ChatGPT, Gemma, Qwen, and GPT-4.1-mini) were evaluated under identical decoding settings. Specifically, the temperature parameter was set to 0.3, the maximum generation length was limited to 2048 tokens, and the context window size was fixed at 4096 tokens. It is evident that no nucleus sampling (top-p), frequency penalty, or presence penalty was applied. Frequency and presence penalties were set to zero, and top-p sampling was disabled to ensure consistent and controlled generation behavior across all models.

For the purpose of retrieval, a constant top-k value of 15 was employed in all experiments. It should be noted that solely top-k based retrieval was utilised, with no combination with top-p sampling, in order to maintain consistency and avoid introducing additional variability in the comparison. Furthermore, the identical system prompts were utilised for all experiments, and evaluations were conducted by modifying solely the models.

4.1 Evaluation LLM Model and Statistical Validation Method

The subsequent section delineates the outcomes of experiments conducted for each LLM model, employing mathematical formulae. In the course of these experiments, queries were executed for each schema, and the results were analysed. Each formula is defined on the basis of the response for a single

sample. Formula (1) calculates the probability that the LLM finds the relevant sample in the Source and that the 'navigate' result returns the correct ID of that sample. As delineated in Table 1, queries corresponding to the resource count for each schema were created and tests were conducted accordingly.

Let N denote the total number of test queries. For each query $i \in \{1, \ldots, N\}$, we define the following binary indicators:

$$s_i^{\text{Name}} = \begin{cases} 1, & \text{if the answer contains the target sample's name} \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

$$s_i^{\text{NavSucc}} = \begin{cases} 1, & \text{if the navigation API call succeeds} \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

$$s_i^{\text{NavAcc}} = \begin{cases} 1, & \text{if the returned sample's ID matches the ground truth} \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

$$r_i = \begin{cases} 1, & \text{if the correct sample is found in the retrieved sources} \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

To explain each value, s_i^{LLM} indicates whether the LLM successfully processed the query correctly, while s_i^{Name} measures whether the exact name of the sample in question is included in the LLM's response. Additionally, s_i^{NavSucc} tests whether the response includes a call to a navigation tool that can direct the user to the page defining the specific sample within the platform corresponding to the query, while s_i^{NavAcc} measures whether the navigation tool returns a sample with an ID that exactly matches the query. Finally, r_i is a binary value indicating whether the RAG system includes the target sample within the retrieved sources.

The following defines the formula for calculating the probability based on the mathematical definitions provided above.

$$\text{Answer Contains Name} = \frac{1}{N} \sum_{i=1}^{N} s_i^{\text{Name}} \tag{6}$$

$$\text{Navigation Success} = \frac{1}{N} \sum_{i=1}^{N} s_i^{\text{NavSucc}} \tag{7}$$

$$\text{Navigation Accuracy} = \frac{1}{N} \sum_{i=1}^{N} s_i^{\text{NavAcc}} \tag{8}$$

$$\text{LLM Success} = \frac{1}{N} \sum_{i=1}^{N} r_i \cdot s_i^{\text{NavAcc}} \tag{9}$$

N is determined based on the resource count values defined in the schema in Table 1. In other words, for all sample data stored according to the schema, a query is sent to the LLM model, and the final success probability is calculated based on the defined values and the results of that response.

Statistical Validation Method

As previously outlined, the findings derived from the evaluation of each model are articulated as binary outcomes. The statistical analysis of these binary test results involves the implementation of both Cochran's Q test and McNemar's test in a combined approach.

- Cochran's Q test

The Cochran test, also known as Cochran's Q test, is a statistical procedure used to assess the consistency of proportions across multiple groups in a dichotomous dataset [35]. The employment of Cochran's Q test was undertaken in order to evaluate the existence of statistically significant differences among k related binary treatments. The Cochran test is a statistical procedure that is employed to ascertain whether the results x_{ij} are distributed uniformly over the interval $[0,1]$ for a given set of binary treatments (or conditions) $j = 1, \ldots, k$, when applied to the same set of subjects $i = 1, \ldots, N$. In the event of multiple objects being present, the results for the same jth condition are calculated. The row sum for that condition is defined as R_i, the column sum as C_j, and the total sum as T, as follows. $x_{ij} \in \{0,1\}$ denotes the result for the i-th object under the j-th condition.

$$R_i = \sum_{j=1}^{k} x_{ij} \tag{10}$$

$$C_j = \sum_{i=1}^{N} x_{ij} \tag{11}$$

$$T = \sum_{i=1}^{N} \sum_{j=1}^{k} x_{ij} \tag{12}$$

For this equation, we define the null hypothesis and the alternative hypothesis. The null hypothesis H_0 states that the probability of success for each object is the same under all conditions, and that no condition acts to the advantage or disadvantage of any particular object.

$$H_0 : P(x_{i1} = 1) = P(x_{i2} = 1) = \cdots = P(x_{ik} = 1) \tag{13}$$

In contrast, the alternative hypothesis H_1 states that the probabilities differ under at least one condition, and that the probability of success may differ under some of these conditions.

$$H_1 : \text{At least one } P(x_{ij} = 1) \text{ differs} \tag{14}$$

Under this hypothesis, the test statistic Q is defined as follows.

$$Q = \frac{(k-1)\left[k\sum_{j=1}^{k} C_j^2 - T^2\right]}{kT - \sum_{i=1}^{N} R_i^2} \tag{15}$$

In Q, k denotes the number of conditions; in this paper, since the conditions being compared refer to different models on the same test, k represents the number of models to be compared. The numerator of the equation, $(k-1)\left[k\sum_{j=1}^{k} C_j^2 - T^2\right]$, represents the number of successes per condition; the greater the difference between conditions—that is, the greater the difference between models—the larger this value becomes. It can be viewed as a measure of the degree of imbalance among the conditions. The denominator, $kT - \sum_{i=1}^{N} R_i^2$, corrects for the basic tendency toward success inherent in each object by reflecting the number of successes per object, R_i. This has the effect of blocking out bias between objects. Therefore, a small value of Q indicates that there is almost no difference between the conditions, which means that the null hypothesis H_0 is supported. If the value of Q increases, it indicates a large difference between the conditions, meaning that H_0 is rejected and H_1 is accepted. Under the null hypothesis, the distribution of Q is defined as follows.

$$Q \sim \chi_{k-1}^2. \tag{16}$$

According to the given distribution, if the value of Q is smaller than χ^2_{k-1}, the null hypothesis is retained; if Q is large, the null hypothesis is rejected.

$$Q > \chi^2_{k-1,\alpha/2} \Rightarrow \text{Reject } H_0 \tag{17}$$

In other words, if the results for the same query across the models tested in this study follow the same distribution rather than being random, the value of Q will be small, and we can verify that the models share the same distribution.

- McNemar's test

McNemar's test is performed to examine the differences between pairs of conditions when Cochran's Q test yields a significant result [36]. The McNemar test is a non-parametric statistical test that is utilised for the analysis of differences in responses to two binary conditions (or stimuli) within the same population. The objective of this test is to compare the frequencies of two discordant cells in a 2×2 contingency table, thereby ascertaining whether the two conditions have the same probability of response. Under the null hypothesis, it is assumed that the two conditions elicit the same response, and any difference in the frequencies of the two discordant cells is considered to be due to chance. The McNemar test is a statistical procedure that is employed to evaluate the statistical significance of the observed difference between the two cells. The purpose of this evaluation is to ascertain whether there is a response difference between the two conditions [37]. When a significant difference was observed in Cochran's Q test, pairwise comparisons between conditions were performed using McNemar's test as a post-hoc analysis. For each pair of conditions, the test statistic is calculated as follows.

$$\chi^2 = \frac{(|b - c| - 1)^2}{b + c} \tag{18}$$

This statistic follows a chi-square distribution with one degree of freedom. Additionally, a Bonferroni correction was applied to control for the family-wise error rate resulting from multiple comparisons. The p-value can be computed from the calculated χ^2 statistic to determine whether the difference between the two compared objects is statistically significant. The formula for computing the p-value is as follows.

$$p\text{-value} = P\left(\chi^2_{df} \geq \text{observed } \chi^2\right) \tag{19}$$

The significance level α used to determine whether a p-value is statistically significant is typically set to 0.05. That is, when $\alpha = 0.05$, if the p-value < 0.05, the probability that the observed result occurred by chance is considered sufficiently small, and the result is regarded as statistically significant.

4.2 Evaluation of User Responses across Different LLM Models

The evaluation in question serves to ascertain whether responses to user queries can be generated while simultaneously linking them to the appropriate MCP service tool. Specifically, the evaluation process encompasses the assessment of the adequacy of the response generation, the degree of correlation between the generated response and the service tool, and the alignment of the content of the user's query with that of the service tool. The function of this process is to measure whether the content of the URL resource returned by the navigate tool corresponds to the ID of the user's query content. The purpose of this is to assess whether the system accurately identified the user's actual intent. This value is referred to as Navigation Accuracy. A variety of MCP service tools have been defined for the purpose of establishing links with specific service tools. Among these tools, there are those that generate URLs capable of navigating to specific locations matching each schema. It is asserted that, based on the aforementioned tools, the process of defining the accuracy and

appropriateness with which the tools are created to provide specific navigation for each schema currently existing in the vector DB is undertaken.

The LLM models used in this study are as follows Table 4.

Table 4: Comparison of representative large language models.

Model	Provider	Scale	Pricing	Reference
ChatGPT-3.5-turbo	OpenAI	Proprietary	Paid (API)	[38]
Gemma-7B	Google	~7B	Free	[39]
Qwen3-14B	Alibaba	~14B	Free	[40]
GPT-4.1-mini	OpenAI	Proprietary	Paid (API)	[41]

The following figure, referred to as Fig. 5, is a graph that analyses the responses received for the entire dataset based on Application, Project, and Job, using the schema provided on the HPC system with four LLM models that has been built so far.

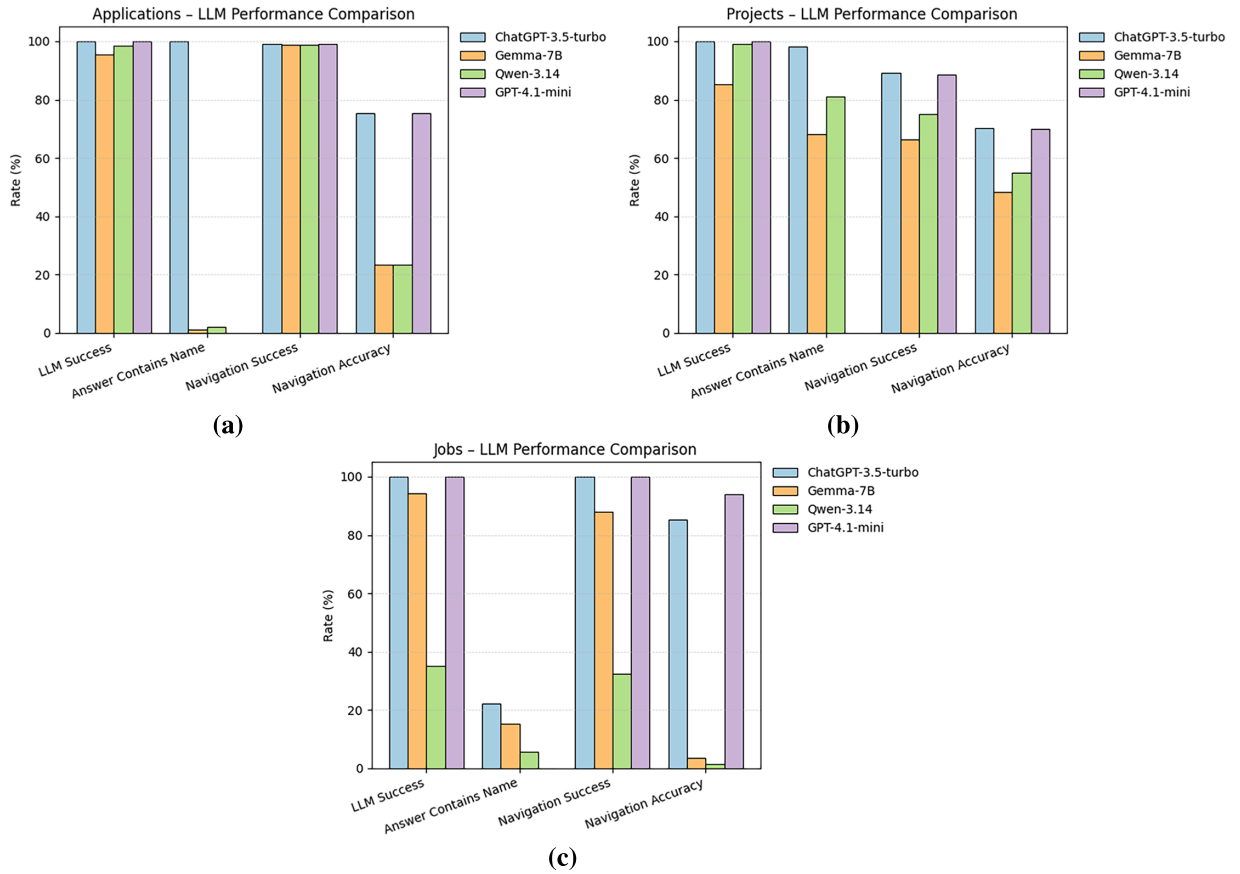

Figure 5: Comparison of LLM performance across different task domains. (**a**) Application-level performance comparison of ChatGPT-3.5-turbo, Gemma-7B, Qwen-3.14, and GPT-4.1-mini, evaluated in terms of LLM success rate, answer name inclusion rate, navigation success rate, and navigation accuracy. (**b**) Project-level performance comparison of the same LLMs, highlighting differences in semantic reasoning and entity navigation accuracy across models. (**c**) Job-level performance comparison, models are evaluated on the full job set 11,279. All bars represent percentage rates. The legend indicates the corresponding LLM for each color.

In the application scheme (see Fig. 5a), ChatGPT-3.5-turbo consistently demonstrates strong performance, with a navigation accuracy of 75.5% and a navigation success rate of 99.2%, while correctly including the application name in 100% of responses. In contrast, both Gemma-7B and Qwen-3.14 exhibit severe degradation in entity naming, with answer-contains-name rates of only 1.2% and 2.0%, respectively, resulting in much lower navigation accuracy (23.3% for both models) despite high navigation success rates. GPT-4.1-mini has been shown to demonstrate comparable navigation accuracy (75.5%) and navigation success (99.2%) to ChatGPT-3.5-turbo. However, it is notable that GPT-4.1-mini does not explicitly mention entity names (0%), suggesting that correct navigation can be achieved through implicit reasoning, even in the absence of explicit name generation.

As illustrated in Fig. 5b, the project-level comparison reveals a reduction in performance disparities among models in comparison to the application domain. ChatGPT-3.5-turbo achieves a navigation accuracy of 70.2%, thus establishing a robust upper baseline. Qwen-3.14 achieved a success rate of 55.1%, surpassing Gemma-7B's 48.2% in both navigation success and accuracy. This outcome indicates a more robust semantic reasoning capability at the project level. GPT-4.1-mini has been demonstrated to achieve the highest navigation accuracy of all non-ChatGPT models, attaining 69.9% accuracy, a figure that approaches the performance of ChatGPT-3.5-turbo, despite the absence of explicit project names in the model's output. The project domain as a whole demonstrates that semantic context has the capacity to compensate, at least in part, for the absence of explicit entity naming. However, it is evident that clear performance differences remain between models.

As demonstrated in Fig. 5c, the job domain is characterised as the most challenging setting. ChatGPT-3.5-turbo maintains high performance with a navigation accuracy of 85.3%, whereas Gemma-7B and Qwen-3.14 experience dramatic drops, achieving only 3.4% and 1.5% navigation accuracy, respectively. Despite the fact that Gemma-7B exhibits a high rate of successful navigation (87.9%), erroneous entity resolution results in a near-failure in accurate job identification. GPT-4.1-mini exhibits a high conditional navigation accuracy of 93.9%, signifying strong precision when execution is successful. The findings underscore the necessity for precise entity disambiguation in job-level queries, rendering them particularly susceptible to limitations imposed by LLMs.

Statistical Validation

The results calculated using statistical verification methods based on the data in Fig. 5 are shown in the following Table 5.

Table 5: Results of Cochran's Q test across schemas.

Schema	N	Cochran's Q	*p*-Value	Interpretation
Application	249	167.3057	4.856×10^{-36}	$<\alpha$
Job	11,279	43.6875	1.758×10^{-9}	$<\alpha$
Project	336	501.1515	2.686×10^{-108}	$<\alpha$

A total of four models (ChatGPT-3.5-turbo, Gemma-7B, Qwen-3.14, and GPT-4.1-mini) were used. When comparing these four models, the results for each schema were all calculated to be less than $\alpha = 0.05$, indicating that the results for each model are statistically significant. The results of the McNemar's test performed based on these Cochran's Q results are shown in Table 6 below.

Table 6: Pairwise McNemar test results grouped by model comparison across schemas.

Comparison		Schema	p-Value	Bonferroni Corrected	$\alpha = 0.05$
chatgpt3.5	gemma7b	Application	2.776×10^{-17}	1.665×10^{-16}	$< \alpha$
		Job	2.441×10^{-4}	1.465×10^{-3}	$< \alpha$
		Project	5.474×10^{-48}	3.284×10^{-47}	$< \alpha$
chatgpt3.5	gpt-4.1-mini	Application	1.000	1.000	$> \alpha$
		Job	1.000	1.000	$> \alpha$
		Project	1.000	1.000	$> \alpha$
chatgpt3.5	qwen3.14b	Application	1.388×10^{-17}	8.327×10^{-17}	$< \alpha$
		Job	1.526×10^{-5}	9.155×10^{-5}	$< \alpha$
		Project	4.078×10^{-56}	2.447×10^{-55}	$< \alpha$
gemma7b	gpt-4.1-mini	Application	2.776×10^{-17}	1.665×10^{-16}	$< \alpha$
		Job	2.441×10^{-4}	1.465×10^{-3}	$< \alpha$
		Project	5.474×10^{-48}	3.284×10^{-47}	$< \alpha$
gemma7b	qwen3.14b	Application	1.000	1.000	$> \alpha$
		Job	0.125	0.750	$> \alpha$
		Project	1.490×10^{-8}	8.941×10^{-8}	$< \alpha$
gpt-4.1-mini	qwen3.14b	Application	1.388×10^{-17}	8.327×10^{-17}	$< \alpha$
		Job	1.526×10^{-5}	9.155×10^{-5}	$< \alpha$
		Project	4.078×10^{-56}	2.447×10^{-55}	$< \alpha$

In this study, Cochran's Q test was employed to assess performance disparities across k associated binary conditions. The analysis yielded statistically significant differences between the conditions. Consequently, McNemar's test was conducted as a post-hoc analysis to examine pairwise differences between conditions in greater detail. In order to control for Type I errors resulting from multiple comparisons, the Bonferroni correction was applied, and statistical significance was determined based on the corrected significance level $\alpha' = \frac{\alpha}{m}$. Consequently, statistically significant variations were identified in specific pairs of conditions ($p > \alpha'$), while no significant variations were detected in other pairs ($p > \alpha'$). This finding suggests that, while a substantial discrepancy exists among all conditions, this variation does not manifest consistently across all possible pairs of conditions. This outcome is attributable to the fundamental characteristics of the two statistical tests. Cochran's Q test is a global test that evaluates the presence of an overall difference, whereas the McNemar test is a paired test that evaluates differences between individual pairs of conditions. In other words, even if a statistically significant difference exists overall, that difference may be driven by specific pairs of conditions, and the same level of difference may not appear in all comparisons between conditions. Furthermore, since the Bonferroni correction reduces the significance level in order to control Type I errors resulting from multiple comparisons, there is a possibility that statistically significant differences will not be detected even when they are actually present. Consequently, the absence of statistical significance observed in certain condition pairs can be attributed to the application of conservative testing criteria. The results of the study indicate that the observed differences between conditions are not uniformly distributed, but rather manifest primarily in specific pairs of conditions. This suggests that performance differences between conditions may emerge selectively.

4.3 Ontology-Based Qualitative Evaluation

The model currently configured and deployed on the platform is Gemma-7B. Utilising this model as a foundation, the subsequent figure presents a comparison of the three schemas from the prior Section 4.2 and the disparities observed when employing the actual ontology.

Fig. 6 compares Gemma-7B performance with and without the application of the application ontology across all LLM-level and exploration-level metrics, excluding the overall success metric. The utilisation of the application ontology has been demonstrated to engender consistent enhancements in both structural and task-oriented behaviour. It is noteworthy that the entity inclusion rate exhibited a substantial increase from 1.2% to 97.6%, signifying that ontology constraints significantly motivate the model to explicitly denote target entities. Search accuracy also underwent a substantial enhancement, rising from 23.3% to 75.5%, thereby indicating that ontology-based guidance effectively reduces the behavioural space during the search process. However, ontology integration resulted in the complete elimination of source-grounded responses, thereby reducing the source-grounded rate from 37.3% to 0%. This finding indicates that while ontologies enhance precision and decision consistency, they may impose excessive constraints on response generation, thereby impeding the model's capacity to leverage retrieved evidence. The findings of this study demonstrate a trade-off between structural accuracy and evidence grounding when integrating application ontologies.

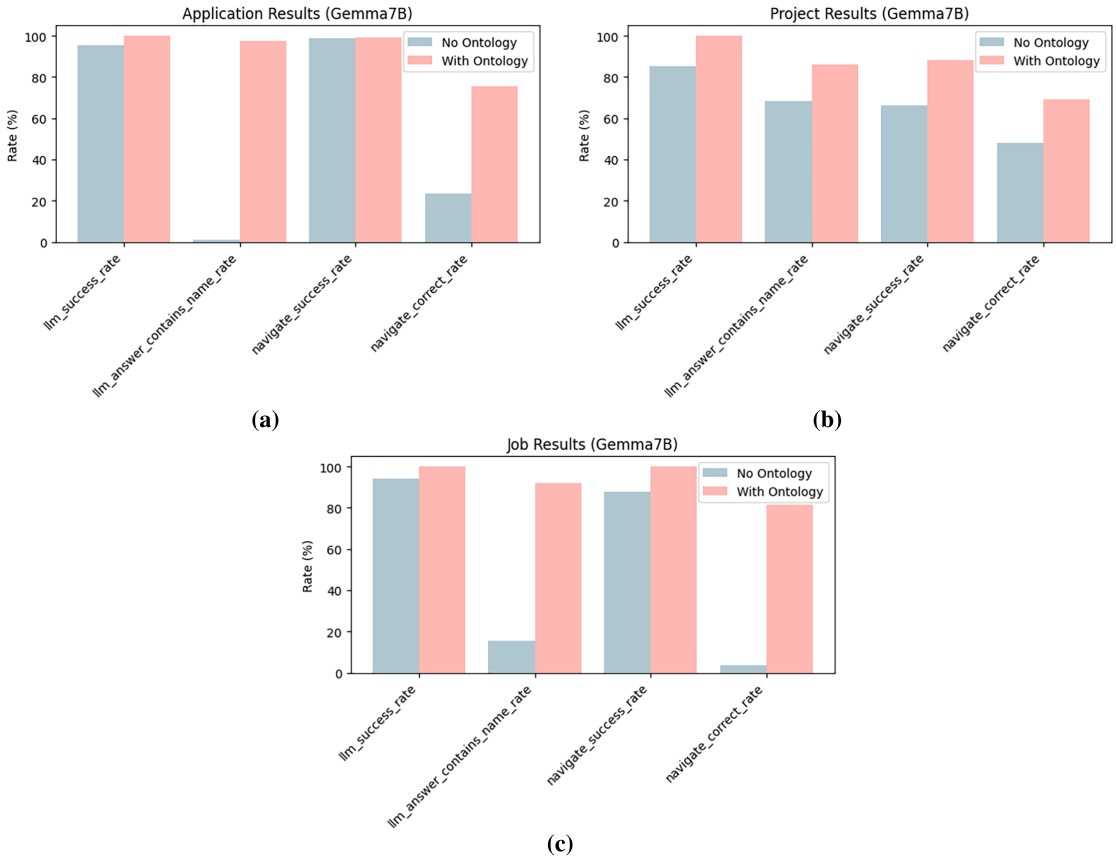

Figure 6: Comparison of ontology usage based on the Gemma-7B model. (**a**) Application-level performance with and without ontology comparison of Gemma-7B evaluated in terms of LLM success rate, answer name inclusion rate, LLM answer contains name rate, navigation success rate, and navigation accuracy. (**b**) Project-level performance comparison of the same LLM, the accuracy of measurement at the same level. (**c**) Job-level performance comparison of the same LLM, the accuracy of measurement at the same level. All bars represent percentage rates. The legend indicates the corresponding LLM for each color.

During the Project phase, overall performance stabilizes compared to the Application phase, yet the effects of the ontology remain consistently observable. Without ontology, the LLM success rate is 85.4% and the answer-contains-name ratio is 68.2%. While this represents an improvement over the application stage, structural errors persist. Applying ontology increases the LLM success rate to 100% and raises the answer-contains-name ratio to 86.1%. From a navigation perspective, the navigation success rate improves from 66.4% to 88.2%, and the navigation correct rate improves from 48.2% to 69.3%. This suggests that ontology simultaneously enhances navigation stability and accuracy even in real-world service scenarios at the project level, functioning as a structural guide beyond a simple prompt-aiding tool.

The effects of ontology are most pronounced during the task execution phase. In the absence of ontology, the LLM exhibits a high success rate of 94.2%. However, it is noteworthy that only 15.3% of responses included names, and the search success rate was a mere 3.4%. These findings suggest that the model is ineffective in conducting correct searches in large-scale automated environments. Conversely, the application of the ontology has been demonstrated to yield a 99.9% success rate in LLMs, a 91.8% rate of responses containing names, a 99.9% search success rate, and a search accuracy that exhibits a marked increase to 81.5%. This finding indicates that ontologies play a particularly crucial role in large-scale, repetitive environments like job-level, enabling the model to reliably perform accurate structural reasoning and search beyond merely generating responses.

Across a range of evaluations at the application, project and job levels, the integration of ontology has been shown to consistently enhance the accuracy of entity grounding and navigation. The most substantial gains have been observed in large-scale job-level settings.

Additionally, the qualitative comparison of LLM responses when using ontology vs. not using ontology is shown in the following Table 7. We performed Korean morphological analysis using Kiwi [42] via its Python interface, kiwipiepy [43].

Table 7: Comparison of text statistics with and without OntoGPT.

Metric	OntoGPT Used	OntoGPT Not Used	Change Rate
Average number of characters	364.12	350.32	+3.94%
Average number of morphemes	135.04	127.52	+5.90%
Average number of unique morphemes	61.48	58.68	+4.77%
Type–token ratio	0.4772	0.4798	−0.54%
Content word ratio	0.4175	0.4123	+1.24%
Average number of sentences	8.04	8.80	−8.64%
Average morphemes per sentence	18.8684	16.0856	+17.3%

The application of ontology guidance resulted in enhanced information density and structural compactness of the generated text. The average character count exhibited an increase from 350.32 to 364.12 (+3.94%), and the average number of morphemes also rose from 127.52 to 135.04 (+5.90%). While the response length exhibited a slight increase, it concomitantly contained a greater amount of information. With regard to lexical diversity, the mean number of unique morphemes increased from 58.68 to 61.48 (4.77% increase). Concurrently, the mean number of sentences diminished from 8.80 to 8.04 (8.64%), whilst the number of morphemes per sentence augmented from 16.09 to 18.87 (17.3%). This finding suggests that the information was condensed into a smaller number of sentences. While the morpheme type-token ratio remained stable at 0.4772 to 0.4798, the content word ratio exhibited a marginal increase from 0.4123 to 0.4175 (+1.24%), suggesting a slightly higher proportion of semantically significant tokens. The results of the

study demonstrate that ontology-based guidance primarily contributes to enhancing structural efficiency and information density rather than superficial lexical variation.

The detailed analysis table is as follows Table 8. This table details the results of repeated testing conducted on MCP calls related to simulation job execution (create_job), specific application navigation tasks (navigate_application), application list queries (simple_list_app), queries concerning the relationship between applications, projects, and developers (simple_ontology_friendly), and queries about project information (simple_project_info).

Table 8: Per-preset average statistics with and without OntoGPT.

OntoGPT	Avg. Chars	Avg. Morphemes	Avg. Sentences	Avg. Content Ratio
Preset: create_job				
O	304.0	109.0	6.2	0.3968
X	403.2	138.0	11.2	0.3995
Preset: navigate_application				
O	421.8	140.4	11.2	0.4842
X	390.4	136.6	10.0	0.4942
Preset: simple_list_apps				
O	402.2	156.6	7.6	0.4720
X	246.6	89.0	6.4	0.3902
Preset: simple_ontology_friendly				
O	450.0	175.0	10.4	0.4040
X	382.2	151.6	9.0	0.4068
Preset: simple_project_info				
O	242.6	94.2	4.8	0.3303
X	329.2	122.4	7.4	0.3710

Illustrated in Table 8, the mean response statistics for each preset under both the OntoGPT-enabled and OntoGPT-disabled settings are reported, including the character count, morpheme count, sentence count, and content ratio. For the create_job preset, responses generated with OntoGPT exhibit an average length of 304.0 characters, 109.0 morphemes, and 6.2 sentences. Conversely, responses generated without OntoGPT demonstrate higher means of 403.2 characters, 138.0 morphemes, and 11.2 sentences. Despite these differences in verbosity, the content ratio remains comparable, with values of 0.3968 when OntoGPT is enabled and 0.3995 when it is disabled. In the navigate_application preset, the average length of OntoGPT-enabled responses is 421.8 characters and 140.4 morphemes, distributed across 11.2 sentences. In the absence of OntoGPT, the respective means are 390.4 characters, 136.6 morphemes, and 10.0 sentences. The content ratio remains high in both settings, measured at 0.4842 with OntoGPT and 0.4942 without it, indicating that the additional text produced under OntoGPT largely preserves semantic relevance. For the simple_list_apps dataset, OntoGPT generates responses with an average length of 402.2 characters, comprising 156.6 morphemes and 7.6 sentences, with a content ratio of 0.4720. In the absence of OntoGPT, responses demonstrate a reduced length, with an average of 246.6 characters, 89.0 morphemes, and 6.4 sentences. These responses exhibit a lower content ratio of 0.3902, indicating a diminished informational

density. In the simple_ontology_friendly preset, responses generated with OntoGPT reach an average of 450.0 characters, 175.0 morphemes, and 10.4 sentences, while responses generated without OntoGPT average 382.2 characters, 151.6 morphemes, and 9.0 sentences. The content ratio remains stable across both settings, with values of 0.4040 and 0.4068, respectively, indicating that OntoGPT primarily affects response elaboration rather than proportional content density. Finally, for the simple_project_info category, OntoGPT-enabled responses demonstrate enhanced conciseness, exhibiting an average of 242.6 characters, 94.2 morphemes, and 4.8 sentences. In the absence of OntoGPT, the response extends to 329.2 characters, incorporating 122.4 morphemes and 7.4 sentences. The content ratio is measured at 0.3303 in the OntoGPT setting and 0.3710 when OntoGPT is disabled, reflecting a trade-off between brevity and content proportion.

The results indicate that OntoGPT systematically modulates response structure across presets, reducing verbosity in task-oriented and summary-style prompts while expanding and enriching responses in ontology-aligned and listing-oriented scenarios.

Statistical Validation

The statistical evaluation results comparing the default RAG and the ontology-based Gemma7B are presented in Tables 9 and 10.

Table 9: Cochran's Q test results for default RAG and ontology-based Gemma7B.

Schema	*N* (Common Items)	Cochran's Q	*p*-Value	Interpretation
Application	249	120.7143	4.41×10^{-28}	$< \alpha$
Job	11,290	9233.00	≈ 0	$< \alpha$
Project	335	43.3252	4.64×10^{-11}	$< \alpha$

Table 10: McNemar test results results for default RAG and ontology-based Gemma7B.

Schema	McNemar Exact *p*-Value	Interpretation
Application	6.21×10^{-34}	$< \alpha$
Job	≈ 0	$< \alpha$
Project	2.08×10^{-11}	$< \alpha$

As shown in Table 9, Cochran's Q test reveals statistically significant differences across all schemas, including *application*, *job*, and *project* ($p < \alpha$). Notably, extremely small p-values, particularly for the *job* schema ($p \approx 0$), indicate a very strong rejection of the null hypothesis, suggesting that the performance differences between the two approaches are highly significant.

To further investigate these differences, McNemar's test was conducted as a post-hoc analysis, and the results are summarized in Table 10. The results show that all pairwise comparisons are statistically significant ($p < \alpha$), with some p-values approaching zero, again indicating strong evidence against the null hypothesis. These findings demonstrate that the performance differences between the default RAG and the ontology-based Gemma7B are not only globally significant but also consistently significant at the pairwise level. In particular, the magnitude of the test statistics and the extremely small p-values suggest that the observed differences are not due to random variation but reflect a systematic effect introduced by the ontology-enhanced LLM system. Overall, the results confirm that the incorporation of ontology leads to a statistically significant shift in model performance across all evaluated schemas.

5 Conclusion

The present study constructed a Retrieval-Augmented Generation (RAG) system on the basis of an extant High-Performance Computing (HPC) platform, and developed a system to generate diverse Large Language Model (LLM) responses to user queries for data continuously produced on the aforementioned platform. Furthermore, an MCP was utilised to define various API service tools available on the existing HPC platform. This approach enabled the definition of ontology-based data associations for executing service tools in response to LLM result data and actual user requests. In addition, an AI agent system was developed to facilitate the execution of services in response to a wide range of user queries. It is important to note that the HPC-based job execution platform continuously runs simulation tasks and stores their results, thereby creating a structure in which data accumulates over time. This necessitates the processing of user queries based on real-time data. In consideration of the data characteristics outlined above, the RAG system processed user queries on the basis of a vector database linked to the platform. Furthermore, the platform's data structure features were defined as an ontology. The definition of an MCP system was informed by the associations and relational characteristics between data, with the objective of linking service tools according to user queries.

It is evident that contemporary service platforms predominantly offer LLM agent services through RAG systems. Extensive research is currently ongoing to deliver a range of services that are customised to these characteristics. The present study defines data characteristics based on an ontology that specifies data formats and relationships, tailored to the nature of HPC-based simulation services with diverse data. In order to enhance the linkage between the actual service tools and the LLM agents, and to improve the quality of the LLM responses for data analysis, an ontology-enhanced LLM system service was added. This service establishes data relationships between existing RAG and MCP systems, thereby enhancing connectivity and introducing relational definitions. The evaluation process revealed that free LLMs such as Ollama, despite their high quality, demonstrated lower accuracy and connectivity in comparison to ChatGPT-based models. Moreover, for ChatGPT-based models, internal calibration yielded negligible disparities between utilisation and non-utilisation of the ontology. However, in order to utilise the free LLM versions available on the platform, it is necessary that services are based on the model with the highest accuracy, rather than ChatGPT-series models. Consequently, subsequent to the establishment of a RAG system, the incorporation of an additional ontology-enhanced LLM system service to delineate accuracy and the relationality of internal platform data can facilitate a more precise and comprehensive distribution of user responses.

Subsequent research endeavours will persist in investigating the evolving expansion of ontology relationships in conjunction with the escalating scale of HPC simulation operations, in consideration of the prevailing characteristics of the platform. The objective of this study is to develop and enhance a real-time, data-customised RAG system that is tailored to the characteristics of these platforms. Specifically, the objective is to conduct research to extract dynamic data relationships aligned with real-time generated data and enhance the quality of qualitative responses to complex user queries and data relationship queries, based on the form of data expandable by the LLM.

Acknowledgement: Not applicable.

Funding Statement: This research was supported by the Global TOP Strategic Research Group Program of the National Research Council of Science & Technology (No. GTL24031-700).

Author Contributions: The authors confirm contribution to the paper as follows: Conceptualization, Yejin Kwon and Jeongcheol Lee; methodology, Youngbom Park; software, Yejin Kwon and Jeongcheol Lee; validation, Youngbom Park, formal analysis, Yejin Kwon and Youngbom Park; investigation, Yejin Kwon; resources, Yejin Kwon and Jeongcheol Lee; data curation, Yejin Kwon; writing—original draft preparation, Yejin Kwon; writing—review and editing, Yejin Kwon;

visualization, Yejin Kwon; supervision, Youngbom Park; project administration, Youngbom Park; funding acquisition, Jeongcheol Lee. All authors reviewed and approved the final version of the manuscript.

Availability of Data and Materials: Not applicable.

Ethics Approval: Not applicable.

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

References

1. Ma J, Lee J R, Cho K, Park M. Design and implementation of information management tools for the EDISON open platform. KSII Trans Internet Inf Syst. 2017;11(2):1089–104. doi:10.3837/tiis.2017.02.026.
2. Suh YK, Ryu H, Kim H, Cho KW. EDISON: a web-based HPC simulation execution framework for large-scale scientific computing software. In: Proceedings of the 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid). Piscataway, NJ, USA: IEEE; 2016. p. 608–12. doi:10.1109/CCGrid.2016.31.
3. Han S, Lee J, Jeon I, Lee J, Choi H. Data framework design of EDISON 2.0 digital platform for convergence research. KSII Trans Internet Inf Syst. 2023;17(8):2292–313. doi:10.3837/tiis.2023.08.019.
4. Ma J, Park SC, Shin JH, Kim NG, Seo JH, Lee JSR, et al. AI based intelligent system on the EDISON platform. In: Proceedings of the Artificial Intelligence and Cloud Computing Conference. New York, NY, USA: ACM; 2018. p. 106–14. doi:10.1145/3299819.3299843.
5. Torres N, Ulloa C, Araya I, Ayala M, Jara S. A comprehensive analysis of gender, racial, and prompt-induced biases in large language models. Int J Data Sci Anal. 2025;20(4):3797–834. doi:10.1007/s41060-024-00696-6.
6. Park JS, O'Brien J, Cai CJ, Morris MR, Liang P, Bernstein MS. Generative agents: interactive simulacra of human behavior. In: Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology (UIST). New York, NY, USA: ACM; 2023. p. 1–22. doi:10.1145/3586183.3606763.
7. Anh-Hoang D, Tran V, Nguyen LM. Survey and analysis of hallucinations in large language models. Front Artif Intell. 2025;8:1622292. doi:10.3389/frai.2025.1622292.
8. Yao S, Yu D, Zhao J, Shafran I, Griffiths T, Cao Y, et al. Tree of thoughts: deliberate problem solving with large language models. In: Advances in neural information processing systems (NeurIPS). Red Hook, NY, USA: Curran Associates, Inc.; 2023. p. 11809–22.
9. Ahn S, Lee J, Kim J, Lee JSR. EDISON-DATA: a flexible and extensible platform for processing and analysis of computational science data. Softw Pract Exp. 2019;49(10):1509–30. doi:10.1002/spe.2732.
10. Ma J, Seo J. Implementation and application of the EDISON platform's integrated file management service. J Internet Comput Serv. 2016;17(6):71–9. doi:10.7472/jksii.2016.17.6.71.
11. On N, Kim NG, Ru K, Jang H, Lee JR. An analysis of the factors affecting user satisfaction in computational science and engineering platforms: a case study of EDISON. J Internet Comput Serv. 2019;20(6):85–93. doi:10.7472/jksii.2019.20.6.85.
12. Kwon Y, Jeon I, Lee JR, Seo JH. Effectiveness analysis of web based simulation for computational science and engineering on improvement EDISON platform. In: Proceedings of the IEEE International Conference on Smart Cloud (SmartCloud). Piscataway, NJ, USA: IEEE; 2018. p. 27–33. doi:10.1109/SmartCloud.2018.00013.
13. Wang L, Ma C, Feng X, Zhang Z, Yang H, Zhang J, et al. A survey on large language model based autonomous agents. Front Comput Sci. 2024;18(6):186345. doi:10.1007/s11704-024-40231-1.
14. Xi Z, Chen W, Guo X, He W, Ding Y, Hong B, et al. The rise and potential of large language model based agents: a survey. Sci China Inf Sci. 2024;68(2):121101. doi:10.1007/s11432-024-4222-0.
15. Lewis P, Perez E, Piktus A, Petroni F, Karpukhin V, Goyal N, et al. Retrieval-augmented generation for knowledge-intensive NLP tasks. Adv Neural Inf Process Syst. 2020;33:9459–74.
16. Arslan M, Ghanem H, Munawar S, Cruz C. A survey on RAG with LLMs. Procedia Comput Sci. 2024;246:3781–90. doi:10.1016/j.procs.2024.09.178.
17. Asai A, Wu Z, Wang Y, Sil A, Hajishirzi H. Self-RAG: self-reflective retrieval augmented generation. arXiv:2310.11511. 2023.

18. Jiang C, Qi B, Hong X, Fu D, Cheng Y, Meng F, et al. On large language models' hallucination with regard to known facts. In: Proceedings of NAACL-HLT 2024. Kerrville, TX, USA: ACL; 2024. p. 1041–53.
19. Lavrinovics LE, Biswas R, Bjerva J, Hose K. Knowledge graphs, large language models, and hallucinations: an NLP perspective. Web Semant. 2024;85(1):100844. doi:10.1016/j.websem.2024.100844.
20. Yang D, Xiao D, Wei J, Li M, Chen Z, Li K, et al. Improving factuality in large language models via decoding-time hallucinatory and truthful comparators. Proc AAAI Conf Artif Intell. 2025;39(24):25606–14. doi:10.1609/aaai.v39i24.34751.
21. Li M, Kilicoglu H, Xu H, Zhang R. BioMedRAG: a retrieval augmented large language model for biomedicine. J Biomed Inform. 2025;162(1):104769. doi:10.1016/j.jbi.2024.104769.
22. Luo G, Arase Y. MedSummRAG: domain-specific retrieval for medical summarization. In: Proceedings of the 24th Workshop on Biomedical Language Processing. Kerrville, TX, USA: ACL; 2025. p. 27–33.
23. Model Context Protocol. Introduction to model context protocol (MCP) [Internet]. 2024 [cited 2025 Apr 10]. Available from: https://modelcontextprotocol.io/introduction.
24. Hou X, Zhao Y, Wang S, Wang H. Model context protocol (MCP): landscape, security threats, and future research directions. arXiv:2503.23278. 2025.
25. Ray PP. A survey on model context protocol (MCP): architecture, state-of-the-art, challenges and future directions. Authorea Preprints. 2025. doi:10.36227/techrxiv.174495492.22752319/v1.
26. Abdullahi S, Danyaro KU, Chiroma H. The rise of hallucination in large language models: systematic reviews, performance analysis and challenges. Cluster Comput. 2026;29(2):124. doi:10.1007/s10586-025-05891-z.
27. Babaei Giglou H, D'Souza J, Auer S. LLMs4OL: large language models for ontology learning. In: Proceedings of the International Semantic Web Conference (ISWC). Cham, Switzerland: Springer Nature Switzerland; 2023. p. 408–27.
28. Lippolis AS, Saeedizade MJ, Keskisärkkä R, Zuppiroli S, Ceriani M, Gangemi A, et al. Ontology generation using large language models. In: The Semantic Web: 22nd European Semantic Web Conference, ESWC 2025. Cham, Switzerland: Springer Nature Switzerland; 2025. p. 321–41.
29. Guarino N, Welty C. An overview of OntoClean. In: Handbook on ontologies. Cham, Switzerland: Springer; 2009. p. 201–20. doi:10.1007/978-3-540-92673-3.
30. Shen Y, He X, Gao J, Deng L, Mesnil G. Learning semantic representations using convolutional neural networks for web search. In: Proceedings of the 23rd International Conference on World Wide Web. New York, NY, USA: ACM; 2014. p. 373–4. doi:10.1145/2567948.2577348.
31. Agrawal G, Kumarage T, Alghamdi Z, Liu H. Can knowledge graphs reduce hallucinations in llms?: A survey. In: Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. Kerrville, TX, USA: ACL; 2024. p. 3947–60. doi:10.18653/v1/2024.naacl-long.219.
32. DeBellis M, Dutta N, Gino J, Balaji A. Integrating ontologies and large language models to implement retrieval augmented generation. Appl Ontol. 2024;9(4):1–19. doi:10.1177/15705838241296446.
33. Feng H, Yin Y, Reynares E, Nanavati J. OntologyRAG: better and faster biomedical code mapping with retrieval-augmented generation (RAG) leveraging ontology knowledge graphs and large language models. In: International Workshop on Knowledge-Enhanced Information Retrieval. Cham, Switzerland: Springer Nature; 2025. p. 71–86.
34. Caufield JH, Hegde H, Emonet V, Harris NL, Joachimiak MP, Matentzoglu N, et al. Structured prompt interrogation and recursive extraction of semantics (SPIRES): a method for populating knowledge bases using zero-shot learning. Bioinformatics. 2024;40(3):btae104. doi:10.1093/bioinformatics/btae104.
35. Aslam M. Cochran's Q test for analyzing categorical data under uncertainty. J Big Data. 2023;10(1):147. doi:10.1186/s40537-023-00823-3.
36. Agresti A. Categorical data analysis. 2nd ed. Hoboken, NJ, USA: Wiley; 2002.
37. Pembury Smith MQR, Ruxton GD. Effective use of the McNemar test. Behav Ecol Sociobiol. 2020;74(11):133. doi:10.1007/s00265-020-02916-y.
38. Chang Y, Wang X, Wang J, Wu Y, Yang L, Zhu K, et al. A survey on evaluation of large language models. ACM Trans Intell Syst Technol. 2024;15(3):39. doi:10.1145/3641289.

39. Google. Gemma: open models based on Gemini research. 2024 [cited 2026 Apr 10]. Available from: https://ai.google.dev/gemma.
40. Alibaba Group. Qwen3 technical report. 2024 [cited 2026 Apr 10]. Available from: https://github.com/QwenLM/Qwen.
41. Kasneci E, Sessler K, Küchemann S, Bannert M, Dementieva D, Fischer F, et al. ChatGPT for good? On opportunities and challenges of large language models for education. Learn Individ Differ. 2023;103(1):102274. doi:10.1016/j.lindif.2023.102274.
42. Lee S, Joo K, Park D, Kim S, Kim S. Artificial intelligence-based prediction modeling of audit dispositions in Korean public institutions. In: Proceedings of the 8th International Conference on Natural Language Processing and Information Retrieval. New York, NY, USA: ACM; 2024. p. 406–12. doi:10.1145/3711542.3711563.
43. Kang J. kiwipiepy: python wrapper for the Kiwi Korean morphological analyzer. 2020 [cited 2026 Apr 10]. Available from: https://github.com/bab2min/kiwipiepy.