DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS
CHENNAI – 600036

Ontology-Driven Automated MCQ Generation

A Thesis

Submitted by

DEBASMITA MANDAL

For the award of the degree

Of

MASTER OF TECHNOLOGY

November 2025

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS
CHENNAI – 600036

Ontology-Driven Automated MCQ Generation

A Thesis

Submitted by

DEBASMITA MANDAL

For the award of the degree

Of

MASTER OF TECHNOLOGY

November 2025

THESIS CERTIFICATE

This is to undertake that the Thesis titled **ONTOLOGY-DRIVEN AUTOMATED MCQ GENERATION**, submitted by me to the Indian Institute of Technology Madras, for the award of **Master Of Technology**, is a bona fide record of the research work done by me under the supervision of **Prof. P Sreenivasa Kumar**. The contents of this Thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Chennai 600036 **Debasmita Mandal**

Date: November 2025

Prof. P Sreenivasa Kumar
Research advisor
Professor
Department of Computer Science and Engineering
IIT Madras

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my mentor Prof. P Sreenivasa Kumar for giving me the opportunity to work under him. I really appreciate his constant support and guidance throughout the semester that have helped me solve the problems and get the results in a short span of time. His willingness to discuss new ideas outside the traditional textbook approaches have greatly helped in giving shape to the project.

I would also like to thank Prof. Madhu Mutyam, Head Of Department, Dept. of Computer Science and Engineering, IIT Madras for providing the environment and resources needed for completing the project work.

A special thanks to my parents, my friends in IIT Madras and my fellow AIDB labmates for extending the constant support throughout the duration of the project.

ABSTRACT

KEYWORDS Ontology, Semantic Web, MCQ, RDF, OWL, NLP

In recent years, the growing emphasis on intelligent education systems has led to an increased interest in automating the process of question generation for learning and evaluation purposes. Traditional approaches often fail to capture the deeper semantics and conceptual dependencies among topics and creates questions that are contextually inconsistent. On the other hand, ontologies provide a structured and machine understandable way to represent domain knowledge through entities, classes, and their relationships. Using this structure offers an opportunity to design automated question generation systems that are not only scalable but also semantically meaningful.

Through this project, we propose an ontology-driven system for generating multiple-choice questions (MCQs) automatically from domain ontologies. The system reads ontology files represented in the Web Ontology Language (OWL) and processes them using a sequence of modular components. It first extracts semantic relationships such as subclass hierarchies, role relations, data properties, and chained relations from the ontology graph using Owlready2 and RDFlib libraries. These relationships are then mapped to parameterized natural language templates, by filling which the system formulates grammatically correct and conceptually valid questions.

Our final aim is to create a platform that can help educational institutions, e-learning platforms, and examination systems in automating the creation of large and meaningful question banks with minimal human involvement.

In this Phase 1 of the project, we have focused mainly on the implemention of the ontology parsing and question generation modules, and displaying our system's ability to generate MCQs across multiple relation types. We have also implemented various ideas to improve linguistic clarity and diversity in the generated questions.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

With the rapid growth of structured data and the rise of education systems, there is a incraesing demand for automated tools that can generate high quality questions for teaching, assessment, and training. Traditional question generation systems usually rely on minimal text parsing, which often fails to capture the deeper semantics between concepts. As a result, the generated question sometimes lack diversity and depth. In contrast, ontologies provides a structured and semantically rich representation of knowledge domains. They have the ability to define concepts, relationships, and hierarchies which can be utilized to generate valid and grammatically correct multiple-choice questions.

We have chosen the system to follow an ontology-driven approach because it allows to operate independently of a specific domain, as it draws directly from the logical relationships encoded in the ontology itself. By representing knowledge in the form of triples (*subject–predicate–object*), the system can look over different kinds of relations, such as subclass hierarchies, role-based properties, and data attributes. This helps to ensure that the generated MCQs are in line with the concepts of the chosen domain. Such a method is adaptive as well, as new domains can be supported simply by loading a different ontology file, without altering the underlying code.

The initial stages of our proposed system introduces a modular pipeline that processes ontologies, by parsing it using the *Owlready2* and *RDFlib* libraries to extract entities and their semantic relations. These relations are then stored as structured datasets and linked to natural-language templates. The core of the system lies in the mapping of ontology triples to these templates to create meaningful questions. In addition, we have build a distractor generation mechanism, that supports hierarchical and sibling relations within the ontology to produce incorrect options for the MCQs.

This framework can enable educational institutions, e-learning platforms and/or digital tutors to automatically produce question banks that comes with the underlying ontology. These generated MCQs could also be used as a foundation for adaptive quizzes, learning assessment tools, or tutoring systems that adjust to a learner's understanding.

This report outlines the following:

- Chapter 2 provides the theoretical background of ontology-based systems and also discusses the existing work and challenges in question generation.
- Chapter 3 explains the design, architecture, and implementation details of our proposed ontology driven MCQ generation system. It covers topic like ontology parsing, relation extraction, question synthesis, and the distractor generation algorithm.
- Chapter 4 presents us with the experimental evaluation, analyzing the quality, diversity, and the accuracy of the generated questions.
- Chapter 5 gives us a summary on the prospect of future work on this project.

CHAPTER 2

BACKGROUND AND RELATED WORK

In this chapter, we will explore the theoretical data of ontology based modeling and how they help in question generation. Ontology driven systems form the base for a wide range of applications such as knowledge retrieval, semantic reasoning, and automated content generation.

- We will begin by introducing the main ideas behind ontologies like, what they are, how they are structured, and why they are important for representing knowledge.
- We will then look at the key Semantic Web technologies that make machine understanding of data possible.
- Finally we will review different studies that are done for MCQ generation and discuss how ontology-based approaches help us overcome the common challenges.

2.1 ONTOLOGY FUNDAMENTALS

An ontology is a representation of the data that defines a set of concepts within a certain domain and the relationships among those concepts. The purpose of an ontology is to provide a common understanding of a domain that can be communicated between humans and machines. An ontology defines the key concepts or categories in a domain (called **classes**), the specific examples/instances of those concepts (**individuals**), and the connections between them (**properties**).

According to Gruber [Gruber (1993)], "an ontology is an explicit specification of a conceptualization." , meaning that the ontology captures the essential structure of knowledge about a particular domain. Each ontology typically comprises of three following fundamental elements:

(i). **Classes:** These represent the primary concepts or categories in the domain. For example, in a movie ontology, classes may include *MovieName*, *Director*, and *Actor*.

(ii). **Properties:** These define the relationships or attributes associated with the classes. They are categorized into the two followings:

- **Object Properties**: relations between two classes, e.g.,

  Movie `hasDirector` *Director*

  So here *hasDirector* is an object property.

- **Data Properties**: relations between a class and literal data, e.g.,

  Movie `hasReleaseYear` *2020*

  So here *hasReleaseYear* is a data property.

(iii). **Individuals:** Instances of classes representing objects (e.g., *Inception*, *Christopher Nolan*, etc.).

Ontologies helps to organize these classes in a hierarchical structure through *subclass–superclass* relationships. This enables properties to be inherited mostly by related concepts and allows the system to reason and interpret from the provided data.

2.2 SEMANTIC WEB AND KNOWLEDGE REPRESENTATION

Ontologies acts as the foundation of the Semantic Web, an advanced layer of the web which is designed to give information, so that computers and humans can work together effectively. The Semantic Web framework includes several technologies which are standardized by the W3C, which are as follows:

- **RDF (Resource Description Framework):** It is the fundamental data model of the Semantic Web, which represents information as a set of triples:

  { subject, predicate, object }

  Each triple means a simple statement about resources and their relationships.

  Example: Consider the statement "Inception is directed by Christopher Nolan." Here,

  - **Subject:** *Inception* (the described resource)
  - **Predicate:** *is directed by* (the relationship)
  - **Object:** *Christopher Nolan* (the linked resource)

  This triple can be represented as:

$$\{Inception,\ isDirectedBy,\ ChristopherNolan\}$$

- **RDFS (RDF Schema):** RDFS is and extension of RDF by adding vocabulary to describe the relationships between classes and properties. It allows the hierarchy creation: for example, defining that one class is a more specific type of another, or that certain properties apply only to particular classes.
- **OWL (Web Ontology Language):** OWL is a language that is built on RDF/RDFS, helping to define complex relationships and allow systems to reason and draw conclusions from existing data.
- **SPARQL:** It is a query language for retrieving and manipulating RDF data. It allows users to extract information and patterns from RDF graphs.

Together these components make it possible to represent and query knowledge in a meaningful way, allowing systems to understand and interpret information from different sources.

2.3 EXISTING WORKS IN ONTOLOGY-DRIVEN QUESTION GENERATION

Several studies have explored ontology based question generation, mainly focusing on academic and e-learning domains. Few examples are:

- The paper by Vinu E. Venugopal [Venugopal and Kumar (2015)] showcases one of the most thorough ontology-driven frameworks for generating questions. Their method uses Description Logic (DL) semantics and the open-world assumption to create concept and role based questions.
- The author Vinu E. V. [V. and Kumar (2017)] presented an automated assessment test generation system that uses domain ontologies to create MCQs, with the approach to integrate ontology reasoning to evaluate data.
- Andrew and his team [Andrew Tran and Rama] explored the use of Large Language Models (LLMs) to generate multiple-choice questions in computing courses. Their study found that GPT-4 was able to produce high-quality questions with nearly two

times the accuracy of GPT-3, suggesting that LLMs can support rapid, real-time quiz generation for any use.

- Group of researchers from East Asia [Shunyang Luo and Jiang (2014)] have proposed a framework for generating domain specific MCQs from scientific literature using LLMs.
- Al-Yahya [Al-Yahya (2014)] uses ontology structures to produce multiple choice question.
- The paper by Andreas [Papasalouros (2008)] elaborated how he has worked on natural language question generation from OWL ontologies.
- OntoQue [Al-Yahya (2011)] framework generates quiz questions from OWL ontologies, for educational assesment based on domain ontologies.

These studies show that ontology-based approaches create questions that are better aligned with the domain and more accurate in meaning, making them easier to interpret than those produced by text-based systems.

2.4 WHY CHOOSE ONTOLOGY OVER TEXTUAL OR DOCUMENT-BASED SOURCES?

While text or PDF based approaches rely on unstructured natural language, ontologies can provide a structured, machine understanding representation of knowledge.

- **Semantic Structure:** Ontologies represent data in the form of well-defined relationships {classes, properties, individuals}, enabling direct reasoning and relation extraction without ambiguity.
- **Domain Independence:** Unlike document based systems bound to a specific content, ontology driven frameworks can be applied across multiple domains by simply loading different OWL files.
- **Reasoning and Consistency:** Ontologies support logical inference and consistency checking, allowing the system to validate relationships, because of which we can avoid generating meaningless questions.

Thus, ontology-based approaches offer greater precision, adaptability, and automation compared to traditional text driven question generation systems.

2.5 CHALLENGES IN ONTOLOGY-DRIVEN MCQ GENERATION

Despite the fact that ontologies provide a strong base, it still faces several challenges:

- **Distractor Quality:** Creating incorrect options that are unambiguous but still clearly wrong is tough. Good distractors need to be close in meaning to the correct answer without being confusing.

- **Ontology Design:** Every ontology is built differently depending on the domain and on the choice of the designer. Sometimes it may be difficult to apply a single method across all ontologies.

- **Natural Language Realization:** Turning structured RDF triples into natural sounding sentences can be a challenge.

- **Evaluation Metrics:** Measuring how good or educationally useful the generated questions are is not easy. Some reliable automatic metrics for question quality are still an open area of research.

2.6 PROJECT PLAN AND PROGRESS

Following are the stages and plan of implementation for the complete ontology-driven MCQ generation framework:

- Develop an ontology parsing and RDF graph construction module using `Owlready2` and `RDFlib` libraries.

- Design and implement a pattern extraction module to identify taxonomy, object property, and data property relations.

- Create a template library that enables question generation independent of any specific domain.

- Implement the MCQ generation pipeline along with the distractor selection algorithm by using the ontology hierarchy and sibling relations.

In this phase of the project, we will be implementing the complete ontology parsing and question generation pipeline. The system will successfully loads OWL ontologies, extract relational patterns, and produce MCQs with logically valid distractors. The framework will be tested with few sample ontologies. In the end, we will evaluate the results with certain parameters to determine its relevance.

CHAPTER 3

ONTOLOGY-DRIVEN MCQ GENERATION FRAMEWORK IMPLEMENTATION

This chapter describes the detailed implementation of our proposed ontology-driven MCQ generation system. The system has been developed entirely in Python and integrates several open-source semantic web libraries such as `Owlready2`, `RDFlib`, and `Pandas`. Each module of the project contributes to a distinct functionality in the MCQ generation pipeline, starting with ontology parsing and ending with natural language question creation. This modular organization of the implementation helps us to maintain flexibility across multiple knowledge domains.

3.1 SYSTEM OVERVIEW

The ontology-based MCQ generation process generally involves the following steps:

1. **Ontology Loading and Parsing:** The system loads an OWL ontology using tools like `Owlready2`, identifying entities, relationships, and literals.

2. **Triple Extraction:** All semantic relations (i.e, triples) are extracted in the form { subject, predicate, object } and are categorized into types such as subclass, object property, or data property.

3. **Template Mapping:** Each type of relation is mapped to a corresponding natural language question template stored in a JSON file. For instance, a triple (`Movie, directedBy, Director`) maps to “Who directed the movie `Movie`?”

4. **Distractor Generation:** For MCQs, we eill be generating incorrect yet plausible options, by finding sibling entities or similar classes.

5. **Question Assembly and Output:** The generated questions are formatted into MCQ format and stored in CSV form for further use.

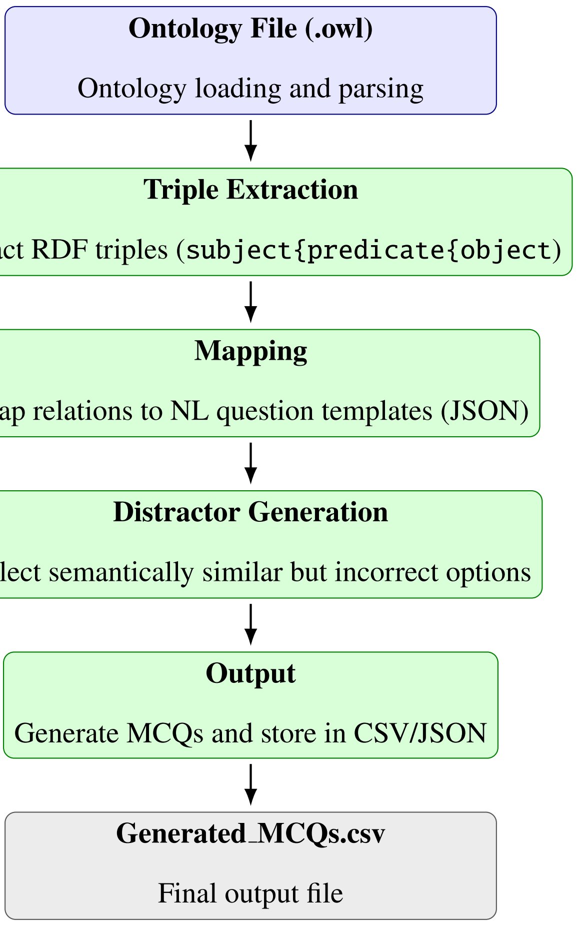
Ontology File (.owl)
Ontology loading and parsing
Triple Extraction
Extract RDF triples (subject{predicate{object)
Mapping
Map relations to NL question templates (JSON)
Distractor Generation
Select semantically similar but incorrect options
Output
Generate MCQs and store in CSV/JSON
Generated_MCQs.csv
Final output file

Figure 3.1: Complete ontology-to-MCQ generation pipeline.

The complete pipeline is composed of seven Python scripts executed sequentially as shown below:

`Environment_Setup.py` ⇒ `Utility_Files.py` ⇒ `PartA_Ontology_Loader.py` ⇒ `PartB_RDF_Graph_Builder.py` ⇒ `PartB_Relation_Extractor.py` ⇒ `PartB_Template_Generator.py` ⇒ `PartC_MCQ_Generator.py`

Each module performs a specific function within the ontology driven question generation pipeline, starting from environment setup to final MCQ formation. The modularity allows individual testing and debugging at every stage.

The overall design emphasizes on separate sets of concerns:

- Data extraction and semantic reasoning are handled independently from question formulation.
- Intermediate CSV and JSON outputs make the process transparent.
- Each stage can be extended (for example, replacing the question template repository or upgrading the distractor algorithm) mostly without modifying the other modules.
- The modular flow can ensure parallel experimentation on individual components such as ontology parsing, relation extraction, and question synthesis.

3.2 ENVIRONMENT SETUP AND DEPENDENCY MANAGEMENT

The initial setup of the project is managed through `Environment_Setup.py` module, ensuring the execution environment is ready with all dependencies. Since all our experiments were conducted using Google Colab, our script also mounts Google Drive for easy file access. It installs libraries essential for ontology parsing and RDF graph processing.

- Installation of the essential libraries: $\boxed{\text{owlready2}}$, $\boxed{\text{rdflib}}$, and $\boxed{\text{pandas}}$.
- Mounting of Google Drive at */content/drive* to enable reading and writing ontology files.

The drive mounting enables reading and writing large ontology files directly from Google Drive, which is important and easier for larger OWL files. The installed libraries play a crucial role in the system, like: `Owlready2` provides interfaces for ontology loading and reasoning, `RDFlib` enables RDF graph parsing and SPARQL querying, and `Pandas` enables structured data manipulation and CSV handling across the pipeline. Together, these libraries form the foundation for semantic data processing and MCQ generation.

3.3 UTILITY FUNCTIONS AND ONTOLOGY HANDLING

The `Utility_Files.py` module is a collection of essential helper functions that support ontology loading, label extraction, file path management, and general debugging. All are used by all other parts of the system. Few of them are as follows:

3.3.1 Ontology Loading and Reasoning

First we are reading the ontology file using *Owlready2* library. Our system provides an option to run reasoning using the **HermiT** reasoner to derive additional relationships such as transitive or equivalent class connections.

Algorithm 1: Ontology Loading and Reasoning

```
Input: Ontology file path $P$, reasoning flag $R$ (boolean)
Output: Ontology object $O$, ontology file reference $F$
1 begin
2     Load ontology from file path $P$ into ontology object $O$;
3     if $R$ is True then
4         Invoke ontology reasoner to infer additional relationships;
5         Save the inferred ontology as reasoned.owl;
6         Set $F \leftarrow$ reasoned.owl;
7     else
8         Set $F \leftarrow P$;
9     return $(O, F)$;
```

This dual mode design allows the user to toggle reasoning on or off depending on the complexity of the ontology. When reasoning is disabled, loading is significantly faster.

3.3.2 Label Extraction and Normalization

Ontology entities are often represented in the form of long IRIs (e.g., *http://example.org/Movie#Inception*). To make sure its readable, our system converts IRIs into clean, human friendly strings. It takes both object attributes and string parsing with regular expressions into account.

Algorithm 2: Human-Readable Label Extraction from IRI

```
Input: Ontology entity $E$
Output: Clean and human-readable label $L$
1 begin
2     if E has a predefined label attribute then
3         Retrieve and return the first available label;
4     Convert $E$ to its string representation (IRI);
5     Extract the fragment after ‘#‘ or the last ‘/‘ as a tentative label;
6     Decode any URL-encoded characters;
7     Replace underscores and hyphens with spaces, and remove extra whitespace;
8     return normalized label $L$;
```

Here, for removal of underscores and hyphens, making labels natural for question templates, we have used `re.sub(r"[_]+", " ", label)` in the code.

3.3.3 Ontology Summary and Debugging

This helper function prints the total count of classes, individuals, and object properties, helping to verify that the ontology has been parsed correctly and contains sufficient content for question generation.

Algorithm 3: Summary

```
Input: Ontology object $O$
Output: Counts of classes, individuals, and object properties
1 begin
2     Retrieve all classes from $O$ and store in list $C$;
3     Retrieve all individuals from $O$ and store in list $I$;
4     Retrieve all object properties from $O$ and store in list $P$;
5     Display the summary to the User;
```

3.4 ONTOLOGY INITIALIZATION

This module is the entry point of the system's execution. It specifies the ontology path, triggers loading, and prepares the ontology for RDF conversion.

It also abstracts the ontology source, allowing easy switching between different OWL files without modifying subsequent code.

3.4.1 Optional Reasoning Feature

Reasoning is a process that states implicit knowledge based on the logical axioms defined in the ontology. When the user sets `USE_REASONER = True`, the system invokes the built in **HermiT** reasoner in Owlready2 to derive additional facts such as transitive subclass relationships, property equivalences, and inherited attributes. This expanded knowledge base enables the generation of more diverse and semantically deeper MCQs.

For example, consider a scenario in an ontology, where the property *partOfFranchise* is defined as a transitive relationship. Suppose the movie *The Dark Knight* is defined as part of the *BatmanSeries*, and the *BatmanSeries* itself is defined as part of the broader *DCUniverse*. Without reasoning, the system recognizes only the two direct relationships: *The Dark Knight* → *BatmanSeries* and *BatmanSeries* → *DCUniverse*.

However, when reasoning is enabled, the ontology infers a new transitive relationship ,that *The Dark Knight* is also part of the *DCUniverse*. This inference is not stated in the ontology but is logically derived through the transitive property of *partOfFranchise*.

Such reasoning allows the MCQ generator to form more in-depth questions (With *The Dark Knight* as the correct option here):

> *"Which of the following movies belong to the DC Universe?"*

3.5 RDF GRAPH CONSTRUCTION

This module is responsible for transforming the ontology into an RDF (Resource Description Framework) graph representation. This conversion is an important step because all subsequent relation extraction, querying, and template mapping operations rely on the structured graph model. RDF provides a standard way to represent semantic data as triples of the form `(subject, predicate, object)` which enables good reasoning.

3.5.1 Graph Parsing and Namespace Detection

Algorithm 4: Conversion of Ontology to RDF Graph and Namespace Detection

```
Input: Ontology file $F$
Output: RDF graph $G$, detected base namespace $GEN$
1  begin
2      Initialize an empty RDF graph $G$;
3      Parse ontology file $F$ in XML format into $G$;
4      Retrieve ontology IRIs from $G$ where type is OWL.Ontology;
5      if ontology IRIs are found then
6          Set base IRI as the first ontology IRI followed by “#”;
7      else
8          Collect all non-w3.org namespaces from $G$;
9          if such namespaces exist then
10             Set base IRI as the first available namespace;
11         else
12             Use default base IRI: http://default.org/ontology#;
13     Define namespace $GEN$ using the detected base IRI;
14     Bind $GEN$ prefix to the RDF graph $G$;
```

This part dynamically detects the base namespace of the ontology and binds it to a generic prefix, avoiding hard-coded URIs. In real world scenarios, each ontology may have a different IRI structure (e.g., *http://example.org/movie#* or *http://data.edu/biology#*). By determining this automatically and binding the namespace, the system makes sure reusablility of the framework across various domains.

3.5.2 Automatic Label Property Detection

Many domain ontologies do not use the standard `rdfs:label` to describe entity names. Instead, they may use predicates such as `hasName`, `title`, or `displayLabel`. If we hardcode label predicates for each ontology, it would make the system very inflexible. Therefore a mechanism is introduced to automatically identify which literal property most frequently represents human readable labels.

Algorithm 5: Dynamic Detection of Label Property

```
Input: RDF graph $G$
Output: Identified label property $L_p$
1 begin
2     Extract all properties $P$ from triples in $G$ whose objects are of type Literal;
3     if $P$ is not empty then
4         For each property $p \in P$, count the number of triples in which $p$ occurs;
5         Select the property with the highest frequency as the label property $L_p$;
6     else
7         Assign $L_p \leftarrow$ RDFS.label as the default label property;
8     Return $L_p$;
```

This idea determines the dominant label property in the RDF graph. To illustrate, consider the following fragment from *Cinema.owl* [Ricardo (2023)]:

```
<Movie rdf:about="Inception">
    <hasName>Inception</hasName>
    <hasReleaseYear>2010</hasReleaseYear>
```

```
</Movie>
```

If the ontology uses `hasName` instead of `rdfs:label`, the above code will detect and assign it as the active label property, ensuring that later modules generate a question like:

> *"Who directed the movie Inception?"*

instead of printing raw URIs such as `http://example.org/cinema#Inception`.

3.5.3 Graph Validation and Triple Counting

After graph construction, it is important to validate whether the ontology has been parsed correctly and contains a sufficient number of semantic relations or not. The below validation code counts the total triples and reports entity diversity:

Listing 3.1: Summary (Here g is the parsed RDF graph)

```
triples = len(g)
subjects = len(set(g.subjects()))
predicates = len(set(g.predicates()))
objects = len(set(g.objects()))
```

A very small triple count might indicate a parsing failure, while unusually high counts might warrant limiting queries for performance reasons. For example, a 20 MB ontology typically yields between 30,000 and 60,000 triples. Such statistics help in setting thresholds for sampling during relation extraction.

3.6 TEMPLATE EXTRACTION

This module is responsible for identifying and extracting different types of semantic relations from the RDF graph created in the previous stage. The extraction is performed using **SPARQL queries** executed over the RDFLib graph. Each type of relations like - $\boxed{\text{taxonomic, object property, or data property}}$ is stored in a separate CSV file to maintain modularity and smooth debugging.

3.6.1 Taxonomy Relations (`rdfs:subClassOf`)

The first type of relation extracted is the taxonomy or hierarchical relationship between ontology classes. This is identified using the `rdfs:subClassOf` predicate, which connects a subclass to its parent class.

Algorithm 6: Extraction of Taxonomy Relations

```
Input: RDF graph $G$
Output: A CSV file with subclass–parent pairs and labels
1 begin
2     Define SPARQL query $Q$: "SELECT ?child ?parent WHERE { ?child
      rdfs:subClassOf ?parent.}";
3     Execute $Q$ on $G$ to obtain result set $R = \{(child, parent)\}$;
4     Create a table $DF$ with columns child, parent from $R$;
5     For each row in $DF$, compute child_label ← safe_get_label(child) and
      parent_label ← safe_get_label(parent);
6     Export $DF$ to a .csv file;
```

This relation is crucial for generating conceptual or category-based questions, such as:

> *"Which of the following is a subclass of Mammal?"*
>
> *"What category does Amphibian belong to?"*

By analyzing class hierarchies, the system can form taxonomy-based MCQs that test conceptual understanding.

3.6.2 Object Property Relations

The second type of relation involves **object properties**, represented in OWL ontologies using the `owl:ObjectProperty` construct. These relations connect two classes or entities and represent real-world associations between them: for instance, *Movie* *directedBy Director* or *Book writtenBy Author*.

Algorithm 7: Extraction of Object Property Relations

```
Input: RDF graph $G$
Output: A CSV file containing {subject–property–object triples} with labels
1 begin
2     Retrieve all properties $P$ from $G$ where type is OWL.ObjectProperty;
3     Initialize an empty list $R$;
4     foreach property $p$ in $P$ do
5         Extract all triples $(s, p, o)$ from $G$ and append them to $R$;
6     Create a table $DF$ with columns property, subject, object using $R$;
7     For each row in $DF$, derive property_label, subject_label, and
       object_label using safe_get_label();
8     Export $DF$ to a .csv file;
```

The object property relations are essential because they allow the system to generate factual, role-based questions, for example:

"Who directed the movie Inception?"

"Which actor performed in the movie Interstellar?"

These questions hold relational knowledge between entities, directly testing the learner's understanding of role associations in the domain.

3.6.3 Data Property Relations

The third type of relation extracted is the **data property**, represented by `owl:DatatypeProperty`. Unlike object properties that connect two entities, data properties connect an entity to a literal value, such as a number, string, or date. Examples include *Movie hasReleaseYear 2010* or *Book hasPageCount 350.*

Algorithm 8: Extraction of Data Property Relations

```
Input: RDF graph $G$
Output: A CSV file containing {subject–property–value} triples with labels
1  begin
2      Retrieve all properties $P$ from $G$ where type is OWL.DatatypeProperty;
3      Initialize an empty list $R$;
4      foreach property $p$ in $P$ do
5          Extract all triples $(s, p, o)$ from $G$;
6          if $o$ is a Literal then
7              Append $(p, s, o)$ to $R$;
8      Create a table $DF$ with columns property, subject, value using $R$;
9      For each row in $DF$, derive property_label and subject_label;
10     Export $DF$ to a .csv file;
```

Data property relations are particularly valuable for generating factual or numeric questions that check for recall of specific information. For example:

"What is the release year of the movie Titanic?"

"What is the runtime duration of Avatar?"

By using literal values, the system can generate diverse questions by covering dates, numbers, and string-based facts.

3.6.4 Relational Chains

A relational chain represents a sequence of linked object properties connecting entities through an **intermediate** node, capturing indirect associations within the ontology.

The implementation randomly samples pairs of object properties from the ontology and constructs SPARQL queries that identify two-step paths between entities, following the pattern *?x p1 ?y . ?y p2 ?z.* This ensures that the system efficiently explores deeper connections without querying all possible combinations, which would be computationally expensive (especially for large ontologies).

Algorithm 9: Extraction of Relational Chain Patterns

```
Input: RDF graph $G$, list of object properties $P$
Output: A CSV file containing two-hop relational patterns
1  begin
2      Randomly select up to 20 properties from $P$ to form a subset $P_s$;
3      Initialize an empty list $R$;
4      foreach pair of properties $(p_1, p_2)$ in $P_s$ do
5          Construct SPARQL query $Q$: “SELECT ?x ?y ?z
6          WHERE { ?x <p1> ?y .  ?y <p2> ?z .}”;
7          Execute $Q$ on $G$ and collect all results $(x, y, z)$;
8          Append each $(p_1, p_2, x, y, z)$ to $R$;
9      Create a table $DF$ with columns property1, property2, subject,
        intermediate, and object;
10     Export $DF$ to a .csv file;
```

Relational chains add semantic depth to the question generation process by linking entities through intermediary relationships. For instance, in the *Cinema.owl* ontology, if the RDF graph contains:

”Movie directedBy Director” and ”Director worksFor Studio”

the system identifies the transitive chain: `Movie` $\longrightarrow$ `Director` $\longrightarrow$ `Studio`.

This can be used to generate higher-order inferential questions such as:

”Which studio is associated with the director of the movie Inception?”

3.6.5 Sibling Class Relations

Two classes are considered siblings when they share the same parent in the ontology hierarchy, i.e., they both appear as subclasses of the same super-class. This relation is identified using a self-join SPARQL query over the `rdfs:subClassOf` predicate, ensuring that pairs of sibling classes are retrieved without duplication.

Algorithm 10: Extraction of Sibling Class Relations

```
Input: RDF graph $G$, predefined result limit $L_{sib}$
Output: A CSV file containing sibling class pairs with labels
1 begin
2     Define SPARQL query $Q$: "SELECT ?e1 ?e2 ?parent WHERE { ?e1
       rdfs:subClassOf ?parent .  ?e2 rdfs:subClassOf ?parent .
       FILTER(?e1 != ?e2)}";
3     Execute $Q$ on $G$ and retrieve up to $L_{sib}$ results;
4     Store results in table $DF$ with columns entity1, entity2, and parent;
5     For each column in $DF$, compute human-readable labels using
       safe_get_label();
6     Export $DF$ to a .csv file;
```

For instance, if in the ontology contain:

```
Dog rdfs:subClassOf Mammal
Cat rdfs:subClassOf Mammal
```

the system identifies `Dog` and `Cat` as sibling classes under `Mammal`. This knowledge allows the generator to construct a question such as:

"Which of the following is also a Mammal like Dog?"

Sibling class relations therefore help to maintain the proximity between correct and incorrect answers, improving the overall standard of the generated MCQs.

3.7 QUESTION TEMPLATE CONSTRUCTION

The *PartB_Template_Generator.py* module defines the linguistic framework for question formulation. Rather than hardcoding fixed question texts, this component dynamically organizes reusable and parameterized natural language patterns based on the semantic relations previously extracted from the ontology. Each relation type has its own set of linguistic templates that act as generic blueprints for converting RDF triples into grammatically correct questions.

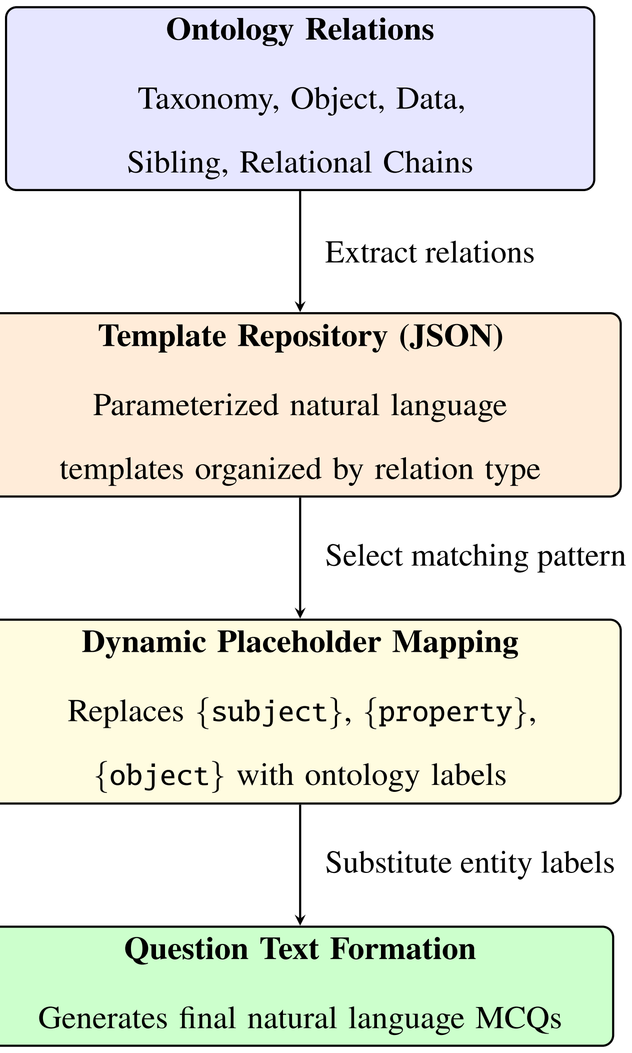
Ontology Relations
Taxonomy, Object, Data,
Sibling, Relational Chains
Extract relations
Template Repository (JSON)
Parameterized natural language
templates organized by relation type
Select matching pattern
Dynamic Placeholder Mapping
Replaces {subject}, {property},
{object} with ontology labels
Substitute entity labels
Question Text Formation
Generates final natural language MCQs

Figure 3.2: Process flow of ontology-driven question template construction and dynamic mapping.

The generated templates are stored in a structured JSON file to ensure modularity and easy customization across different domains. This design abstracts the question language layer from the ontology structure, enabling the same templates to be applied to any OWL file by simply substituting relation-specific variables.

Unlike a static or rule-based system, this design treats the templates as a **linguistic mapping layer** between ontology relations and human language. The placeholders enclosed in braces (e.g., {`subject`}, {`property`}, {`child`}) are automatically substituted with the corresponding labels extracted from the RDF graph.

Motive for Template-Based Generation

The reason behind using this module is because of language flexibility. The key advantages of this approach are as below:

- **Domain Independence:** The same template repository can be reused across multiple ontologies (e.g., *Cinema.owl*, *Biology.owl*) because placeholders are replaced with domain-specific entities at runtime.
- **Language Flexibility:** New linguistic syntax or alternate phrasings can be introduced simply by appending new entries in the JSON file without modifying program logic.
- **Maintainability:** All templates are centrally stored, making the system easily updatable and less prone to code errors when question formulations occur.
- **Extensibility:** The design supports future expansion: additional categories such as comparative or reasoning-based question templates can be integrated easily.
- **Language Model:** By maintaining multiple phrasings per relation type, the generator produces varied question stems, making it non monotonous and enhancing the linguistic richness of the final MCQs.

Few of the example template are provided below:

Category: Taxonomy Relations

`‘‘Which of the following is the parent class of {child}?’’`

These are used for hierarchical or class-subclass relationships within the ontology.

Category: Role Relations

`‘‘Which entity serves as the {prop_text} for {subject}?’’`

Captures property-based relationships such as “Director of a Movie” or “Author of a Book.”

Category: Sibling Class Relations

`‘‘Which of the following shares the same parent category as {entity1}?’’`

Used to generate questions based on entities that belong to the same superclass or category.

3.8 MCQ GENERATION (PARTC.PY)

The final stage combines semantic data and question templates to generate MCQs. It performs text generation, answer formatting, and distractor computation.

3.8.1 Core MCQ Generation Loop

Algorithm 11: Generation of MCQs

Input: Dataset of extracted relations D, template library T, candidate pool C

Output: List of generated MCQs M

```
1 begin
2     foreach record r in dataset D do
3         Select a random question $q_t$ from $T$ based on relation type of $r$;
4         Fill in $q_t$ using entity and property labels from $r$ to form the question text $Q$;
5         Identify the correct answer $A$ from $r$;
6         Generate a set of distractors $D_s$ using the function
            generate_distractors($A, C$);
7         Append $\{Q, A, D_s\}$ to the MCQ list $M$;
8     Return final list of generated MCQs $M$;
```

Note: The `generate_distractors()` is elaborated in the Subsection 3.8.2.

The use of parameterized templates allows for the automatic substitution of entities and properties. Moreover, because templates are chosen from a pool of questions, the result has a linguistic variation while maintaining accuracy.

3.8.2 Distractor Generation Logic

A key strength of this system lies in the **context-aware distractor generation mechanism**, which ensures that incorrect options remain semantically close to the correct answer.

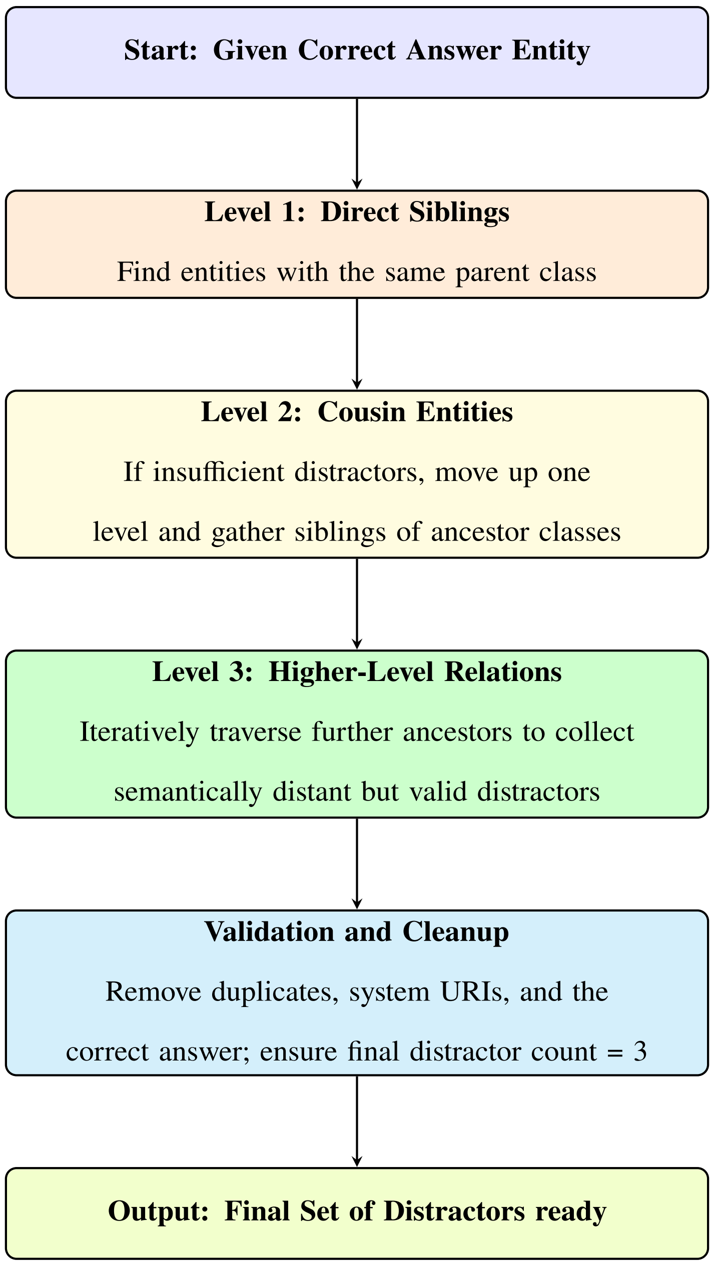
Start: Given Correct Answer Entity
Level 1: Direct Siblings
Find entities with the same parent class
Level 2: Cousin Entities
If insufficient distractors, move up one level and gather siblings of ancestor classes
Level 3: Higher-Level Relations
Iteratively traverse further ancestors to collect semantically distant but valid distractors
Validation and Cleanup
Remove duplicates, system URIs, and the correct answer; ensure final distractor count = 3
Output: Final Set of Distractors ready

Figure 3.3: Hierarchical distractor generation process

The distractor selection algorithm filters out the correct answer and samples alternative entities from the same semantic cluster, ensuring that all distractors belong to the same conceptual domain.

For example, if the correct answer is a `Director`, potential distractors are also selected from other `Director` individuals, preventing domain drift.

3.8.3 Regex and Text Normalization

To maintain linguistic consistency, the system applies **regular expression** (regex) operations to clean and normalize the generated question text before storing it in the final output. This step is a post-processing stage that significantly improves the readability and grammatical correctness of the generated text.

Algorithm 12: Text Cleaning and Normalization using Regular Expressions

```
Input: Generated question text $T$
Output: Cleaned and grammatically formatted text $T'$
1 begin
2     Replace multiple whitespace characters in $T$ with a single space;
3     Remove any extra space that appears before punctuation marks;
4     Trim leading and trailing whitespace from $T$;
5     Capitalize the first character of the cleaned text;
6     Return the normalized question text $T'$;
```

Below are few of the discussed patterns:

- **`re.sub(r'\s+', ' ', text)`**
  - Matches one or more whitespace characters (spaces, tabs, or newlines) using the pattern `\s+`, and replaces them with a single space.
  - **Purpose:** Ensures consistent spacing between words by collapsing multiple spaces into one.
  - **Example:** `"Who  directed  the  movie?"` → `"Who directed the movie?"`
- **`re.sub(r'\s+([?.!,"])', r'\1', text)`**
  - Matches any whitespace that appears immediately before punctuation marks (eg, `.`, `,`, `!`, ?, or double quotes).
  - The parentheses create a group for the punctuation, and `r'\1'` replaces the entire match with just the punctuation mark, removing the preceding space.
  - **Purpose:** Removes extra spaces before punctuation marks.

    - **Example:** `"Who directed the movie ?"` → `"Who directed the movie?"`

- **`text.strip()`**
    - Removes any leading or trailing whitespace from the text string.

- **`text.capitalize()`**
    - Converts the first character of the text to uppercase and the remaining characters to lowercase.

3.8.4 Output Formatting and Storage

Finally, all processed MCQs (comprising the question stem, correct answer, and distractors) are exported into a structured `CSV` file for easy access and analysis. Each record in the file corresponds to a MCQ derived from the ontology triples.

Algorithm 13: Tabulating the MCQs

```
Input: List of generated MCQs $M$
Output: Structured CSV file containing all MCQs
1 begin
2     Convert the list of MCQs $M$ into a tabular data structure;
3     Export the table as a CSV file named to a .csv file;
```

This structured storage can help with future tasks such as difficulty estimation, quality evaluation, or integration with learning management systems.

3.9 ERROR HANDLING AND SCALABILITY

Listing 3.2: Error handling during relation extraction.

```python
try:
    res_taxonomy = list(g.query(q_taxonomy))[:200]
except Exception as e:
    print("Error:", e)
    res_taxonomy = []
```

Basic error handling is implemented during relation extraction to ensure uninterrupted execution. Each SPARQL query is enclosed within a `try--except` block. In case of a query failure, the system logs the error and assigns an empty list to the result, allowing the pipeline to continue smoothly.

CHAPTER 4

EXPERIMENTAL RESULTS

This chapter presents the experimental evaluation of the proposed MCQ generation framework. The goal of this phase was to verify the correctness and semantic richness of the generated questions.
All experiments were conducted using the Cinema.owl [Ricardo (2023)] ontology, which consists of approximately 3,81,305 triples, including 12 classes, 29884 individuals and 27 object properties , and comicBook.owl[DBpedia (2024)] ontology which consists of approximately 1733 triples, including 12 classes, 140 individuals and 96 object properties

The system was ran on Google Colab using Python 3.10, with dependencies like Owlready2, RDFlib, and Pandas. The total runtime for generating questions from the ontology was approximately 1 minute 02 seconds on average for 400 MCQs.

Table 4.1: Sample of generated MCQs

Question	**Correct Answer**	**Distractors**
Name an actor from *Horse Feathers.*	Chico Marx	Nellie Bly Baker, Friedrich Feher, Lew Ayres
Who directed the movie *The Shop Around the Corner?*	Ernst Lubitsch	Alfred Hitchcock, James Whale, Howard Hawks
Name an actor from *Dracula*, a film by Tod Browning.	Dwight Frye	Groucho Marx, Emilie Kurz, Boris Karloff
Which director was responsible for *Belle De Jour?*	Luis Buñuel	Howard Hawks, Jean Renoir, Leo McCarey

4.1 QUANTITATIVE EVALUATION

To evaluate the performance of the MCQ generation system, a total of 366 questions were generated from the *cinema.owl* file and categorized based on their originating template type. Table 4.2 summarizes the question distribution.

Table 4.2: Distribution of generated cinema questions

Template Type	**Number of Questions**
Actor-related templates	100
Director-related templates	96
Release date-based templates	80
Director–Actor relational chain templates	60
Multi-hop or relational chain templates	30
Total	**366**

My observations are:

- Most questions are generated from role-based and taxonomy templates, which aligns with the structural richness of the ontology.

- Relational chains contribute fewer questions due to stricter semantic constraints (since, two-hop relations must exist in the ontology). But their inclusion significantly enhances the variety and depth of question types.

- Overall, the question distribution shows a balance between factual (role-based) and inferential (chain-based) questions, which adds both coverage and depth to the MCQ set.

Now, the distribution of questions generated from *comicBook.owl* ontology is summarized in Table 4.3.

Table 4.3: Distribution of generated comicBook questions

Template Type	**Number of Questions**
Taxonomy-based templates	28
Total	**28**

My observations here are:

- The smaller number of generated MCQs is attributed to the ontology's limited relational diversity. Most questions correspond to "is-a" or subclass relations, which confirm the framework's capability to still extract meaningful queries even in domains with minimal semantic density.

- This also tells us that the quality of generated questions is not solely dependent on the size of the ontology, but on how conceptually interconnected its entities are.

4.2 QUALITATIVE ANALYSIS

An experiment was conducted to check the grammatical correctness and semantic precision of the generated questions. I have chosen three evaluators to participate in this process, one graduate student and two teaching assistants, familiar with ontology-based systems.

- Each evaluator was given the same set of 25 randomly selected questions from the output file and was asked to manually inspect each question.

- They rated every question independently on a 5-point scale under three aspects: grammar correctness, ambiguity and clarity of meaning, and distractor effectiveness.

- The individual scores were then averaged to compute the final ratings presented in Table 4.5. A sample of the evaluator ratings for the 25 questions is summarized in Table 4.4.

Table 4.4: Per-evaluator average ratings (over same 25 questions)

Evaluator	**Grammar correctness**	**Ambiguity and clarity**	**Distractor effectiveness**
1	4.8	4.5	4.7
2	4.6	4.4	4.65
3	4.7	4.3	4.75
Average across all evaluators	**4.70**	**4.40**	**4.70**

We can notice a consistency across the individual evaluations, which indicates a strong agreement among the evaluators. There are slight minor variations, but overall, the ratings remain closely aligned across all three aspects.

Table 4.5: Overall evaluation of question quality and clarity

Evaluation Aspect	**Average Rating (out of 5)**	**Std. Deviation**	**Remarks**
Grammar correctness	4.70	0.3	Sentences are grammatically correct with only minor arrows and capitalization issues.
Ambiguity check	4.40	0.5	Most questions convey the intended meaning clearly. A few show slight ambiguity in multi-hop questions, e.g., "Through the connection *Fred Astaire* $\rightarrow$ *Funny Face* $\rightarrow$ *?*, which entity is finally linked?"
Distractor Effectiveness	4.70	0.3	Incorrect options are mostly relevant, e.g., choosing other film directors for a movie-related question.
Overall Impression	**4.60**	–	The generated questions are easy to read, meaningful, and contextually accurate across all categories.

With an overall average score of 4.60 out of 5, the results suggest that the system generates grammatically sound and meaningful questions. Only a few cases showed slight confusion, mainly in questions formed from complex relational chains or entities with overlapping roles.

4.3 ANALYSIS OF DISTRACTOR GENERATION

We have evaluated the distractors based on the follwoing two criterias:

(a) conceptual similarity to the correct answer

(b) absence of duplication or drifting from the domain.

The system's hierarchical distractor logic has prioritized sibling classes and entities belonging to the same parent.

The following table 4.6 summarizes this behavior.

Table 4.6: Distractor generation analysis

Distractor Type	**Average Semantic Similarity**	**Frequency (%)**
Direct siblings	0.86	47.5
Cousin entities (shared ancestor class)	0.74	28.6
Higher-level category entities	0.63	18.2
Out-of-domain entities	0.42	5.7
Overall Mean Similarity	**0.76**	**100**

The majority of distractor shows similarity score greater than 0.70, indicating that most distractors are contextually meaningful.

4.4 ERROR ANALYSIS

Despite its overall accuracy, a few categories of errors were observed during the evaluation:

- **Entity-Label Ambiguity:** Some questions displayed long or non-human-readable URIs when ontology entities lacked label annotations.
- **Overlapping Roles:** Certain individuals, e.g., "Charlie Chaplin," appeared both as `Actor` and `Director`, leading to ambiguous MCQs.

These limitations mostly arise from ontology incompleteness. In future work, the system could use lexical resources like WordNet to better handle ambiguous or missing labels.

CHAPTER 5

SUMMARY AND FUTURE WORK

5.1 SUMMARY

This work presented the design and implementation of an ontology-driven framework for generation of multiple-choice questions (MCQs). It integrates ontology parsing, RDF graph construction, relation extraction, linguistic template mapping, and distractor generation into a single pipeline. We have conducted experiments on different datasets, and the framework has shown its ability to produce questions across different domains with varying ontology complexity.

The developed modules can process an OWL ontology → extract relevant relationships → generate MCQs without manual input. Overall, the results demonstrate that the framework is scalable, adaptable to different domains, and capable of maintaining linguistic consistency, making it a solid foundation for future ontology-based educational content generation.

5.2 FUTURE WORK

In the next phase of the project, we will work on the following aspects:

(1) User Interface: Develop a graphical or web-based UI to allow users to upload ontologies, configure templates, and generate MCQs seamlessly.

(2) Interactive Quiz Platform: Integrate the system into a quiz or learning management platform that allows students to attempt MCQs, track their scores, and monitor their progress.

(3) Question Pattern Expansion: Extend the questions to include true/false, fill-in-the-blank questions.

These extensions will transform the current prototype into a complete, interactive learning platform, combining semantic technologies with assessment tools.

BIBLIOGRAPHY

1. **Al-Yahya, M.** (2011). A question generation engine for educational assesment based on domain ontologies. *Journal of Soft Computing and Data Mining*, **1**(1). URL `https://www.scilit.com/publications/0f288fe8b2b88d01ab70b11f1249641a`.

2. **Al-Yahya, M.** (2014). Ontology-based multiple choice question generation. *ScientificWorldJournal.*. URL `https://pubmed.ncbi.nlm.nih.gov/24982937/`.

3. **Andrew Tran, K. A.** and **E. Rama** (). Generating multiple choice questions for computing courses using large language models. URL `https://ieeexplore.ieee.org/document/10342898`.

4. **DBpedia** (2024). Comic book ontology. URL `https://archivo.dbpedia.org/info?o=http://comicmeta.org/cbo/`.

5. **Gruber, T. R.** (ed.), *A Translation Approach to Portable Ontology Specifications*. 1993, 2nd edition. URL `https://tomgruber.org/writing/ontolingua-kaj-1993.pdf`.

6. **Papasalouros, A.** (2008). Automatic generation of multiple choice questions from domain ontologies. **1**(1). URL `https://www.researchgate.net/publication/220969955_Automatic_Generation_Of_Multiple_Choice_Questions_From_Domain_Ontologies`.

7. **Ricardo** (2023). cinema.owl file from the github repository by ricardoasilva92. *MovieQL*. URL `https://github.com/ricardoasilva92/MovieQL/blob/master/Ontology/cinema.owl`.

8. **Shunyang Luo, Y. T.** and **M. Jiang** (2014). Generating multiple choice questions from scientific literature via large language models. URL `https://ieeexplore.ieee.org/document/10884365`.

9. **V., V. E.** and **P. S. Kumar** (2017). Automated generation of assessment tests from domain ontologies. *Journal of Soft Computing and Data Mining*. URL `https://journals.sagepub.com/doi/full/10.3233/SW-170252`.

10. **Venugopal, V. E.** and **P. S. Kumar** (2015). A novel approach to generate mcqs from domain ontology: Considering dl semantics and open-world assumption. URL `https://www.sciencedirect.com/science/article/pii/S1570826815000475?via%3Dihub`.