Research Article

Applied Ontology
2025, Vol. 20(1) 16–35
© The Author(s) 2025
Article reuse guidelines:
sagepub.com/journals-permissions
DOI: 10.1177/15705838251329266
journals.sagepub.com/home/apo

OI-LLM: A Scalable Framework to Integrate Large Scale Ontologies Using Large Language Model

Sujata Pardeshi[1], Virat Giri[2] and Sushopti Gawade[3]

Abstract
Ontology integration plays a vital role in forming a unified knowledge base through existing knowledge. The integration process is triggered through ontology matching and ontology merging processes. The scale of ontologies dominates the ontology matching process. Earlier researchers have used the Ontology (Meta) Matching (OMM) technique to generate an efficient set of ontology alignments. The set of ontology alignments is used to match two ontologies. This process involves limitations such as the help of domain experts in choosing applicable similarity measures, opting for appropriate and applicable resources (such as WordNet, UMLS, etc.) to generate efficient semantic similarity, and requiring more computations to generate alignments. This raises the issue of increased computational complexity. To resolve this problem, researchers have employed a divide-and-conquer algorithmic strategy for executing multiple matching tasks concurrently with parallel processing. There is still a need to improve computational complexity while handling the integration of large-scale ontologies. With such needs, this article proposes a general and scalable framework to integrate large-scale ontologies. The proposed algorithm uses nature-inspired computing algorithms to apply the natural behavior of the ant-colony optimization algorithm to balance and optimize the working behavior of the machines during parallel processing. The LLM is used to generate highly efficient similarity results during the matching process, which results in a cohesive integrated ontology. The OAEI Anatomy track is used to test the performance of the framework to assess the computational time. The experimental results show that there is an improvement in computational time. It is possible to achieve the principles of ontology coherence and entity coverage.

Keywords
ontology integration, ontology matching, ontology (meta) matching, ontology merge, large language model

Received: 3 October 2024; accepted: 27 February 2025

1 Introduction

The semantic web and ontology community (Machado et al., 2020; Stoilos et al., 2018) looks forward to enriching ontologies to share a knowledge base (He et al., 2023a, 2023b; Norouzi et al., 2023) across intelligent systems. It will be useful to recommend the required data to make better decisions. The biomedical ontologies, Human Phenotype Ontology (HP) (Köhler et al., 2017), Mammalian Phenotype Ontology (MP) (Cynthia et al.), and Human Disease Ontology (DOID) (Lynn Schriml et al., 2003), were provided and used for biomedical research purpose. HP provides vocabulary used for computational analysis of the human phenome and for diagnosing monogenic diseases. The DOID contains classification of human disease terminologies related to biomedical resources, human health, neurological disease, and neurological disorders.

[1]Computer Science and Engineering, Sanjay Ghodawat University, Kolhapur, Maharashtra, India
[2]Computer Science and Engineering, Sanjay Ghodawat Institute, Kolhapur, Maharashtra, India
[3]Information Technology, Vidyalankar Institute of Technology, University of Mumbai, Maharashtra, India

Corresponding Author:
Sujata Pardeshi, Computer Science and Engineering, Sanjay Ghodawat University, Kolhapur, Maharashtra, India.
Email: sujatapardeshi252@gmail.com

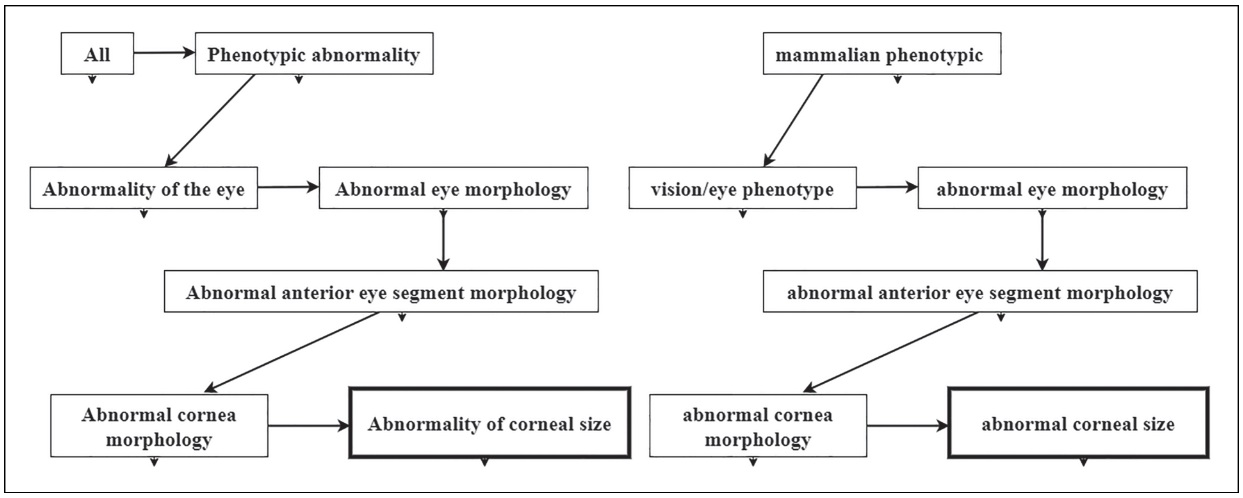
All
Phenotypic abnormality
Abnormality of the eye
Abnormal eye morphology
Abnormal anterior eye segment morphology
Abnormal cornea morphology
Abnormality of corneal size
mammalian phenotypic
vision/eye phenotype
abnormal eye morphology
abnormal anterior eye segment morphology
abnormal cornea morphology
abnormal corneal size

Figure 1. Semantic Heterogeneity Issue of HP Ontology and MP Ontology.

MP provides information about mammalian organisms that are used to demonstrate morphological, physiological, and behavioral changes in mammals during testing.

Various ontology engineers have designed biomedical ontologies for related domains that generate heterogeneous ontologies (Xue et al., 2020). Biomedical ontologies involve syntactic and semantic heterogeneity. In HP, most class labels are written in capital letters, but in MP, all class labels are written in small case letters. Consequently, such syntactic heterogeneity occurs when similar terminologies are used but written in different cases. Figure 1 also shows the presence of semantic heterogeneity in the HP and MP ontologies. In HP, the class name is “Abnormality of corneal size,” and with a similar meaning, the MP ontology defines that class as “abnormal cornea size.” It is essential to address such data heterogeneity issues to ensure the robust working of an intelligent system.

These biomedical ontologies are also used by researchers during their research. Researchers in the biomedical domain want to investigate medicines for hearing abnormalities on the basis of genetic data as well as patient disease data. In this case, researchers cannot use either the HP or DOID ontology to make better decisions during their research work. However, the HP ontology provides only genetic information, and the DOID ontology provides only disease data. A new ontology must be provided by integrating the knowledge bases of both ontologies to make better decisions.

The ontology integration process either forms a new integrated ontology or enriches the existing ontology. The integration process depends on ontology matching and ontology merging processes. The integration of ontologies involves merging similar entities by following the structural relationships of input ontologies. By considering earlier research, the Entity-to-Entity Ontology Model (EEOM) (Ocker et al., 2022; Xue et al., 2021a, 2021b, 2021c) and the Graph-Based Ontology Model (GPOM) (Ochieng and Kyanda 2018; Suleiman et al., 2024) approaches are used to integrate the ontologies. In the EEOM approach, ontology alignments are used to identify similar entities via different types of mapping rules. These similar entities are merged to form the integrated ontology incrementally. The GPOM approach represents candidate ontologies as graph data structures. The traversing algorithms, such as breadth first search or depth first search (DFS), are used to traverse the graphs. Different graph theory algorithms are used to determine a set of subgraphs that cover the maximum number of nodes. Afterward, the subgraphs are taken one by one to identify similar entities via a set of alignments or without the use of alignments. Once similar entities are found, the merging process initiates and continues to cover all the subgraphs. In this way, an integrated ontology is formed.

Either the EEOM or the GPOM approach is used; the ontologies are integrated into three categories. In the first category, defined as OIT_1, an integrated ontology is formed via ontology alignment (Osman et al., 2021a, 2021b; Portisch et al., 2022; Xue et al., 2021a, 2021b, 2021c; Zhaoming & Rong, 2022; Zhou et al., 2023; Zhu et al., 2022). The ontology matching process is used to generate ontology alignments. The resulting set of alignments is used to integrate the ontologies. The second category, defined as OIT_2, integrates ontologies by using reference alignment provided by external resources. Some researchers have also used the support of human domain experts to provide information about reference alignment (Geng, 2023; Norouzi et al., 2023; Ochieng & Kyanda, 2018; Portisch et al., 2022; Xue et al., 2021a, 2021b, 2021c). In the third category, defined as OIT3, an integrated ontology is formed without the use of ontology alignments. In this category, the matching process is used to identify similar entities. These similar entities are merged to form the integrated ontology.

OIT_1 comprises a more exhaustive alignment process that requires more time and memory. To match and merge two ontologies with "n" number of entities may have $T(n) = O(n)\ X\ O(n)\ + O(P)$ time complexity. The OIT_2, OIT_3 categories avoid the alignment process and require less time. However, the OIT_2 approach is not applicable to match large-scale ontologies. This indicates that the ontology integration process is dominated by the scale of the participating ontologies during the matching process. It is critical to match large-scale ontologies (He et al., 2023a, 2023b; Xue et al., 2023); however, these ontologies contain more heterogeneous entities (Osman et al., 2021a, 2021b). The scale of ontologies continues to increase, and accordingly, the computation time also increases.

Earlier research has yielded remarkable results in the use of the Ontology (Meta) Matching (OMM) technique while matching heterogeneous ontologies. Section 3 describes the OMM technique in detail. The OMM technique may consume approximately 2/3 of the computational time while matching large-scale ontologies. To address the computational time issue, this work is proposed under the OIT_3 category. The proposed system first divides the large-scale ontologies into small-scale ontologies via superclass and subclass structural relations. The divide strategy balances the heavy load of the matching process by harnessing the power of parallel computing. The load balancing process is optimized via the ant colony optimization (ACO) algorithm. The matching process uses the large language model (LLM) to avoid the lengthy OMM process. Recently, LLM has also been used to match large-scale ontologies efficiently (He et al., 2023a; Zhang et al., 2024). The merge process runs in the background to merge similar kinds of entities using the similarity score returned by the LLM. Accordingly, this work proposes the design and development of a generic framework for integrating large-scale ontologies using LLM.

This article has different sections. Section 2 specifies related work to identify limitations for integrating ontologies. This section also reviews the existing methodologies that have used LLM to match large-scale ontologies with significant results. Section 3 provides an overview and challenges of the OMM approach. Section 4 presents the design of the proposed model. Section 5 presents the working methodology and proposed algorithms for implementation of the proposed model. Section 6 presents an evaluation of the proposed framework. Section 7 presents concluding remarks about the purpose of the research article.

2 Related Work

2.1 Integration of Ontologies and Related Work

Babalou and Konig-Ries (2020) provide a framework, CoMerger, which uses the structural similarity-based partitioning technique to divide large-scale ontologies into "n" blocks. The classes that are closest to each other in terms of structural similarity are retained in one block. In this way, the "n" number of blocks is formed to keep relevant and closer classes. Afterward, a direct merging process is initiated to merge these blocks to form a merged ontology. This work does not use similarity measures or functions, hence reducing the computational time. The authors have also suggested to take the benefit of parallel processing.

The authors (Patel & Jain, 2019) addressed computational time issues while integrating large-scale ontologies using a divide-and-conquer greedy algorithmic strategy and parallel processing. The input ontologies are treated as source ontologies and target ontologies on the basis of the degree of coupling and cohesion. The ontology that has maximum coupling and minimum cohesion among the concepts is taken as the source ontology. The other one is treated as the target ontology. The Lin measure agglomerative algorithm is applied to partition the source ontology into small clusters of concepts. These clusters are used to partition the target ontology into clusters. The cluster matching and matrix aggregation algorithms are applied to match and integrate the source and target ontologies via parallel computing. The pool of ontology matchers is used to generate efficient alignments. Ultimately, this work reduces the computational time, but it has to follow a critical process of the OMM. Future work should apply a workload balancing strategy to efficiently utilize the "n" number of processors involved in parallel processing.

Li et al. (2020) provided an ontology matching process using filters and verification phases. During the filtering phase, typical entities that have high similarity scores are identified using syntactic and semantic similarity functions. Afterward, such typical entities are used to divide the large-scale ontologies into subontologies. The clustering approach is applied to form clusters or blocks of such typical entities. These blocks are again partitioned to form pairs of blocks. The pair of blocks is used to form pairs of subontologies via extension methods. The verification phase matches the relevant subontologies. During the filtering phase, the irrelevant entities are excluded before matching, which reduces the computational time. The partitioning process changes the structure of the original ontologies so that subontologies are not matched properly, which results in a low recall rate. Babalou and Konig-Ries (2020), and Patel and Jain (2019) employed parallel computing, but there is still a need to apply and optimize a load balancing strategy during parallel computing. The proposed framework fulfills this shortcoming.

The authors (Xue et al., 2023) specified that, for integrating large-scale ontologies, the running time is greater since thousands of classes are matched with other classes. The nondominated sorting genetic algorithm (NSGA-II) is used to resolve the computational time. It integrates knowledge of different large-scale economics and finance domain ontologies to resolve heterogeneity issues. This work only evaluates the efficiency of the matching process and does not consider the evaluation of integrated ontologies to measure the degree of consistency. The authors (Huang et al., 2022) used a multiobjective OMM technique to find an optimized set of ontology alignments by collaborating with the particle swarm intelligence algorithm. The framework achieves better performance for small-scale ontologies. While matching large-scale ontologies, the framework fails to reach local optima. The diversity-enhancing strategy is used to solve the issue of local optima. This work reviews guidelines to avoid premature convergence while applying a population-based nature-inspired algorithm. Huang et al. (2020), Lu and Xue (2020), Xue et al. (2021a, 2021b, 2021c), and Zhou et al. (2023) have matched large-scale ontologies using nature-inspired computing algorithms. These algorithms yield the best results in handling a large search space but do not consider the evaluation of computational time.

Suleiman et al. (2024) merged RDF graph ontologies iteratively in memory to form a new ontology. To identify similarity scores, different tools of natural language processing (NLP), such as BERT, SM-DTR, fuzzy string matching algorithms, and the WordNet and Word2Vec databases, have been used. This work efficiently matches and updates the nodes of the RDF graph ontology in memory during the merging process. Hence, the usage of memory is optimized. This work efficiently merges middle-scale ontologies with minimum computational time. However, there is a need to assess the required computational time while integrating large-scale RDF graph ontologies.

Xue and Zhang (2021) used the Unified Medical Language System (UMLS) to generate efficient semantic similarity, as WordNet is not suitable for biomedical ontologies. A similar type of issue is also observed in Xue et al. (2018, 2020), where the authors used UMLS. However, this indicates the limitations of the OMM technique related to the use of the resources used to generate efficient semantic matching. This issue is resolved with the help of a LLM in the proposed system.

Osman et al. (2021b) provided a holistic approach for integrating large-scale ontologies via LogMap. It is a benchmark matching tool that provides reference alignments. This approach integrates multiple ontologies via simple merging and full merging at a time that avoids pairwise merging of two concepts incrementally. This minimizes the computational time but is not suitable for real-time environments. The authors (Ocker et al., 2022) provided a semiautomatic framework to integrate small-scale production domain ontologies. It focuses on validating the syntactic, semantic, and structural inconsistency among the ontologies to be integrated. Overall, this work highlights the need to provide a general framework that integrates large-scale ontologies not only for the production domain but also for any other domain. He et al. (2023b) developed a novel framework for integrating small-scale energy storage system ontologies. This work has addressed issues such as incoherence, ambiguity, and redundancy of integrated ontologies that are created for a variety of applications. Semantic-based ontology matching is used to identify a set of correspondences. The semantic heterogeneity is resolved via a pretrained and well-established universal sentence encoder (USE) language model. During the matching process, candidate entities are converted into embedded vectors using the USE and compared using cosine similarity to identify the semantic similarities. The authors also compared the manual efforts required by developers to align the ontologies. This indicates that the manual alignment process is a hectic and time-consuming task. Hence, it is essential to prohibit user interventions during the ontology integration process. Hnatkowska et al. (2020) provided attribute-based integration of ontologies. This model relies on human experts to identify semantic similarities with the help of the WordNet database. This approach helps in the generation of accurate semantic matching. Moreover, user interventions resulted in an increase in computational time.

2.2 Large Language Model and Related Work

He et al. (2023a) used LLM to test zero-shot performance to match large-scale ontologies of the OAEI Bio-ML track. The well-structured CIT-DOID and SNOMED-FMA ontologies are used to test the performance of the LLM. The framework works in two rounds to perform concept and structural-level matching of ontologies. It provides input as a prompt to ChatGPT (GP-4). The first round uses a binary classification method to identify overlapping concepts. It returns a value of “1” for overlapping concepts; otherwise, it returns a “0” value. In a second round, the framework is asked to identify hierarchical parent–child relationships among the concepts through prompts. In this way, a critical ontology matching task is performed very quickly with the help of LLM. Compared with BERTMap, the framework with LLM achieves outstanding performance. The authors suggested first designing a framework, and accordingly, designing a prompt to generate efficient matching results is needed. The working of model is reviewed to design an algorithm to generate efficient calls to the LLM according to the design of a proposed framework.

Earlier work employed ChatGPT-4 to match ontologies using LLM. This approach will generate error-prone responses while matching large-scale ontologies; however, some LLMs can handle limited amounts of data. To address this issue, Giglou et al. (2024) provide a framework, LLMs4OM that uses a dual module strategy defined as the retrieval phase and matching phase. During the retrieval phase, the user uses either queries or prompts to the LLM to retrieve similar entities. Matching is subsequently performed by escalating the power of the LLM on the basis of the prompt. In this work, zero-shot prompts were used to accomplish the matching task efficiently without the use of trained models. The authors perform a zero-shot performance comparison with seven (7) LLMs. Therefore, authors have provided an approach to choose a specific LLM to integrate large-scale ontologies of any domain. The proposed framework uses the work done for the matching phase using LLM of this article.

Zhang et al. (2024) developed a multiagent dialog model that cognitively assists the framework through LLM. It provides the reference alignment to initiate the ontology matching process without intervention by the domain expert. Thus, it helps to minimize decisions about the formation of reference alignment through human experts. Hertling and Paulheim (2023) used LLM to perform large-scale ontology matching. This work uses the NLP approach to identify overlapped entities and generate ontology alignments. The matching is initiated via zero-shot and few-shot prompts to the LLM. This model uses ontology alignment, which results in the computational time required to complete the ontology matching process.

3 Ontology (Meta) Matching Technique and its Challenges

The OMM technique uses similarity measures to identify heterogeneous entities. The similarity measures are also called ontology matchers. Appendix A specifies different similarity measures through Tables A1 to A5. The syntactic similarity measure is also called terminological similarity. It is used to determine the morphological similarity between entities. The semantic measure is also called a linguistic measure. It considers the synonym or hypernym relationship between two entities (Xue et al., 2021a) and calculates the semantic distance between two entities. The structural similarity measure is also called a context similarity measure. In this measure, the similarity between entities is calculated by comparing the structural information of two entities. Figure 2 shows the generic steps used in the OMM technique mentioned by Xue et al. (2021a).

OMM is defined as a mathematical function $A = f(O1, O2, RA, P, R)$, where O_1 and O_2 are input ontologies, RA is a reference alignment, P is a parameter, and R is a resource. This function returns the actual alignment, A, which is a set of ontology alignments and the similarity score of each alignment. The similarity score between two entities is calculated for each similarity measure. It is stored in the respective weighted similarity matrices. In this way, suppose that "n" number similarity measures will be used and that many weighted similarity matrices will be formed to store similarity scores. This approach leads to the consumption of more time and memory (Xue & Zhang, 2021; Xue et al., 2021a). A weighted sum aggregation method is used to combine the similarity results generated by each similarity measure. Although one cannot depend on the use of any one similarity measure, entities may be syntactically, semantically, or contextually different. Therefore, it is necessary to calculate average weighted similarity scores using similarity weights of each similarity measure to generate ontology alignments. The average weight is calculated using the mathematical formulas mentioned in Appendix A, Table A5. This equation involves setting the weight value, for example, $W_1, W_2 \ldots W_n$, which should be between 0 and 1 to the "n" number of similarity measures. It is difficult to set an appropriate weight for each similarity measure to calculate an efficient final similarity score. Afterward, the average weighted similarity matrix is used to choose candidate correspondences that satisfy the threshold value. The threshold value helps to disqualify correspondences whose similarity score is below the threshold value. In this way, a set of qualified alignments is generated and used to perform mapping between source and target ontologies to resolve heterogeneity issues.

The recall, precision, and f-score metrics of the TF-IDF technique are used to assess the quality of the alignments. The f-score value is optimized via either a single-objective or multiobjective optimization function. In such cases, the ontology matching function is called the single objective (Huang et al., 2022; Verhodubs, 2020; Xingsi & Pan, 2018) or multiobjective (Zhou et al., 2023) OMM problem. The optimized f-score is used to choose highly capable alignments to identify similarity among the ontologies. In this way, OMM operations are accomplished, and then the merging process is initiated to integrate the ontologies.

The OMM technique faces certain challenges while handling the integration of large-scale ontologies. These are stated as follows:

1. It is critical to choose appropriate mathematical equations according to the required syntactic, semantic, and structural similarity measures. Most of the time, it will choose either by means of expert suggestions or on the basis

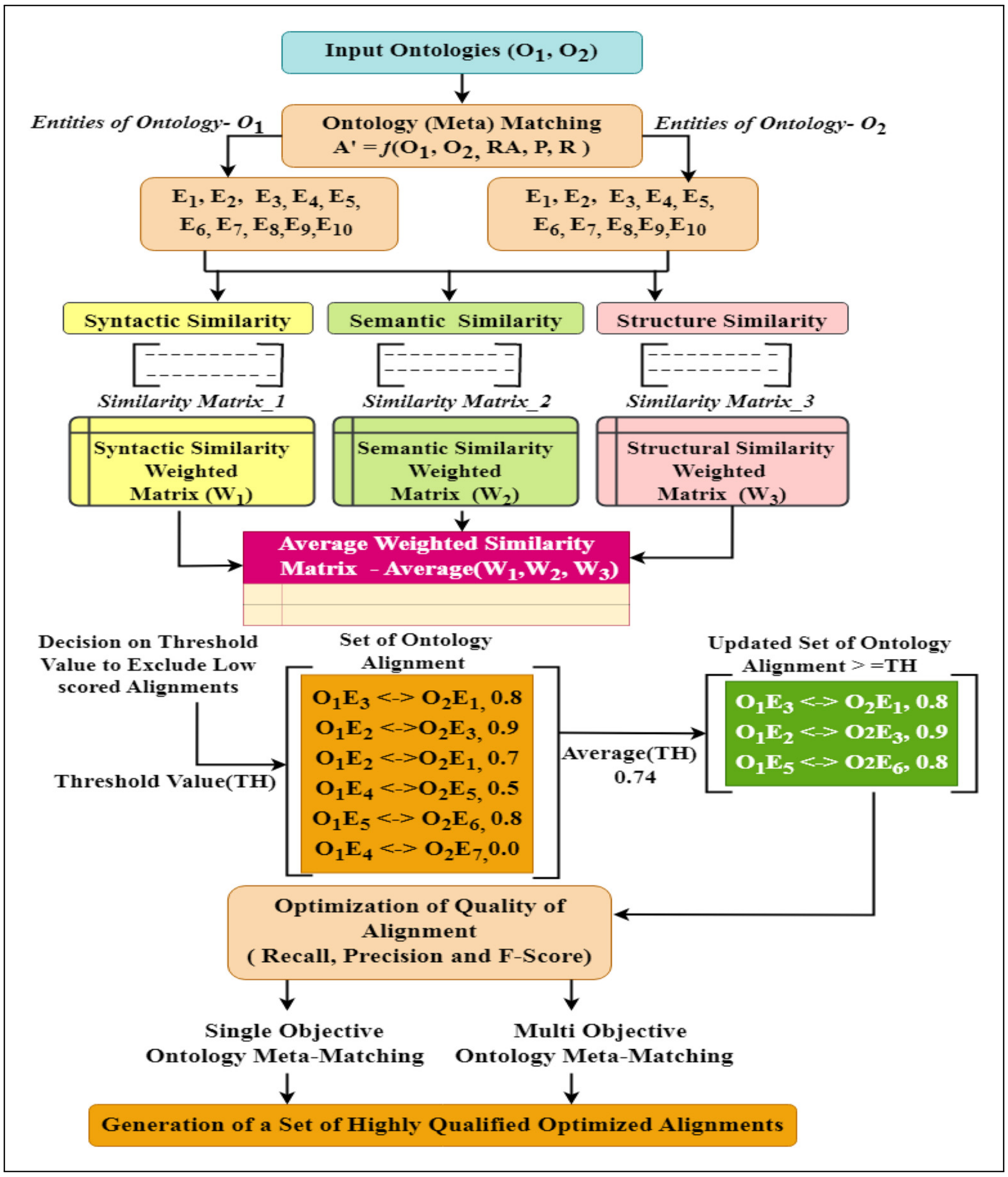
Input Ontologies (O_1, O_2)
Entities of Ontology- O_1
Ontology (Meta) Matching
A' = f(O_1, O_2, RA, P, R)
Entities of Ontology- O_2
E_1, E_2, E_3, E_4, E_5, E_6, E_7, E_8, E_9, E_{10}
E_1, E_2, E_3, E_4, E_5, E_6, E_7, E_8, E_9, E_{10}
Syntactic Similarity
Semantic Similarity
Structure Similarity
Similarity Matrix_1
Similarity Matrix_2
Similarity Matrix_3
Syntactic Similarity Weighted Matrix (W_1)
Semantic Similarity Weighted Matrix (W_2)
Structural Similarity Weighted Matrix (W_3)
Average Weighted Similarity Matrix - Average(W_1,W_2, W_3)
Decision on Threshold Value to Exclude Low scored Alignments
Threshold Value(TH)
Set of Ontology Alignment
O_1E_3 <-> O_2E_1, 0.8
O_1E_2 <->O_2E_3, 0.9
O_1E_2 <->O_2E_1, 0.7
O_1E_4 <->O_2E_5, 0.5
O_1E_5 <-> O_2E_6, 0.8
O_1E_4 <-> O_2E_7,0.0
Average(TH) 0.74
Updated Set of Ontology Alignment >=TH
O_1E_3 <-> O_2E_1, 0.8
O_1E_2 <-> O2E_3, 0.9
O_1E_5 <-> O2E_6, 0.8
Optimization of Quality of Alignment (Recall, Precision and F-Score)
Single Objective Ontology Meta-Matching
Multi Objective Ontology Meta-Matching
Generation of a Set of Highly Qualified Optimized Alignments

Figure 2. Ontology (Meta) Matching Approach.

of past experimental experience. Appendix A, Table A5 presents variations in the equation used to numerate the weighted sum similarity measure.

2. The computational time is greater when storing and computing the data of each similarity matrix.
3. The provision of domain experts to provide the reference alignment is a critical and time-consuming task when dealing with large-scale ontologies.
4. In the OMM technique, the majority of studies use the WordNet Electronics Database to identify the semantic similarity. This database might not be suitable for some domains, such as the biomedical domain (Xue & Zhang, 2021; Xue et al., 2018, 2020).
5. There is no hard or fast rule for setting the threshold value. Researchers either set it with the help of human domain experts or set it by choosing the maximum similarity score or average similarity score (Xue et al., 2021a, 2021b, 2021c).

Table 1. Bucket List of Parameters for Ontology Matching.

Param#	Parameter Type	Parameter Value
OMTE_1	Ontology Representation	RDF Graph Based Representation
OMTE_2	Format of Ontology	Web Ontology Language (OWL)
OMTE_3	Type of Matching	Straight Forward Matching using RDF Graph Model
OMTE_4	Matching Elements	Class Name, Properties of the Class, Comments of Class.
OMTE_5	Type of Correspondence	Simple Type of Correspondence (1:1)
OMTE_6	Use of Set of Predefined (Golden/Reference) Alignments	No
OMTE_7	Types of Relations	Equivalence Relation, Subsumption Relation

Table 2. Bucket List of Parameters for Ontology Integration.

Param#	Parameter Type	Parameter Value
ONITE_1	Ontology Integration Framework	General
ONITE_2	Number of Input Ontologies.	Primary Step: Two Ontologies
ONITE_3	Scale of Ontologies	Large-Scale Ontologies of Similar Domain
ONITE_4	Integration Type	Asymmetric Merge
ONITE_5	Purpose of Integration	Enrichment of Ontology
ONITE_6	Output Ontology	Generation of Consistent Integrated Ontology.
ONITE_7	Preservation of Knowledge of Source Ontology	Compulsory

Table 3. List of Parameters for Evaluating the Integrated Ontology.

No.	List of Parameters	Value of Parameter
EVITO_1	Human Expert Involvement	Yes/No
EVITO_2	Scalability	Large Scale
EVITO_3	Computational Time	HR: MIN : SEC
EVITO_4	Prohibition of Structural and Relational Redundancy	Yes/No
EVITO_5	Knowledge Preservation of Source Ontology	Yes/No
EVITO_6	Consistent Integrated Ontology: Evaluation of Semantic Consistency	Yes/No
EVITO_7	Redundant Entities	Numerical Value
EVITO_7	Number of Unconnected Entities	Numerical Value
EVITO_8	Principle of Ontology Coherence	Yes/No
EVITO_9	Coverage of Entities : Number of Entities Preserved from Input Ontologies	Numerical Value

4 Design of the Proposed Research Work

This phase designs the proposed framework to set the baseline for examining the working of the proposed framework. It has two steps: the preliminary setup of the proposed framework and the design of the mathematical model.

4.1 Preliminary Setup of the Proposed Framework

The lists of parameters and their values are defined with reference to earlier work (Babalou & Konig-Ries, 2019; Negi & Malik, 2018; Osman et al., 2021a). Tables 1 and 2 list the bucket list of parameters used to trigger the ontology matching and integration processes, respectively. Table 3 lists the applicable list of parameters for evaluating the integrated ontology.

4.2 Mathematical Model of the Proposed Framework

Figure 3 shows the mathematical model of the proposed framework. It is represented as a theoretical machine, $M = (Q, \sum, q_0\ F, \delta)$, where $Q = q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8, q_9$ is a set of states and $\sum$ is a finite set of input data. q_0 is the start state of a machine, F is the final state of a machine, and the delta transition function δ: $Q \times \sum \rightarrow Q$. The delta ($\delta$) transition function shows the working of a machine on the basis of input parameters. Table 4 describes the operation of the machine via the delta transition function.

4.2.1 General Workflow of the Theoretical Model.

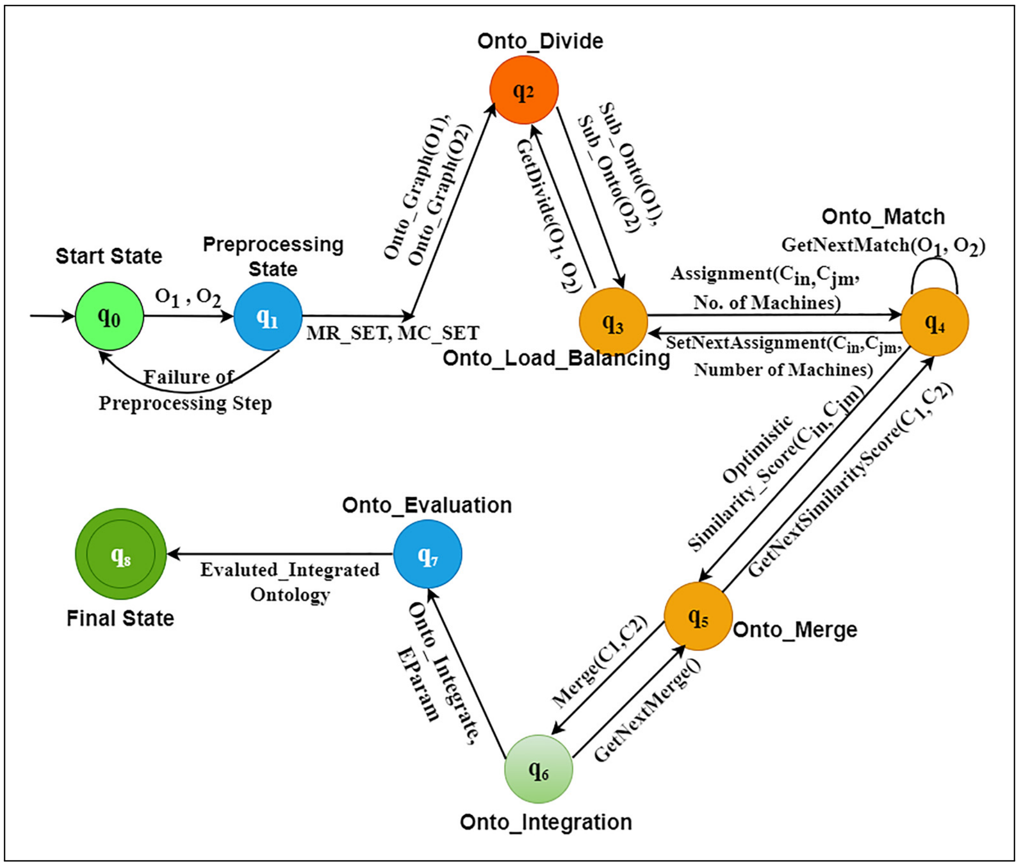
Onto_Divide
q2
Sub_Onto(O1), Sub_Onto(O2)
GetDivide(O1, O2)
Onto_Graph(O1), Onto_Graph(O2)
Start State
Preprocessing State
q0
O1 , O2
q1
MR_SET, MC_SET
Failure of Preprocessing Step
q3
Onto_Load_Balancing
Onto_Match
GetNextMatch(O1, O2)
Assignment(Cin,Cjm, No. of Machines)
SetNextAssignment(Cin,Cjm, Number of Machines)
q4
Optimistic Similarity_Score(Cin,Cjm)
GetNextSimilarityScore(C1,C2)
Onto_Evaluation
q8
Evaluted_Integrated Ontology
q7
Final State
Onto_Integrate, EParam
q5
Onto_Merge
Merge(C1,C2)
GetNextMerge()
q6
Onto_Integration

Figure 3. Theoretical Model of the Proposed Framework.

Table 4. List of States of the Theoretical Model and the Delta Transition Function.

State Name	Delta Transition Function—δ: $Q \times \sum \rightarrow Q$
Start State ($q_{0)}$	$\delta(q_0, \{ O_1, O_2 \}) \rightarrow q_1$
Preprocessing State (q_1)	$\delta(q_1, \{MR\_SET, MC\_SET \},\{Onto\_Graph(O_1), Onto\_Graph(O_2)) \rightarrow q_2$
Onto_Divide ($q_{2)}$	$\delta(q_2, \{ Sub\_Onto(O1), Sub\_Onto\ (O2)\}) \rightarrow q_3$
Onto_Load_Balancing (q_3)	$\delta(q3, \{ Cin, Cjm, Number\ of\ Machines\}) \rightarrow q_4$
Onto_Match (q_4)	$\delta(q4, \{ GetNextMatch(O_1,O_2)\}) \rightarrow q_4$
	$\delta(q4, \{ Optimistic\_Similarity\_Score(C_n, C_m)\}) \rightarrow q_5$ $\delta(q4, \{Cin, Cjm, Number\ of\ Machines\}) \rightarrow q_3$
Onto_Merge($q_{5)}$	$\delta(q6, \{ Merge(C_n, C_m)\}) \rightarrow q_6$
	$\delta(q6, \{ GetNextSimilarityScore(C_n, C_m)\}) \rightarrow q4$
Onto_Integration($q_{6)}$	$\delta(q6, \{ Onto\_Integrate, Evaluation\ Parameters\}_{)} \rightarrow q_7$
Onto_Evaluation(q_7)	$\delta(q7, Consistent\ Integrated\ Ontology) \rightarrow q_8$
Final State (q_8)	Formation of Integrated Ontology.

4.2.1.1 Phase I. The theoretical model starts working at the start state q_0, reads input ontologies O_1 and O_2 and moves to state q_1. The state q_1 preprocesses the input ontologies. The sets of preliminary parameters MC_SET and MR_SET are formed for the ontology matching and merging processes, respectively. The input ontologies are converted into RDF graph ontologies and stored as Onto_Graph (O_1) and Onto_Graph (O_2). Afterwards, preliminary analysis of RDF graph ontologies is performed to identify the number of superclasses and subclasses of all the classes. Then, the machine moves to state q_2 with MC_SET, MR_SET Onto_Graph (O_1), and Onto_Graph (O_2). At a state q_2, Onto_Graph (O_1) and Onto_Graph (O_2) are divided into small-scale ontologies using superclass and subclass structural relations to minimize the large search space. This state forms the subontologies Sub_Onto (O_1) and Sub_Onto (O_2) and is used for further processing.

4.2.1.2 Phase II. At a state q_3, the load balancing strategy is applied through parallel processing to distribute the heavy matching load among the machines. At this stage, a function, Assignment (C_{in}, C_{jm}, Number of Machines), is initiated to assign C_{in}, C_{jm} classes of Sub_Onto(O_1) and Sub_Onto(O_2) to the "*n*" number of machines. This helps to reduce the computational time. The state q_4 called Onto_Match performs matching among C_{in} and C_{jm} in a loop until all classes of Sub_Onto(O_2) are matched with Sub_Onto (O_1). If there is similarity among classes C_{in}, C_{jm} is found, then state q_4 sends the Optimistic_Similarity score to state q_5. Otherwise, state q_4 is repeated through the calling of a function GetNextClass() to match the next classes. To match these classes, state q_4 sends requests via SetNextAssignment (C_{in}, C_{jm}, number of machines) to state q_3. In this way, the state q_3 runs in a background to perform parallel processing. The Optimistic_Similarity score for each matching step is stored and referred to at state q_5.

4.2.1.3 Phase III. At state q_5, a merge operation is performed to merge the overlapping classes. Simultaneously, at state q_6, the integration process starts to integrate the overlapped classes. The integrated ontology and list of evaluation parameters are sent as inputs to a state q_6. This state is responsible for evaluating the integrated ontology via a list of evaluation parameters. The machine moved to q_7 to evaluate the integrated ontology. In this way, the machine enters the final state q_8 and provides an evaluated integrated ontology.

5 Proposed Framework Architecture

5.1 Working Methodology and Algorithms of the Proposed Framework

Figure 4 shows the proposed system architecture diagram. The proposed framework is implemented by considering the general workflow of the mathematical model mentioned in Section 4.2.1. The working methodology has three (3) stages. Stage I is the preprocessing stage, Stage II is ontology matching and load balancing, and Stage III is the ontology integration stage.

Algorithm_1 mentions the essential steps to be followed by the proposed framework to generate the integrated ontology. The preprocessing stage starts at Step No. 1 and ends at Step No. 8 of Algorithm_1. This module establishes the preliminary setup of the ontology matching process in correlation with Section 4.1, Table 1. The preprocessing stage reads two ontologies (in Web Ontology Language (.owl) format). Afterward, the framework gives freedom to the user to decide the sequence of ontologies to be merged. Suppose that the user provides a sequence as (O1, O2); then, the O1 ontology is treated as the source ontology, and the O2 ontology is treated as the O1 ontology. This step resolves the limitations of earlier work (Patel & Jain, 2019), in which the framework chooses the source and target ontologies.

The proposed framework uses an RDF graph format to integrate large-scale ontologies. Therefore, the input ontologies are converted to RDF graphs and named RDF Graph1 and RDF Graph2. The source ontology and target ontology partitioning algorithms are written using superclass and subclass structural relations (Babalou & Konig-Ries, 2020). These are applied to divide large-scale source and target ontologies into small-scale ontologies and generate the RDF SubGraph1 and RDF SubGraph2. The partitioning strategy minimizes the large search space (Hnatkowska et al., 2020; Li et al., 2020; Xue & Zhang, 2021; Xue et al., 2021a, 2021b, 2021c; Zhu et al., 2022).

Algorithm_2 gives the pseudocode to partition the source ontology. Algorithm_3 gives the pseudocode to partition the target ontology.

The ontology matching and load balancing stage starts at Step No. 9 and ends at Step No. 11 of Algorithm_1. At this stage, RDF SubGraph1, RDF SubGraph2, and the candidate class are provided as inputs to the ontology matching module. The candidate class is compared with each class of RDF Graph1. The RDF SubGraph1 and RDF SubGraph2 are traversed via the DFS algorithm. The ontology matching module applies parallel processing through the load balancing module. This is mentioned at Step No. 10 in Algorithm_1.

The load balancing module is responsible for managing and optimizing the heavy matching load. It is implemented with the help of the ACO algorithm (Krishnaveni et al., 2019). At this stage, the "*n*" number of machines/nodes is considered to distribute the heavy load of the matching process. During parallel processing, the matching tasks are assigned to machines via their working probability. With this aim, the working probability of each machine is determined via the current running load of each machine, the current maximum load among the "*n*" number of machines, and pheromones. At the initial stage, the same pheromone is assigned to each machine. If all the machines have approximately similar loads, then equal working probabilities are assigned to them. In other cases, the working probability is normalized to the scale of the working load. It supports the fair distribution of matching tasks according to the working ability of each machine. Afterward, the cumulative working probability of each machine is calculated, and accordingly, all the nodes are sorted. It helps to choose first of all the highly probable machines to assign the matching task. This strategy also involves nodes that have low working probabilities during the matching process. The working load of machines to which the matching task is assigned

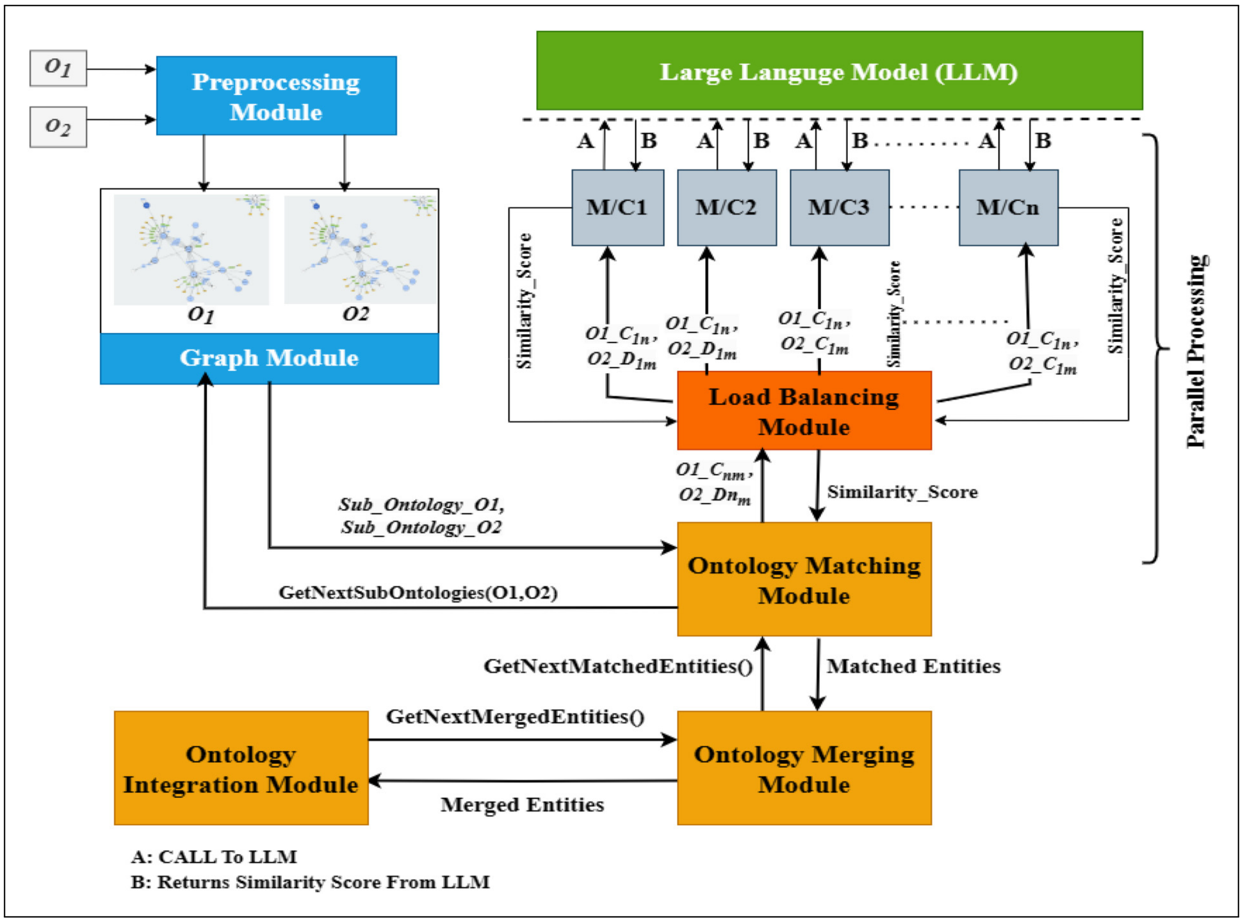

Figure 4. Proposed System Architecture Diagram.

Algorithm_1: OI-LLM

Inputs: Two ontologies O1, O2
Output: Integrated Ontology

```
Step 1   Read Two Ontologies.
Step 2   Ask the user to set the sequence of ontologies to be merged. (Decision about the source and target ontologies).
Step 3   Preliminary Set up of proposed framework.
         Graph_Model1 ← Onto_Graph ($O_1$) // generate RDF graph model for source ontology $O_1$
         Graph_Model2 ← Onto_Graph ($O_2$) // generate RDF graph model for target ontology $O_2$
Step 4   Identify Best Entry Class and prepare a list of Super Best-Entry Classes of source ontology.
Step 5   According to result of Step No. 4, Apply partition algorithm to divide the large-scale source ontology into small-scale
         subontologies.
Step 6   Apply Botttom_Up approach to partition the target ontology
Step 7   According to result of Step No. 6, divide the target ontology into small-scale subontologies.
         Prepare a list of classes of small-scale subontologies.
Step 8   Send a Best Entry Class and a list of Super Best-Entry classes of source ontology and subontologies of target ontology to
         ontology matching module.
Step 9   Recursively call ontology matching process based on Step No. 8 to match the candidate classes of source and target
         ontologies based on Step No. 12.
Step 10  Apply Load Balancing Algorithm to optimize and balance the matching load using parallel computing through Load_Balancing
         Module.
Step 11  Generate API Call to LLM to identify similarity score.
Step 12  Recursively call Merge Process to merge the candidate classes of source and target ontologies based on Step No 11.
Step 13  Repeat the Steps 8, 9, 10, 11, 12 until the end of matching of classes of subontologies.
Step 14  Generate the Integrated Ontology.
```

Algorithm_2: Partitioning of Source_Ontology

```
1 **Input:** RDF Graph1
2 **Output:** Best Entry Point Class and RDF Subgraph1
3 subclasscardinality ← |subclass of O1| //cardinality of subclasses of O1
4 superclassset← { }
5 Bestentry_class← { }
6 count←{ }
7 RDFSubgraph1←{ }
8 C← getallclassesof (O1)
9 For each class Ci in C
  //if Ci is a subclass identified using subclasscardinality
        a. superclassset.add (Ci)
        b. count = count + 1
  //Process to form the divide the source ontology into subontology
10 Sort the elements of superclassset in descending order in the order of count
11 For i = 1 to superclassset.size
        a. Bestentry_class.add (superclassset[i].class)
        b. // add Best_Entry class and all its subclasses and form a subgraph
        c. RDFSubgraph1.add (BestEntryclass)
        d. Remove superclassset[i].class
```

Algorithm_3: Partitioning of Target Ontology

```
1 **Input**: RDF Graph2
2 **Output**: Candidate class and RDF Subgraph2
3 subclasscardinality ← |subclass of O2|
4 superclassset←
5 Candidate class←
6 count←
7 RDFSubgraph2←
8 C← getallclassesof (O2)
9 For each class Ci in C
  //if Ci is a subclass identified via subclass cardinality
        a. superclassset.add (Ci)
        b. count = count + 1
  //Process to divide the target ontology into subontology
10 Sort the elements of the superclass set in ascending order in the order of count
11 For i = 1 to superclassset.size
        a. Candidateclass.add (superclassset[i].class)
        //add Candidate class and all its subclasses and form a subgraph
        b. RDFSubgraph2.add (Candidate class)
        c. Remove superclassset[i].class
```

is increased. In this way, the classes of RDF SubGraph1 are divided among the "*n*" number of machines. The execution of the matching task is initiated to match the candidate class with the classes of RDF SubGraph1.

The pseudocode mentioned in Algorithm_4 describes the operation of the load balancing module. The ACO instance is created at line number 11 of Algorithm_4. The pseudocode of ACO is presented in Algorithm_5.

Algorithm_51 uses the pseudocode to assign matching tasks to different machines via their working probabilities. At line no. 3 of Algorithm_51, the pseudocode for computing the working probability of machines is invoked. It is mentioned in Algorithm_511. Algorithm_52 uses the pseudocode to execute the matching task by internally calling the API calls to LLM and generate the similarity score.

Algorithm_1 proceeds to Step No. 11, where the machines initiate API calls to the LLM through a prompt construction process. The LLM identifies the semantic similarity and sends the answer as a "true" or "false" or "a subclass of Class 1." This result is treated as a similarity score. If the similarity score is "true," then the candidate class is merged with a class of RDF SubGraph1. Otherwise, if the answer is supposed to be "a subclass of Class 1," then the candidate class is merged, and its subclasses are added to RDF SubGraph1 according to the structural relation. The RDF SunGraph2 is traversed via a bottom-to-the-up approach to move to the superclass of the candidate class. The superclass of the earlier candidate class

Algorithm_4: Load_Balancing(List_of_Filtered_Classes, Class2)

Inputs: List of classes of best entry node of O_1 ontology, Candidate class of O_2 ontology
Output: Semantic Result

```
1  Define a class Node NodeId, LLMAPI, Load = 0.0
2  Define a class Task TaskId, Class1_O1, Class2_O2
3  Define a class ACO nodes[Node], tasks[Task], evaporationRate, pheromoneIncrease
4  multipleApis [1...n] ←Call getMultiple.Apis () from APIs
5  pheromones[multipleApis.size] ←1.0
   pheromoneDecreaseFactor ← 0.1
   minPheromoneLevel ← 0.01
6  timeTookArray[multipleApis.size] ← 0
7  compromiseNodes[multipleApis.size]
8  For i = 1 to tasks.size
           unassignedTasks[i] ← tasks[i]
9  For i = 1 to multipleApis.size
           allLLMApiNode.add (Node (i, multipleApis[i]))
10 For i = 1 to List_of_Filtered_Classes.size
           tasks.add (Task (i, List_of_Filtered_Classes [i]), Class2)
11 #create an instance of ACO class
   aco = ACO(allLLMApiNode, tasks, evaporationRate ← 0.1, pheromoneIncrease ← 0.1)
12 Return (Results ← aco.RunACO ())
```

Algorithm_5: ACO

Inputs: Machines/Nodes to generate call to LLM, Matching Tasks, Evaporation Rate, Pheromone Increase Level
Outputs: Collecting Result Of Matching Tasks, Identification of Compromise Machines

```
1   Results←false
2   For task = 1 to unassignedTasks.size
          a. assignment ← assignTask(task, Node)
          b. resultDeffered[result][timeTook]← executeTask(assignment)
          c. allDefferedresults.add(assignment, resultDeffered[result][timeTook])
    //sort the nodes as per required time took to complete the given matching task
    allDefferedresults.sort(Task, Node, timeTook)
3   Call Function Update_Pheromones (allDefferedresults:assignment(Task, Node), result, timeTook)
4   Call Function Evaporate_Pheromones()
5   Identify Compromise Machines to avoid premature convergence of ACO.
```

Algorithm_51: assignTask(task, Node)

```
1   Input : Matching Task and Nodes/Machines
2   Output: Assignment of Matching Task to Machines.
3   Probabilities ← Caclculate_Probabilities (nodes[i])
4   SelectedNode ← Selection of Node Using cumulativeProbability(Probabilities)
5   nodes[SelectedNode].load = nodes[SelectedNode].load + 1
6   return(task to nodes[SelectedNode])
```

is identified as the new candidate class and is involved in the matching process. The matching process is called recursively to match the newly identified candidate class with each class of RDF Graph1. This process is repeated until the end point of RDF SubGraph2 is reached. In this way, the matching process runs in parallel.

The framework stores the matching results that are returned from "*n*" machines and referred during the merging process. The merge process also starts at the same time to merge the overlapped classes of RDF SubGraph2 to RDF SubGraph1. The pseudocode for the above process is presented in Algorithm_521, Algorithm_6. At, line no. 5, of Algorithm_521, Algorithm_6 is invoked that generate API call to LLM. Algorithm_61 uses the pseudocode to call to LLM

Algorithm_511: Caclculate_Probabilities ()

```
1  Input: Nodes/Machines
2  Output: Working Probability of Machines
3  sumCombined←0.0
4  MaxLoad ← getMaxLoadof(nodes.load)
5  For i = 1 to nodes.size
6  loadFactor[i] ← 1 - (nodes[i].load / (maxload + 1))
7  combined[i] ← pheromones(nodes[i]) * loadFactor[i]
8  sumCombined = sumCombined + combined[i]
9  For i = 1 to combined.size
10   If (sumCombined = = 0.0) then // if all nodes have same working load
11   return (probabilities[i] ← 1.0 / combined.size) //assign equal probability to each node
12   Else
return (probabilities [i] ← combined[i] / sumCombined) // normalize the probability value
```

Algorithm_52: executeTask

```
1  Input: assignment
2  Output: Matching Result, timeTook result, time to take
3  task ← assignment.task
4  node ← assignment.node
5  starttime ← Now().time
6  //generate HTTP call to LLM to identify similarity score among the entities assigned through a //task
7  matchingresult = makeHTTP(node, task)
8  endtime ← Now().time
9  calculate timeTook ← endtime - starttime
10   timeTookArray[node.id] + = timeTook
11   return (Matching Result, timeTook)
12   Identification of the heavily loaded machines and the less loaded machines.
13   Shift the load from heavily loaded machines and less loaded machines
```

Algorithm_521: makeHttpCall

```
1    Input: Node, Task
2    Output: Semantic Similarity
     //Extract class1 and class2 from the task
3    Class1 = Task.class1
4    Class2 = Task.class2
5    SemanticSimilarity = Identification_of_Similarity_Score (task.class1, task.class2, Apis. LlmApi, nodeid)
6    Return SemanticSimilarity call Function makeHTTP(node, task)
```

and generate the similarity score. In Stage III, the proposed framework follows Steps No. 12 to No. 14 of Algorithm_1 to generate the integrated ontology in an incremental way.

The pheromone levels of the machines are updated after the execution of Stage III. The proposed framework intelligently handles parallel processing to utilize the computing power of machines. For this purpose, the total amount of time required to complete the assigned task by each machine is computed. The well-worked machines respond in less time. The pheromone level of such well-worked machines is increased. It may be happened that poorly worked machines respond slowly or might not respond after some time. In such cases, the matching process slows down and results in an increase in computational time. Therefore, it is essential to identify such machines to improve the running time of the matching process. For this purpose, the pheromone level of poorly working machines decreases slowly and is repeated for a certain number of iterations. After a certain number of iterations, suppose that one of the not well-worked machine has a negative pheromone level; then, that machine will receive a penalty. This node is declared a compromised node. The compromised node is disqualified in the next iteration, which helps optimize the parallel operation of the matching process. This strategy indirectly improved the computational time.

Algorithm_6: Identification of Similarity Score (using LLM)

Input: Class1 of O1 Ontology, Class2 of O2 Ontology, List of Matching Task, List of Nodes

Output: Similarity Score

```
1   class1Label ← class1.getLabel()
2   class2Label ← class2.getLabel()
3   class2DescriptionApis ← class2.getComment()
4   If (class1Label == null || lass2Label==null) return false
5   If (class1.getComment() == null || class1.getComment() == " ")
    class1Description ← class1.getComment(null)
6   If (class2.getComment() == null || class2.getComment() == " ")
    Class2Description ← class2.getComment(null)
7   isSimilar = askApi(class1Label, class1Description, class2Label, class2Description, Apis.LlmApi, nodeId)
8   Return isSimilar to Function SemanticallySimilar()
```

Algorithm_61: askApi

```
1   Input: class1Label, class1Description, class2Label, class2Description, Apis.LlmApi, nodeId
2   Output: Response from LLM
3   toAsk ← Prompt_Construct
4   //Check cache for existing result If ChatCache.containsKey(toAsk): Return
5   if (ChatCache.get(toAsk))
6   isSimilar ← ChatCache.cache(toAsk)
7   If (isSimilar! = Null) return isSimilar
8   llmApi ← Apis.getLLMApi()//Create ChatCompletionRequest with system and user messages
9   Chatrequest = ChatCompletionRequest
10    (model = LLMDetails.Model_Name + nodeId,
11    messages = [ Message("system," "I will provide two ontology classes class1 and class2 where class1 belongs to O1 ontology
         and class2 belongs to O2 ontology Respond with 'Yes' or 'No' only."), Message("user," toAsk) ])
12    //Send request and get response from LLM
13    response ← llmApi.chatCompletion(request).executeSync()
14    If response is not null && response. Message is not null// validation of a response
15    responseText ← response.message.content.trim().toLowerCase()//cross verifying the response
16    If responseText = = "yes"
17    ChatCache.put(toAsk, true) Return true
18    Else If responseText = = "no"
19    ChatCache.put(toAsk, false) Return false
20    Else Print("Response was not expected: " + responseText)
21    Else Print ("No response from LLM.")
```

Algorithm_612: Prompt_Construct

```
1   Input: class1Label, class1Description, class2Label, class2Description)
2   Output: Prompt
3   Prompt1← "Class1 Name: " + class1Label + ", Description: " + class1Description + "Prompt2← Class2
        Name:" + Class2Label + ", Description: " + class2Description + " Is Class2 is similar to or a subclass of Class1?"
4   Prompt ← Prompt1 + Prompt2
```

6 Evaluation of the Proposed Framework

6.1 Experimental Setup

The benchmark dataset of Anatomy Track provided by the Ontology Alignment Evaluation Initiative (OAEI, 2022) is used to evaluate the performance of the proposed framework. Table 5 lists the data of the input ontologies.

The proposed framework is developed using Kotlin (JVM—version 1.9.22), JDK 19, Apache Jena (version 5.0.0), and LLM Studio (version 0.2.27) with Llama-3-8B Instruct. The proposed framework is executed in a cloud computing environment. Three A100 80G X 896 CPU core machines with 773 GB of RAM and 640 GB of GPUs are used. To run the

Table 5. Dataset of the Input Ontologies.

OAEI track name/Year	Ontologies description	Number of classes
Anatomy Track/2022	Human Anatomy (NCI Thesaurus)	3304
	Adult Mouse Anatomy	2744

Table 6. Evaluation of the Proposed Framework Results With Existing Frameworks.

Ref. & Year of Work	(1)	(2)	(3)	(4)	(5)
(Zhang et al., 2024)LLMA Dialog Model	OIT_1	Ontology Matching(Large Scale)	√	Anatomy Track/2023	Use of Llama Dialog Model.Required Computational Time - 1:56:26 (HH:MM:SS).
(Giglou et al., 2024)LLMs4OM	OIT_1	Ontology Matching(Large Scale)	X	Anatomy Track/2017	Best Model of LLM : GPT-3.5 + Ada
				Phenotype Track/2017	Best Model of LLM: Mistral (CP) + BERT for matching (DOID-DRDO) (HP-MP) Ontologies.
				BIOML Track/2017	Best Model of LLM: GPT-3.5(CC) + Ada for matching (YAGO - Wikidata) Ontologies.
				CommonKG Track /2017	Best Model of LLM : LLaMA-2(C)+Ada
				MSE Track /2017	Best Model of LLM : LLaMA-2(CC)+BERT
(Suleiman et al., 2024)Multi-Criteria Ontology Merging	OIT_3	Ontology Matching and Ontology Integration (Middle Scale)	√	Code Ontology, Software Ontology.	Use of Natural Language Processing Tools and Machine Learning Techniques (BERT) for semantic matching purpose.Required Computational Time - 01:48:50 (HH:MM:SS)
(He et al., 2023a)ELLMOA	OIT_1	Ontology Matching(Large Scale)	X	Bio-ML	It suggested that most capable variant is the GPT-3.5 series.
(Hertling & Paulheim, 2023)OLaLa	OIT_1	Ontology Matching(Large Scale)	√	Anatomy Track/2022	Use of Llama-2-70B-instruct-v28 for matching large-scale ontologies.Required Computational Time: 2:41:23 (HH:MM:SS)
The Proposed Framework	OIT_3	Ontology Matching and Ontology Integration (Large Scale)	√	Anatomy Track/2022	Use of Llama-3-8B-Instruct-v27 for matching large-scale ontologies.Required Computational Time - 00:49:50 (HH:MM:SS)

proposed framework, Kotlin (JVM—version 1.9.22), JDK 19 and Apache Jena (version 5.0.0) are taken into consideration. LLM Studio (version 0.2.27) with Llama-3-8B Instruct was used specifically for the ontology matching process. During the execution process, virtualization is performed along with parallel computing. Therefore, three machines of A100 80G X 896 CPU are virtualized, and four (4) instances are created. Llama-3-8B is loaded on these 3 machines. In this way, three (3) machines X four (4) = 12 instances are created. Therefore, it is possible to match and integrate large-scale ontologies with the use of the minimum number of nodes in parallel processing. In this way, execution of the proposed framework is accomplished to integrate large-scale biomedical human anatomy track ontologies.

6.2 Proposed Framework and Comparative Analysis—Part I

For comparative analysis purposes, existing frameworks that specifically use the LLM in ontology matching and integration processes are considered. The main purpose is to compare the computational time required by the proposed framework with the computational time required by the other frameworks as per their methodologies. The set of parameters is defined and are coded using numbers from (1) to (5). The coding descriptions are the integration category (1), type of ontology

Table 7. Evaluation of Integrated Ontology—Part I.

Input Ontologies	Number of Classes	Number of Properties
Source Ontology : Human Anatomy (NCI Thesaurus)	3304	2
Target Ontology : Adult Mouse Anatomy	2744	3
Integrated Ontology : Human_Mouse	4351	2

Table 8. Evaluation of Integrated Ontology—Part II.

	Source Ontology	Target Ontology	Integrated Ontology
Graph Metrics	Human Anatomy	Adult Mouse	Human_Mouse
Absolute Root Cardinality (ARC)	7	8	7
Absolute Leaf Cardinality (ALC)	2631	2261	2953
Average Depth (AD)	6.255766	3.674295	5.454259
Maximal Depth (MD)	13	8	13
Average Breadth (AB)	4.582656	5.775862	3.54463
Maximal Breadth (AB)	151	1049	454

process and scale of the participating ontologies (2), computational time (3), dataset information (4), and evaluation result details (5). The computation parameter is assessed via (√) and (X) symbols. The (√) symbol indicates that work is done for computation time, and the symbol (X) indicates that work is not done for computational time. Table 6 presents a comparative analysis of the proposed framework with existing frameworks. A comparative analysis revealed that the proposed method completed ontology matching as well as the integration process in only 49 min and 50 s. Compared with the LLMA Dialog Model and OLaLa existing frameworks, the proposed framework yields improvements in computational time.

6.3 Proposed Framework and Comparative Analysis—Part II

The proposed framework considers the RDF graph representation of input ontologies and builds the integrated ontology as RDF graph. Therefore, different graph metrics (Suleiman et al., 2024) are used to evaluate the quality of the integrated ontology. In this study, graph metrics such as Absolute Root Cardinality, Absolute Leaf Cardinality (ALC), Average Depth, Maximal Depth, Average and Maximum Breath are used. The online tool OntoMetrics (Lantow, 2016) is used to evaluate the integrated ontology with the above graph metrics.

Table 7 shows the numbers of classes and object properties. The numbers of classes in the source ontology and integrated ontology are not the same. This finding indicates that one thousand forty-seven (1047) classes of Adult Mouse Anatomy ontology are similar to some classes in Human Anatomy. These were merged into the Human Anatomy ontology to enrich it.

Table 8 shows that the ALC of the integrated ontology is greater than the ALC value of the source ontology. This finding indicates that the proposed framework generates a cohesive integrated ontology. Thus, it has enriched the source ontology with additional related data from the target ontology. Table 9 shows the evaluation of the integrated ontology. Here, the ontology is evaluated as per the setup of evaluation parameters mentioned in Table 3, Section 4.1. This finding indicates that the proposed framework successfully integrates large-scale ontologies.

7 Conclusions

A scalable framework is provided to integrate large-scale ontologies that address the main challenges, such as scalability and computational complexity. It works at different stages. In the first stage, the proposed framework is designed to provide a preliminary setup of ontology matching and integration processes. Afterward, the general workflow of the proposed research is designed and represented via a mathematical model. In the second stage, a mathematical model is used to develop the proposed framework. The proposed framework uses partitioning and load-balancing algorithms to enhance the performance of the critical matching process. The partitioning algorithm uses superclass and subclass relations and the cardinality of subclasses to partition the large-scale RDF graph ontologies into subgraphs without disturbing the structural relations.

Table 9. Evaluation of Integrated Ontology—Part III.

No.	List of Parameters	Requirement	Evaluation Result
EVITO_1	Human Expert Involvement	Yes/No	No
EVITO_2	Scalability	Large Scale	3304 + 2744
EVITO_3	Computational Time	HR: MIN : SEC	00:49:50
EVITO_4	Prohibition of Structural and Relational Redundancy	Yes/No	Yes : AD Value is more than AB Value
EVITO_5	Knowledge Preservation of Source Ontology	Yes/No	Yes
EVITO_6	Consistent Integrated Ontology: Evaluation of Semantic Consistency	Yes/No	Yes
EVITO_7	Redundant Entities	Numerical Value	0
EVITO_8	Number of Unconnected Entities	Numerical Value	454
EVITO_9	Principle of Ontology Coherence	Yes/No	Yes
EVITO_10	Coverage of Entities : Number of Entities Preserved from Input Ontologies	Numerical Value	ALC Value—2953

After partitioning, the classes within the subgraphs are matched concurrently. Here, at this point, the proposed framework applies an ACO algorithm that not only balances the concurrent execution of the matching task but also optimizes it. The load-balancing stage initiates matching and merging processes concurrently and generates the integrated ontology iteratively. The matching process benefits the ability of the LLM to calculate the similarity score. This helps to avoid the extensive computations required to identify the syntactic, semantic, and structural similarity. The partitioning of the ontologies, matching of classes, and merging of classes are performed in an RDF graph only, which also decreases the computational complexity.

The proposed framework is compared with existing frameworks that have specifically used LLM to examine the improvement in computational time. Afterward, the quality of the integrated ontology is evaluated via base and graph metrics. This indicates that the integrated ontology is consistent. The findings revealed that the proposed algorithm successfully integrates large-scale ontologies. In the future, there is a plan to enhance the proposed framework to evaluate the computational time required to integrate large-scale BIO-ML ontologies.

ORCID iDs

Sujata Pardeshi https://orcid.org/0000-0002-5279-1735
Virat Giri https://orcid.org/0000-0002-7497-4057
Sushopti Gawade https://orcid.org/0000-0001-5630-0895

Statements and Declarations

Funding

The authors received no financial support for the research, authorship, and/or publication of this article.

Conflicting Interests

The authors declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

References

Babalou, S., & Konig-Ries, B. (2019). GMRs: Reconciliation of generic merge requirements in ontology integration. International Conference on Semantic Systems.

Babalou, S., & Konig-Ries, B. (2020). Towards building knowledge by merging multiple ontologies with merger: A partitioning-based approach. *CoRR*. https://doi.org/abs/2005.02659

Cynthia et al. Mammalian Phenotype Ontology. https://www.informatics.jax.org/

Geng, A. (2023). A multiobjective particle swarm optimization with density and distribution-based competitive mechanism for sensor ontology meta-matching. *Complex & Intelligent Systems*, *9*(1), 435–462. https://doi.org/10.1007/s40747-022-00814-6

Giglou, H. B., D'Souza, J., Engel, F., & Auer, S. (2024). LLMs4OM: Matching ontologies with large language models. Proc. of the Extended Semantic Web Conference (ESWC) (pp. 1–12). https://doi.org/10.48550/arXiv.2404.10317

He, Y., Chen, J., Dong, H., & Horrocks, I. (2023a). Exploring large language models for ontology alignment. 22nd International Semantic Web Conference (ISWC).

He, F., Wang, D., & Sun, Y. (2023b). Ontology integration for building systems and energy storage systems. 10th ACM International Conference on Systems for Energy-Efficient Buildings. https://doi.org/10.1145/3600100.3623720

Hertling, S., & Paulheim, H. (2023). OLAla: Ontology matching with large language models. K-CAP '23: Proceedings of the 12th Knowledge Capture Conference (pp. 131–139). https://doi.org/10.1145/3587259.3627571

Hnatkowska, B., Kozierkiewicz, A., & Pietranik, M. (2020). Semi-automatic definition of attribute semantics for the purpose of ontology integration. *IEEE Access*, *8*, 107272–107284. https://doi.org/10.1109/ACCESS.2020.300003

Huang, Y., Xue, X., & Jiang, C. (2020). Optimizing ontology alignment through improved NSGA-II. *Discrete Dynamics in Nature and Society*, *2020*. https://doi.org/10.1155/2020/8586058

Huang, Y., Zhuang, Y., & Xue, X. (2022). Solving ontology metamatching problem through improved multiobjective particle swarm optimization algorithm. *Wireless Communications and Mobile Computing*, *2022*, 1–15. https://doi.org/10.1155/2022/1634432

Köhler et al. (2017). Human Phenotype Ontology (jax.org). https://hpo.jax.org/

Krishnaveni, A., Shankar, R., & Duraisamy, S. (2019). A survey on nature-inspired computing (NIC): Algorithms and challenges. *Global Journal of Computer Science and Technology: DNeural & Artificial Intelligence*, *19*(3), 23–37.

Lantow, B. (2016). Ontometrics: Putting metrics into use for ontology evaluation. Proceedings of the 8th International Joint Conference, IC3K (pp. 186–191).

Li, Y., Zhou, J., Liu, J., & Hou, Y. (2020). Matching large scale ontologies based on filter and verification. *Mathematical Problems in Engineering*, *2020*. https://doi.org/10.1155/2020/8107968

Lu, J., & Xue, X. (2020). A compact brain storm algorithm for matching ontologies. *IEEE Access*, *8*, 43898–43907. https://doi.org/10.1109/
ACCESS.2020.2977763

Lynn Schriml et al. (2003) Human Disease Ontology. DO (disease-ontology.org)

Machado, L. M. O., Almeida, M. B., & Souza, R. R. (2020). What researchers are currently saying about ontologies: a review of recent web of science articles. *Knowledge Organization*, *199*(3),199–219. https://doi.org/10.5771/0943-7444-2020-3-199

Negi, S., & Malik, S. K. (2018). An algorithm for merging two ontologies: A case study. *International Journal of Applied Engineering Research*, *13*(12), 10327–10338.

Norouzi, S. S., Mahdavinejad, M. S., & Hitzler, P. (2023). Conversational ontology alignment with ChatGPT. *CEUR Workshop Proceedings*, *3591*, 61–66. https://doi.org/10.48550/arXiv.2308.09217

Ochieng, P., & Kyanda, S. (2018). Large scale ontology matching: State of the art analysis. *ACM Computing Surveys*, *51*(4),1–35. https://doi.org/10.1145/3211871

Ocker, F., Vogel-Heuser, B., & Paredisc, C. J. J. (2022). *A framework for merging ontologies in the context of smart factories. Computers in Industry Journal*, *135*. https://doi.org/10.1016/j.compind.2021.103571

Ontology Alignment Evaluation Initiative - OAEI 2022 Campaign. (2022). https://oaei.ontologymatching.org/2022/index.html

Osman, I., Ben Yahia, S., & Diallo, G. (2021a). Ontology integration: Approaches and challenging issues. *Journal of Information Fusion*, *71*, 38–63. https://doi.org/10.1016/j.inffus.2021.01.007

Osman, I., Pileggi, S. F., Ben Yahiaa, S., & Diallo, G. (2021b). An alignment-based implementation of a holistic ontology integration method. *MethodsX*, *8*, 1–29. https://doi.org/10.1016/j.inffus.2021.01.007

Patel, A., & Jain, S. (2019). A partition based framework for large scale ontology matching. *Recent Patents on Engineering*, *14*(3), 488–501. https://doi.org/10.2174/1872212113666190211141415

Portisch, J., Hladik, M., & Paulheim, H. (2022). Background knowledge in ontology matching: A survey. *Semantic Web, IOS Press*, *14*(2), 1–5. https://doi.org/10.3233/SW-223085

Stoilos, G., Geleta, D., Shamdasani, J., & Khodadadi, M. (2018). A novel approach and practical algorithms for ontology integration. Proc. of 17th International Semantic Web Conference Part I (ISWC) (pp. 458–476). https://doi.org/10.1007/978-3-030-00671-6

Suleiman, M., Rudwan, M., & Fonou-Dombeu, J. V. (2024). A novel algorithm for multi-criteria ontology merging through iterative update of RDF graph. *Big Data Cognitive Computing*, *8*(19), 1–20. https://doi.org/10.3390/bdcc8030019

Verhodubs, O. (2020). Merging of ontologies through merging of their rules. *Applied Mathematics & Information Sciences*. https://doi.org/10.48550/arXiv.2001.04326

Xingsi, X., & Pan, J.-S. (2018). An overview on evolutionary algorithm based ontology matching. *Journal of Information Hiding and Multimedia Signal Processing*, *9*(1), 75–88.

Xue, X., Chen, J., Chen, J., & Chen, D. (2018). Using compact coevolutionary algorithm for matching biomedical ontologies. *Computational Intelligence and Neuroscience*, *2018*, 1–8. https://doi.org/10.1155/2018/2309587

Xue, X., Tan, W., & Lv, J. (2023). Integrating large-scale ontologies for economic and financial systems via adaptive co-evolutionary NSGA-II. *Fractals*, *31*(6), 1–10. https://doi.org/10.1142/S0218348X23401059X

Xue, X., & Tang, Z. (2017). An evolutionary algorithm based ontology matching system. *Journal of Information Hiding and Multimedia Signal Processing*, *8*(3), 551–556.

Xue, X., Wang, H., Zhou, X., Mao, G., & Zhu, H. (2021a). Matching heterogeneous ontologies with adaptive evolutionary algorithm. *Connection Science*, *34*(1), 811–828. https://doi.org/10.1080/09540091.2021.1991278

Xue, X., Wu, X., & Chen, J. (2020). Optimizing biomedical ontology alignment through a compact multiobjective particle swarm optimization algorithm driven by knee solution. *Discrete Dynamics in Nature and Society*, *2020.* https://doi.org/10.1155/2020/4716286

Xue, X., Wu, X., & Chen, J. (2021b). Optimizing ontology alignment through an interactive compact genetic algorithm. *ACM Transactions on Management Information Systems*, *12*(2),1–22. https://doi.org/10.1145/3439772

Xue, X., Yang, C., Jiang, C., Tsai, P.-W., Mao, G., & Zhu, H. (2021c). Optimizing ontology alignment through linkage learning on entity correspondences. *Hindwadi Complexity*, *2021*, 1–12. https://doi.org/10.1155/2021/5574732

Xue, X., & Zhang, J. (2021). Matching large-scale biomedical ontologies with central concept based partitioning algorithm and adaptive compact evolutionary algorithm. *Applied Soft Computing*, *106*(6), 107343. https://doi.org/10.1016/j.asoc.2021.107343

Zhang, S., Dong, Y., Zhang, Y., Payne, T. R., & Zhang, J. (2024). Large language model assisted multi-agent dialog for ontology alignment. Proc. of the 23rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS).

Zhang, L.-Y., Ren, J.-D., & Li, X.-W. (2018). OIM-SM: A method for ontology integration based on semantic mapping. *Journal of Intelligent and Fuzzy Systems*, *34*(4), 1983–1995. https://doi.org/10.3233/JIFS-161553

Zhaoming, L., & Rong, P. (2022). Improving the efficiency of multi-objective grasshopper optimization algorithm to enhance ontology alignment. *Journal of Natural Sciences*, *27*(3), 240–254. https://doi.org/10.1051/wujns/2022273240

Zhou, X., Lv, Q., & Geng, A. (2023). Matching heterogeneous ontologies based on multi-strategy adaptive co-firefly algorithm. *Knowledge and Information Systems*, *65*(6), 2619–2644. https://doi.org/10.1007/s10115-023-01845-2

Zhu, H., Xue, X., & Wang, H. (2022). Matching ontologies through multi-objective evolutionary algorithm with relevance matrix. *Mathematics*, *10*(12), 2077. https://doi.org/10.3390/math10122077

Appendix A: Summary of similarity measures

See Tables A1–A5.

Table A.1. Syntactic Similarity Measures.

References	Equation Title	Mathematical Equation
(Huang et al., 2022)(Xue & Zhang, 2021)(Xue et al., 2018)	N – Gram	$Sim(S_1,\ S_2\ ) = \frac{2\times com(s_1,s_2)}{N_{s1}+N_{s2}}$ Where $com(s_1, s_2)$ is the number of common substrings of s_1 and s_2. N_{s1}, N_{s2} are the number of substrings.
(Xue et al., 2021a, 2021b, 2021c)(Huang et al., 2020)	Levenshtein Distance	$Sim(E_1,\ E_2\ ) = max\left(0,\ \frac{min(\lvert e\rvert,\lvert e_2\rvert - d(e_1,e_2))}{min(\lvert e_1\rvert,\lvert e_2\rvert\ )}\right)$
(Ocker et al., 2022)(Xue et al., 2021a, 2021b, 2021c)(Li et al., 2020)(Xue et al., 2020) (Machado et al., 2020)	String Metric for Ontology Alignment	$Sim(E_1,\ E_2\ ) = com(e, e_2) - diff(e_1, e_2)\ + WinklerImper(e_1, e_2)$
(Xue et al., 2021a, 2021b, 2021c)	Jaro–Winkler Distance	$Sim(E_1, E_2)\ = 1/3\ \times(\lvert common(e\_1, e\_2)\rvert/e\_1 + \lvert common(e\_1, e\_2)\rvert/e\_2 + (\lvert common(e\_1, e\_2)\rvert - \lvert transp(e\_1, e\_2)\rvert) / \lvert common(e\_1, e\_2)\rvert)$
(Xue & Zhang, 2021)	Researcher_Defined	$Sim(S_1, S_2) = \begin{cases} 1 & If\ terms\ are\ synonyms \\ TF\ --IDF(S_1, S_2\ ) & Else \end{cases}$

Table A.2. Semantic Similarity Measures.

References	Resource Name	Narration
(Xue et al., 2021a, 2021b, 2021c) (Hnatkowska et al., 2020) (Zhang et al., 2018) (Xue & Tang, 2017)	WordNet	Used to calculate distance between two words using linguistic relationship such as synonym and hypernym. For this purpose electronic database of vocabulary that combines different words into a group of synonyms is used.
(Xue & Zhang, 2021) (Xue et al., 2020) (Xue & Tang, 2017)	UMLS	It calculates synonym based similarity using largest biomedical Unified Medical Language System (UMLS).
(Huang et al., 2022) (Xue et al., 2021a, 2021b, 2021c) (Li et al., 2020)	Wu and Palmer	$Sim(S_1, S_2) = \frac{2 \times depth(LCA(S_1, S_2)}{depth(S_1) + depth(S_2)}$
(Machado et al., 2020)	Researcher Defined	$Sim(S_1, S_2) = \begin{cases} 1, & \text{if two concepts are synonymous then} \\ 0.5, & \text{if one word is the hypernym of the other} \\ 0, & \text{if synonym and hypernym conditions are false} \end{cases}$
(Xue et al., 2021a, 2021b, 2021c)	Cosine Distance	$cosine\_distance(V_1, V_2) = \frac{V_1 \, . \, V_2}{\lvert V_1 \rvert \times \lvert V_2 \rvert}$
(Huang et al., 2020) (Xue & Tang, 2017) (Machado et al., 2020)	Similarity Flooding	$\delta^{i+1} = norms(\delta^i + (\delta^i))$

Table A.3. Other Types of Similarity Measures Used to Measure Syntactic, Semantic, and Structural Similarity.

References	Equation Title	Mathematical Equation
(Xue et al., 2021a, 2021b, 2021c) (Lu & Xue, 2020) (Xue et al., 2020) (Zhang et al., 2018)	Researches Defined	$Sim(S_1, S_2) = \frac{\sum_{i=1}^{\lvert p_1 \rvert} max_{j=1\,..\,\lvert p^2 \rvert}(sim'^{(P_1,i,P_2,j)}) + \sum_{i=1}^{\lvert p_2 \rvert} max_{i=1\,..\,\lvert p^2 \rvert}(sim'^{(P_2,j,P_1,i)})}{\lvert P_1 \rvert + \lvert P_2 \rvert}$
(Xue et al., 2020)	Researcher Defined	$Sim(S_1, S_2) = \frac{\sum_{i=1}^{\lvert p_1 \rvert} max_{j=1\,..\,\lvert p^2 \rvert}(sim'^{(P_1,i,P_2,j)}) + \sum_{i=1}^{\lvert p_2 \rvert} max_{i=1\,..\,\lvert p^2 \rvert}(sim'^{(P_2,j,P_1,i)})}{\lvert P_1 \rvert + \lvert P_2 \rvert}$

Table A.4. Structure-based Similarity Measures.

References	Equation Title	Mathematical Equation
(Xue et al., 2021a, 2021b, 2021c) (Xue et al., 2021a, 2021b, 2021c)	Researcher_Defined	$Sim(S_1, S_2) = \frac{SimsubNum + SimsuperNum + SimsumSim + SimsuperSim}{4}$
(Xue & Zhang, 2021)(Li et al., 2020)(Xue et al., 2020)	Researcher_Defined	$Sim(S_1, S_2) = \frac{2 \times depth_{dcm}(S_1, S_2)}{depth_{S1} + depth_{S2}}$

Table A.5. Weighted Sum Method Similarity Measures.

References	Equation Title	Mathematical Equation
(Huang et al., 2022)	Weighted Sum	$W_s = \sum_{i=1}^{n} W_i \, S_j$
(Huang et al., 2020)(Xue et al., 2021a, 2021b, 2021c)(Machado et al., 2020)	Average Weighted Aggregation	$A_w = \frac{\sum_{i=1}^{n} W_i \, S_j}{n}, \; W_s = \sum_{i=1}^{n} W_i \, Similarity_{(s1,s2)}$
(Ochieng & Kyanda, 2018)	Average Sum Aggregation	$A_w = \frac{\sum_{i=1}^{n} S_j}{n}$