OM4OV: Leveraging Ontology Matching for Ontology Versioning

Zhangcheng Qiang*
Australian National University, Canberra, Australia

Kerry Taylor
Australian National University, Canberra, Australia

Weiqing Wang
Monash University, Melbourne, Australia

Abstract

Due to the dynamics of the Semantic Web, version control is necessary to manage changes in widely used ontologies. Despite the long-standing recognition of ontology versioning (OV) as a crucial component of efficient ontology management, many approaches treat OV as similar to ontology matching (OM) and directly reuse OM systems for OV tasks. In this study, we systematically analyse similarities and differences between OM and OV and formalise an OM4OV framework to offer more advanced OV support. The framework is implemented and evaluated in the state-of-the-art OM system Agent-OM. The experimental results indicate that OM systems can be effectively reused for OV tasks, but without the necessary extensions, can produce skewed measurements, poor performance in detecting update entities, and limited explanations of false mappings. To tackle these issues, we propose an optimisation method called the cross-reference (CR) mechanism, which builds on existing OM alignments to reduce the number of matching candidates and to improve overall OV performance.

2012 ACM Subject Classification Information systems → Information integration

Keywords and phrases ontology matching, ontology versioning

Digital Object Identifier 10.4230/TGDK.4.2.6

Category Research

Related Version *Full Version*: `https://arxiv.org/abs/2409.20302`

Supplementary Material For more information on Supplementary Materials and Additional Resources, including all third-party resources used, we refer to the list at the end of this article.
Software (OM4OV and Agent-OV): `https://github.com/qzc438/ontology-versioning`
Software (Brick Schema Version Track): `https://github.com/qzc438/brick-version-track`

Acknowledgements The authors thank reviewers for their comments that improved the manuscript. The authors thank Alice Richardson of the Statistical Support Network (Australian National University) for helpful advice on the statistical analysis in this paper. The authors thank Gabe Fierro at Colorado School of Mines for providing useful insights on Brick Schema version changes. The authors also thank the Commonwealth Scientific and Industrial Research Organisation (CSIRO) for supporting this project.

Received 2025-07-02 **Accepted** 2026-06-22 **Published** 2026-09-03

Editors Stefania Dumbrava, George Fletcher, Matteo Lissandrini, and Riccardo Tommasini

Special Issue Data Management for (Knowledge) Graphs

Generative AI Declaration During the preparation of this work, no Generative AI was used to produce the manuscript. However, Generative AI is intrinsic to the research reported here and is properly identified throughout the work.

* Corresponding author

1 Introduction

Ontologies serve as the backbone of the Semantic Web, providing formal concept descriptions of shared concepts across various applications [14]. An ontology is not static, and the need for version control arises. While web data is dynamic, any ontology in use must undergo periodic revisions to keep pace with the growth in domain knowledge, modifications to the application adaptation, or corrections to the shared conceptualisation [20]. For example, it is unrealistic to expect ontologies created in the 1990s to contain concepts such as "touchscreen", "fingerprint sensor", or "WiFi antenna" [15]. These modifications may cause undesirable deficiencies in artifacts that conform to or that reuse the ontology being changed, leading to severe non-compliance and incompatibility issues in downstream tasks.

Ontology versioning (OV), also known as ontology evolution, aims to distinguish and recognise changes between different ontology versions. This enables data that applies the ontology, ontologies that reuse the ontology, and software that uses the ontology to correctly respond to the ontology version changes [20]. Although OV plays an important role in ontology management, this area remains underexplored. Many studies [30, 29, 31, 8, 13, 41, 22, 23, 24, 34] treat OV as similar to ontology matching (OM) and assume OM systems can be reused for OV tasks. OM is a well-studied problem, so leveraging this deep research legacy for OV should be beneficial. We call this approach "OM4OV".

We analyse the complementary relationship between OM and OV tasks and provide a task formulation for the OM4OV framework. We implement the framework in our Agent-OM [37] system to validate its effectiveness in unifying the OM and OV tasks. We investigate the links between these two tasks in practice and explore optimisations for OM4OV. To the best of our knowledge, this work is the first to systematically analyse the OM4OV framework and to provide theoretical foundations and empirical evidence for its effectiveness and pitfalls. Specifically, our key contributions include:

- We analyse the similarities and differences between OM and OV. In the OM4OV framework, we describe four disjoint subset alignments produced in the OV process: *remain*, *update*, *add*, and *delete*, each corresponding to a subset of match or non-match in OM. We provide formal definitions and formulations.
- We extend Agent-OM with the OM4OV framework and call the extended system Agent-OV. We evaluate the performance of OV on the testbed derived from the Ontology Alignment Evaluation Initiative (OAEI) [32] datasets. We find that OM systems can be reused for OV tasks, but extensions are necessary to improve both overall OV performance and its meaningful quality measurement.
- We propose an optimised OM4OV framework. The new framework introduces a cross-reference mechanism, which builds on prior OM alignments to overcome the pitfalls of the naive OM4OV framework by reducing the number of matching candidates and filtering out ambiguous false mappings, thereby significantly improving overall OV performance.

In this paper, we assume the OWL 2 Web Ontology Language [5] is used to represent an ontology. We use both *concept* and *entity* to refer to OWL classes or properties. We use *subsumption* to mean `rdfs:subClassOf` or `rdfs:subPropertyOf` and *equivalence* to mean `owl:equivalentClass` or `owl:equivalentProperty` as formal class and property inter-relations.

The remainder of the paper is organised as follows. Section 2 reviews the related work. Section 3 provides the task formulation for the OM4OV framework, while Section 4 evaluates the OM4OV framework and discusses several common pitfalls for improvement. Section 5 proposes a cross-reference mechanism as a framework optimisation method. Section 6 discusses the use of OM4OV in real-world applications. Section 7 concludes the paper.

2 Related Work

Version control is recognised as a vital element in ontology management. Different versions need to support interoperability so that version changes do not impede the effective and sustainable use of the ontology.

One manual approach is to extend the ontology itself with internal version information. The Simple HTML Ontology Extensions [17] uses the tag `BACKWARD-COMPATIBLE-WITH` to record version information. The authors of [21] argue that a carefully-managed version numbering system embedded in the URI of the ontology (and therefore the fully expanded name of entities defined in the ontology) can minimise the impact of adopting updated versions because unchanged entities will be unaffected in practice. These approaches have been largely adopted by the later OWL languages, where a set of annotation properties related to version information is defined. These include `owl:versionInfo` and `owl:priorVersion` to describe the version number of the ontology, `owl:backwardCompatibleWith` and `owl:incompatibleWith` to specify an entity's compatible or incompatible corresponding entity in the previous version, and `owl:DeprecatedClass` and `owl:DeprecatedProperty` to declare deprecated entities. Later, the ontology language τOWL [42] was introduced to extend the OWL triple schema to a 4-tuple schema to represent the versioning of concepts within an ontology. This idea is now incorporated in the new proposals for RDF 1.2 Schema [38], which allow time-varying information to be deduced from a temporal dimension within the 4-tuple. An alternative option is to create a separate version log to track changes in versions. Unlike earlier approaches that use an unstructured plain text file, the authors of [35] propose a novel approach that uses a version ontology with a change definition language to create a version log. In [7], the authors construct an historical knowledge graph. Storing the version log in a knowledge graph not only avoids repetition, but also enables advanced search functions. The authors of [39] argue that version logs may contain redundancy and inconsistent information. They propose a graph-of-relevance approach to interlink different version logs and remove less relevant versions.

Maintaining version information in the ontology can be labour-intensive and error-prone. Either extending the current standard 3-tuple schema or using change logs requires consistent updating over time. In most cases, this process is hand-crafted by the ontology engineer or requires human intervention (e.g. pre-defining a schema or creating a template for the change log). A manual process is more likely to make mistakes and fail to propagate changes to dependent artifacts. Moreover, an ontology may lack or have incomplete version information. In such cases, current approaches have limited capability to detect incorrect or missing version information. These approaches rely on the version information contained in or attached to the ontology by manual methods. If such information is missing or incorrect, automated versioning of ontology concepts becomes necessary.

Reusing OM systems for OV tasks (OM4OV) is a lightweight and fully automatic approach to detecting changes in ontologies. OM4OV is compatible with ontologies that use different recording methods, including URIs or additional versioning triples, as well as ontologies that lack internal version information. The PromptDiff suite [30, 29, 31] is the canonical work that integrates different heuristic matchers and applies a matching-like comparison for OV. The concept of "matching" is later used in the H-Change methodology [8] to assess the assimilation of new concepts into the old ontology. Several studies [13, 41] note that updates to an ontology will generally create a requirement to update its existing alignments to external ontologies. Methods [22, 23, 24] are proposed to identify changes that affect alignments, and in some cases the framework is proposed to automate alignment updates [34], but these works do not address the discovery of alignments from an ontology to its own updated version.

While existing OM4OV approaches directly use OM systems for OV and focus on developing new algorithms and architectures for ontology version control, they pay less attention to understanding the underlying differences between OM and OV. OM is not exactly the same as OV. OM determines *matched* entities between two different ontologies, while OV can distinguish *add*, *delete*, *remain*, and *update* changes over different versions of one ontology. The OM input consists of two distinct ontologies, whereas the OV input is expected to consist of two versions of a single ontology. The output of OM is a set of entity mappings, whereas the output of OV comprises four sets, each defined by the nature of the change made. These differences may affect the adaptability of reusing OM systems for OV tasks.

3 Task Formulation

We now draw on concepts in OM to formalise OV tasks. Given an ontology O, we say that some entity $e \in O$ when e is of interest to ontology matching or ontology versioning, typically representing a concept of the domain of interest being modelled by O. We consider an entity to be the full URI of the reference in an ontology, or its equivalent abbreviation via a prefix, for example `cmt:ProgramCommitteeChair`. We consider an *entity name* to be the substring of an entity that provides a meaningful description of the entity (e.g. *ProgramCommitteeChair*), excluding any entity prefix or expanded URI (e.g. `cmt:ProgramCommitteeChair` is not an entity name).

3.1 OM Preliminaries

OM identifies correspondences between entities of two ontologies [11]. Formally, given a source ontology (O_s) and a target ontology (O_t), the OM task is to find an alignment A (i.e. a set of mappings) with respect to a given similarity threshold $s \in [0, 1]$, defined as [10]: $A = \{(e_1, e_2, r, c) \mid e_1 \in O_s, e_2 \in O_t, s \leq c \leq 1\}$, where e_1 and e_2 are ontology entities in O_s and O_t respectively, r is the relation between e_1 and e_2 that may be equivalence ($\equiv$) or subsumption ($\subseteq$), and c is the confidence. For example, `(cmt:ProgramCommitteeChair, confof:Chair_PC, ≡, 0.90)` means `cmt:ProgramCommitteeChair` and `confof:Chair_PC` are equivalent with a confidence of 0.90.

An entity match is asserted by a matching system according to its own internal algorithm, its own interpretation of "match", and its own assessment of confidence. In general for matching, a source entity may occur in multiple mappings that vary with respect to the target entity, the relation, and the confidence. In this study, we consider only alignments made by matching systems that are comprised of one-to-one equivalence mappings, admitting only matches with the highest c as determined by the matching system and requiring the matching system to resolve tied confidence scores. Typically, equivalence relations are determined by lexical or embedding similarity in matching systems, although logical entailment is equally admissible in our study. OM performance is measured by comparing the matching system alignment (A) with a reference alignment (R) using precision, recall, and F1 score (see Section 4.3 for details).

3.2 OM4OV Formalisation

OV identifies correspondences between two versions of a single ontology [20]. Given an old version of an ontology (O) and a new version of the same ontology (O'), the OV task can be considered as finding an alignment (A) from the old ontology to the new one. As illustrated in Figure 1, while an alignment for OM only mentions matched entities, OV needs to recognise both matched and non-matched entities. Furthermore, OV matched entities are comprised of two subsets: *remain* and *update*, while non-matched entities are comprised of *add* and *delete* entities. While some of these sets are comprised of entities, and others of mappings, when convenient henceforward, we will loosely refer to all of these as either alignments or sets of entities interchangeably.

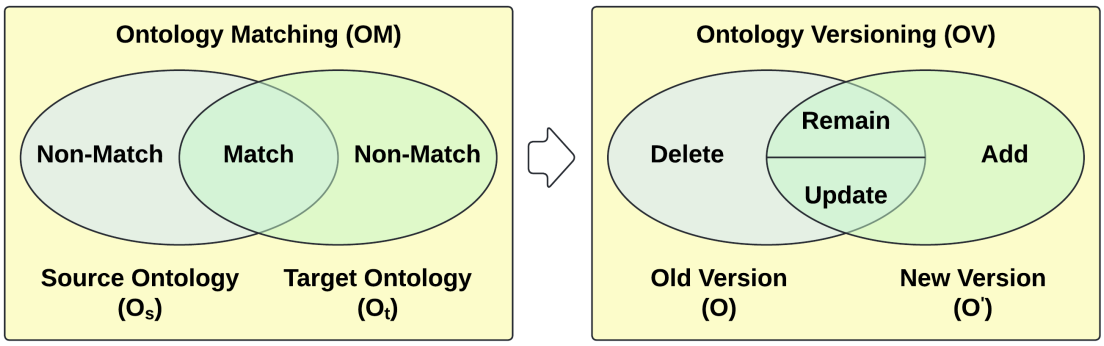


Figure 1 Overview of the OM4OV framework. The sets are comprised of entities, with the matched intersection containing entities that are considered equivalent by an ontology alignment.

An OV task can be formalised as finding an alignment and a non-alignment, A_{match} and $A_{non-match}$ respectively, between different ontology versions O and O' with respect to a given similarity threshold $s \in [0, 1]$. OV focuses only on the equivalence relation; subsumption mappings are beyond the scope of versioning.

▶ **Definition 1.** *Given ontologies O and O' and similarity threshold $s \in [0, 1]$, the OV task is to find A_{match} and $A_{non-match}$ as follows:*

$$\begin{aligned} A_{match} = \{(e_1, e_2, c) \mid e_1 \in O, e_2 \in O', s \leq c \leq 1 \\ \text{and there is no } (e_1, e_i, c') \text{ with } e_i \neq e_2 \text{ or with } c \neq c' \\ \text{and there is no } (e_j, e_2, c') \text{ with } e_j \neq e_1 \text{ or with } c \neq c'\} \\ A_{non-match} = \{e \mid e \in O \cup O' \text{ and there is no } (e_1, e_2, c) \in A_{match} \text{ with } e \in \{e_1, e_2\}\} \end{aligned} \tag{1}$$

We say that $e1$ and $e2$ are equivalent whenever there is a c satisfying $(e1, e2, c) \in A_{match}$.

For convenience, we will sometimes refer to the elements of A_{match} as *entities* rather than *mappings*, by which we mean entities that are mentioned in mappings in A_{match}.

The *update* and *remain* entities are entities successfully matched over different ontology versions. The *update* entities are, in general, the closest near-equivalent matches of entities from different ontology versions. We recommend that matching systems assign lower confidence to entity matches when either (a) structural changes have been made near the entity, such as changes to class membership or property restrictions, or (b) the entity's name has changed.

We recommend that all other successful matches not satisfying either (a) or (b) are assigned very high confidence by matching systems, most especially when a so-called *trivial* match is found. These matches are then interpreted as *remain* entities for OV. In particular, we say that an entity is interpreted as not changed in OV for any successful match where the matching system reports a confidence of 1. Further, we recognise that some OV systems may prefer, additionally, to interpret an entity as not changed when the confidence is only slightly less than 1.

▶ **Definition 2.** *$A_{remain}(A\odot)$ and $A_{update}(A\otimes)$ are defined as:*[1]

$$\begin{aligned} A\odot &= \{(e_1, e_2, c) \mid (e_1, e_2, c) \in A_{match} \text{ and } c = 1\} \\ A\otimes &= \{(e_1, e_2, c) \mid (e_1, e_2, c) \in A_{match} \text{ and } c < 1\} \end{aligned} \tag{2}$$

[1] In some cases, OV systems may consider matches that are near-enough with a confidence slightly less than 1 to be perfect matches. Without loss of generality, we assume in such cases that the OV system reassigns c to be 1.

We can see that $A_{match} = A \odot \cup A\otimes$. By definition, $A\odot$ and $A\otimes$ are disjoint.

The *add* and *delete* entities are actually non-matched entities between different ontology versions. The *add* entities are the entities in O' that have no matches in O. Similarly, we can interpret the *delete* entities as entities in O that have no matches in O'.

▶ **Definition 3.** *$A_{add}(A\oplus)$ and $A_{delete}(A\ominus)$ are defined as:*

$$\begin{aligned} A\oplus &= \{e \mid e \in O' \text{ and } e \in A_{non-match}\} \\ A\ominus &= \{e \mid e \in O \text{ and } e \in A_{non-match}\} \end{aligned} \tag{3}$$

We can see that $A_{non-match} = A \oplus \cup A\ominus$. By definition, $A\oplus$ and $A\ominus$ are disjoint.

4 OM4OV Evaluation

We now consider the evaluation of OM4OV. In the literature, OV are evaluated using OM measures, which we refine here to give better insights into OV performance.

4.1 Dataset Construction

There is a dearth of benchmark datasets for evaluating OV. The Ontology Alignment Evaluation Initiative (OAEI) contains several datasets related to OM tasks, but none are specifically designed for OV tasks. We propose an approach to constructing synthetic OV datasets from real-world OM datasets. Figure 2 illustrates the generation of OM4OV datasets.

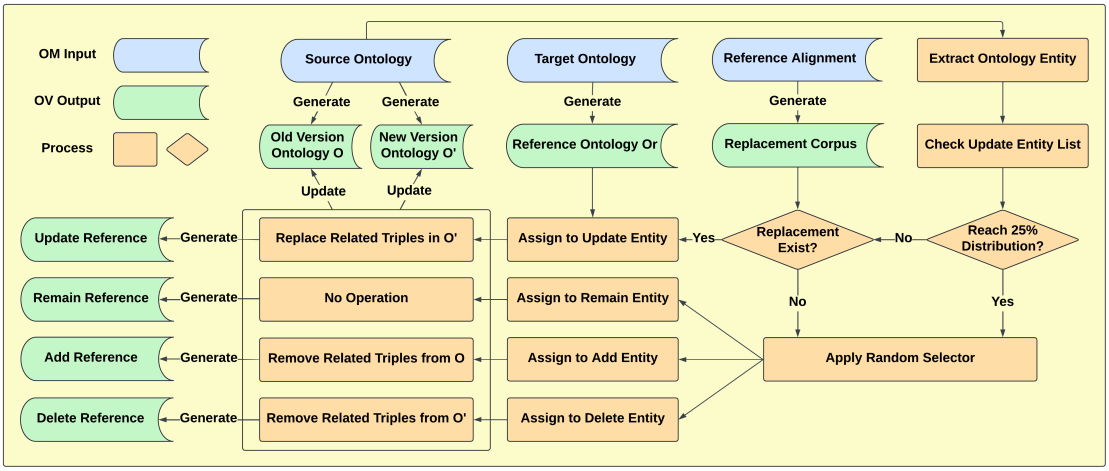


Figure 2 Generation of OM4OV datasets from initial OM inputs of a source ontology, a target ontology, and a reference alignment, and outputs of versioning ontologies and reference alignments with four categories. Optionally, a reference ontology and a replacement corpus may be produced.

(1) The original OAEI datasets for OM provide two ontologies, the *source* O_s and the *target* O_t. We choose either one as the ontology for versioning and duplicate it into O and O'. Another ontology will be used later to provide references for synthetic *update* entities. We duplicate this ontology into O_r.

(2) There are four possible entity changes in OV tasks: *remain*, *update*, *add*, and *delete*. We retrieve all ontology entities from O or O' (since they are identical at this stage). Each ontology entity will be randomly assigned to one of these categories, and the four sets are

pairwise disjoint by construction. In real-world OV, entity updates are relatively infrequent. We allow a user-defined parameter to fix the proportion of update entities in the synthetic dataset, defaulting to 25%. The update entities are randomly selected from the replacement corpus, so the number of update entities is limited by the size of the replacement corpus, which may be less than the user-defined parameter value. For this reason, selection of update entities is prioritised over other categories, and excess entities are redistributed evenly over the other three categories.

(3) For each ontology entity e assigned to a category, the corresponding dataset operations are described as follows. While there is no operation for *remain* entities, entities assigned to *add* and *delete* are deleted from all triples that mention them in O or O' respectively. Each *update* mapping requires that all occurrences of its domain entity in triples in O' are replaced by the corresponding co-domain entity of the mapping. We generate four corresponding versioning references based on the entity assignments. These files provide the ground truth for evaluation.

To ensure the replaced entity is reasonable and reflects real-world facts, we use equivalent classes or properties specified in the reference alignment. For example, we could replace `cmt:ProgramCommitteeChair` with its equivalent class `confof:Chair_PC`. This operation is cascaded, meaning that we update not only the entity name but also its related triples, such as changing the semantic information from `(cmt:ProgramCommitteeChair, rdfs:subClassOf, cmt:ProgramCommitteeMember)` to `(confof:Chair_PC, rdfs:subClassOf, confof:Person)`. While this procedure can be unlimited, we restrict it to extract only the triples directly related to the entity (i.e. a 1-hop subgraph) in the ontology O_r. For entities whose names are unique identifiers or codes (and not textually meaningful names), we retrieve their meaningful descriptions from annotation properties. For example, we extract an additional annotation triple `(mouse:MA_0000270, rdfs:label, "eyelid tarsus")` for the entity `mouse:MA_0000270`. After extraction, we also need to update the prefix from *"confof"* to *"cmt"* where the new entity URI will be `cmt:Chair_PC` and the replacement triple will be `(cmt:Chair_PC, rdfs:subClassOf, cmt:Person)`. Finally, we add these new triples to the existing ontology O'. Note that where entity names do not change but structural changes have been made near the entity, this will be represented as an entity update. Table 1 summarises the steps to construct an *update* entity e in the benchmark dataset.

Table 1 Steps to construct an *update* entity e in the benchmark dataset.

Step	Dataset Operation
#1	Remove all the triples related to e in O'.
#2	Find the equivalent class/property e_r in *replacement corpus*.
#3	Extract all the triples related to e_r in O_r.
	(Optional) Extract annotation triples related to e_r in O_r.
#4	Update the prefix from e_r to e and create a new entity e'.
#5	Add all the triples related to e' to O'.

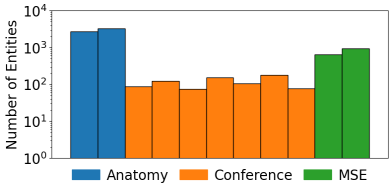
Unlike the original OAEI datasets used for OM, we introduce randomness to ensure that the synthetic OAEI datasets for OV are different each time they are constructed. This suits the dynamic nature of OV, where the changes vary between different versions. For this reason, we consider the new OAEI datasets for OV more like a testbed, as they can simulate a variety of situations for OV tasks. When reproducibility is required, we recommend using a fixed seed to control the randomness.

Table 2 details the OAEI tracks selected for the OV testbed. The current version of the OV testbed contains a total of 11 distinct ontologies from three different OAEI tracks. The anatomy track contains two large ontologies, while the MSE track has two medium ontologies, and the conference track has 7 small ontologies. We exclude EMMO [9] from the MSE track because it assigns opaque symbolic codes to entity names, using a naming convention that differs from MaterialInformation [3] and MatOnto [19], which use meaningful entity names. Figure 3 shows the number of entities in each ontology in each track.

Table 2 Selected OAEI tracks for the OV testbed.

Track	Domain	# of Ontologies
Anatomy	Human and Mouse Anatomy	2
Conference	Research Conference	7
MSE	Materials Science and Engineering	2

Figure 3 Number of entities.



4.2 Evaluation System

We choose the matching system Agent-OM and its extended version Agent-OV to demonstrate the OM4OV framework. Agent-OM is one of the flagship OM systems powered by large language models (LLMs) and multi-agent architectures. Its foundation framework is designed for traditional OM tasks. We extend Agent-OM with the OM4OV framework so that it can be used for OV tasks. Inherited from Agent-OM, Agent-OV supports a wide range of LLMs, including commercial API-accessed LLMs OpenAI GPT [33] and Anthropic Claude [2], as well as open-source LLMs Meta Llama [28], Alibaba Qwen [1], Google Gemma [12], and ChatGLM [40]. Inherited from Agent-OM, *similarity_threshold* and *top@k* are the adjustable key hyperparameters for Agent-OV.

As an aside, we observe that confidence scores for successful matches as determined by OM systems may be unhelpful for direct use in OV for distinguishing *remain* entities from *update* entities. This is because, in practice, the confidence boundary to best distinguish *remain* and *update* entities varies in different domains and datasets. In Agent-OV, we handle this problem in the following way. A matched entity is determined to be a *remain* entity when its syntactic name and lexical descriptions are unchanged. Otherwise it is determined to be an *update* entity. We observe that this approach means that Agent-OV usually considers entities to be *remain* entities when they have structural changes but conserve syntactic names and lexical descriptions. This approach will tend to consider ambiguous entities as *remain* in preference to *update*, so tending to trade off performance in *remain* versus performance in *update*. While other OV systems may handle this differently, we propose that all OM4OV systems could benefit from this approach to partitioning successful matches into *remain* entities and *update* entities.

4.3 Evaluation Criteria

OM typically measures performance using precision, recall, and the F1 score. Given a gold standard reference (R) and a system-discovered alignment (A), precision measures matching correctness and recall measures matching completeness, while F1 score offers a harmonic mean to balance correctness and completeness. OV can reuse these measures, but they need to be extended into four sub-measures for add, delete, remain, and update performance. While an OV method will trade off performance over each of these sets, an OV application problem may well prioritise accuracy on some over others.

$$Precision = \frac{|A \cap R|}{|A|} \quad Recall = \frac{|A \cap R|}{|R|} \quad F_1\ Score = \frac{2}{Precision^{-1} + Recall^{-1}} \tag{4}$$

4.4 Results

We choose to use the Meta open-source model llama-3-8b [27] for our experiments in this paper, eschewing the higher-end models that offer small improvements that may not justify the financial investment. We use embeddings from the same LLM model to further avoid expensive API calls and thereby provide an entirely free-to-use version of Agent-OV. The hyperparameter settings are $similarity\_threshold = 0.90$ and $top@k = 3$ across all the system alignments generated. We set $top@k = 3$ based on empirical experience that 3-5 works well for Agent-OM. We experiment with the alternative values of the similarity threshold later in this paper. The fixed seed for the OV testbed is set to 42. Due to the non-determinism of LLMs, repeated experiments can produce insignificantly different results.

Figure 4 shows the performance of Agent-OM and Agent-OV on the OAEI Conference Track. The results indicate that it is possible to unify the OM and OV tasks using the OM4OV framework. Both Agent-OM and Agent-OV can produce alignments with precision, recall, and F1 scores. However, we can see that Agent-OM consistently performs surprisingly well across all alignments. This is because the traditional OM system captures only matched entities between two ontologies, which in the OV task correspond to the *remain* and *update* entities. Because *remain* entities typically comprise the major proportion of entities over two different versions, the correctness of *remain* entities dominates the overall evaluation and other categories are de-emphasised. We believe this is a common situation that leads to misleading performance measurement when using an OM system in OV tasks.

Agent-OV overcomes the skewed measure in Agent-OM. By decomposing the measure into four sub-measures, we can observe more informative matching performance across different OV tasks. In general, we observe that the matching performance is highest in *remain*, followed by *add* and *delete*, and relatively low in *update*. This trend is consistent across different tracks and ontologies. The measures for *remain* are typically very close to 100%. The measures for *add* and *delete* are generally good. Our system uses LLMs as the backend, and LLMs generally have strong background knowledge to detect non-matched entities. The measures for *update* show scope for improvement. This is because finding non-trivial alignments and appropriate similarity thresholds is not easy. We examine the false mappings produced by Agent-OV. Unlike OM tasks, we find that some of these identified false mappings are not actually false. Several examples are listed below with explanations.

(1) False mappings in OV can result from flawed ontology design choices, as shown here.

```
(edas:ConferenceEvent, edas:Event, c)
(edas:ConferenceEvent, edas:AcademicEvent, c)
```

In the example, the reference shows `edas:ConferenceEvent` is updated to `edas:Event`, but the OV system predicts `edas:ConferenceEvent` is updated to `edas:AcademicEvent`. An event can be a non-academic event (e.g. a social event) or an academic event (e.g. a seminar). In the context of a research conference, a conference event is more likely to mean the latter. However, mapping to event still makes sense. Within one ontology, the meaning of two entities is too close to be distinguished and therefore leads to them being synonyms for each other. Ideally, we should avoid this type of ontology design. This example also demonstrates a unique benefit of using OM for OV, which could potentially assist in ontology design.

(2) False mappings in OV can create ambiguous "equivalent" relationships, as shown here.

```
(confof:dealsWith, confof:hasSubjectArea, c)
(confof:dealsWith, confof:reviewes, c)
```

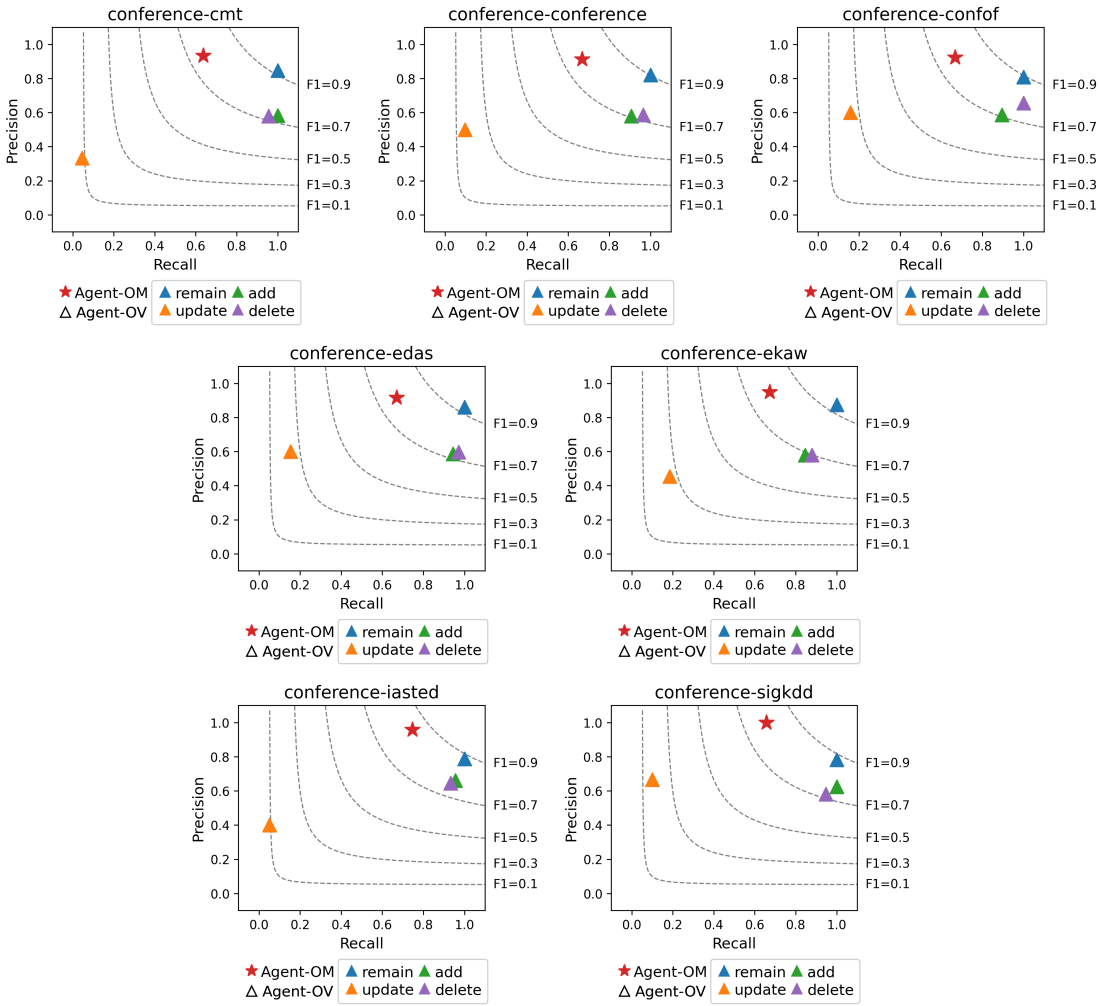


Figure 4 Evaluation of OAEI Conference Track on the OV testbed. Agent-OM, shown as a red star, determines only the single category of entity matches. The extended Agent-OV, shown as triangles, determines four different categories of entity matches (*remain* or *update*) and non-matches (*add* or *delete*).

In the example, the reference shows `confof:dealsWith` is updated to `confof:hasSubjectArea`, but the OV system predicts `confof:dealsWith` is updated to `confof:reviewes`. This is not necessarily wrong, but it interprets "deal with" differently. It is vital to notice that the term "equivalent" in OV is weaker than that in OM. OV allows for roughly "equivalent" mappings. The entities mapped in OV can slightly alter their meanings in response to real-world changes in the domain or evolution of natural language.

(3) False mappings in OV can arise from different similarity thresholds, as shown here.

```
(sigkdd:Start_of_conference, None, 0.90)
(sigkdd:Start_of_conference, sigkdd:startDate, 0.80)
```

In the example, if $similarity\_threshold = 0.90$, `sigkdd:Start_of_conference` in O will be assigned to *delete* entities as it does not have matching entities; if $similarity\_threshold = 0.80$, `sigkdd:Start_of_conference` in O will be assigned to *update* entities as it has a matching entity `sigkdd:startDate` in O'. Both results are valid because defining the boundary between

matching and non-matching is context- and application-dependent. For example, the similarity threshold could be relatively higher in the biomedical domain to ensure that each term is unique, whereas the similarity threshold in the conference domain could be relatively lower to improve the interoperability of terminologies used in research conferences.

We extend our analysis to other OAEI tracks and varying similarity thresholds. Figure 5 shows that the trends are consistent with other medium and large OAEI tracks evaluated on the OV testbed. Additionally, we observe a longer computation time in the Anatomy Track. Although Agent-OV has an optimisation module for the matching candidate selection process (inherited from Agent-OM), it is still insufficient for some OV tasks. There is a need to optimise the framework for OV in large-scale ontologies. Figure 6 studies the effect of varying similarity thresholds for the Mouse ontology in Agent-OV. We change the similarity threshold from 1.00 to 0.80 to evaluate its effect on matching performance. While OV entities are classified into four different categories, each category requires a corresponding sub-measurement. Similar to OM tasks, changes in hyperparameter settings in OV tasks lead to a trade-off between precision and recall. Moreover, the hyperparameter settings also influence the sub-measures. We find that the similarity threshold has no effect on detecting *remain* entities, but significantly affects the detection of *update* entities and slightly affects the detection of *add* and *delete* entities. Detecting *update* entities and detecting *add* and *delete* entities are negatively correlated, as is expected because a near-miss *update* alignment will fall back to a pair of *non-match* entities, one from each ontology. Therefore, lower similarity thresholds can result in more *update* entities being detected, while higher similarity thresholds may find more *add* and *delete* entities.

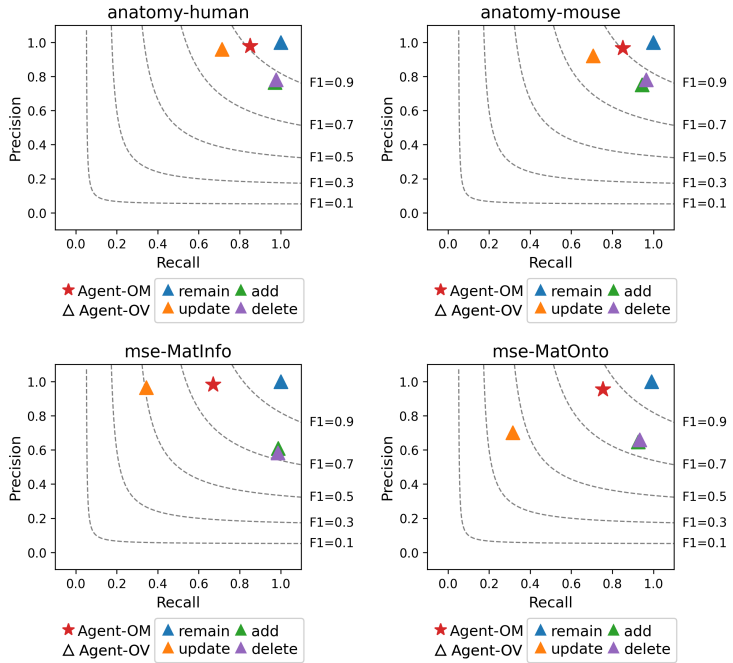


Figure 5 Evaluation of OAEI Anatomy and MSE Tracks on the OV testbed has shown similar trends.

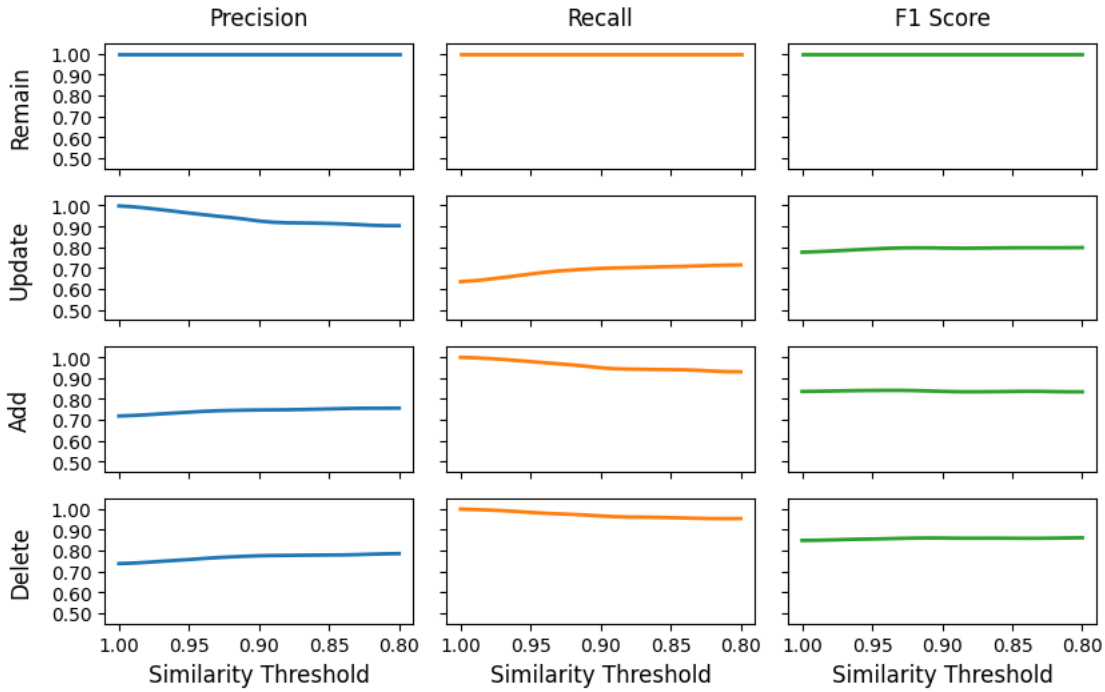


Figure 6 Evaluation varying similarity thresholds in Agent-OV. We apply a 1-D Gaussian filter for smoothing. The standard deviation for a Gaussian kernel is 1 (i.e. $sigma = 1$). We can see that the similarity threshold has no effect on detecting *remain* entities in the top row, but affects the detection of *update*, *add*, and *delete* entities.

4.5 Discussion

The experimental results have shown that OM systems can be reused for OV tasks, but they have limitations and require necessary extensions. The following key points need to be taken into account when reusing OM systems for OV tasks.

(1) **Skewed Measurement:** OM systems exhibit skewed measurement of OV tasks because unchanged *remain* entities dominate in practice. OM measures are reusable, but these measures need to be extended into four sub-measures for *add*, *delete*, *remain*, and *update* performance. Within each sub-measure, the inherent precision-recall trade-off still holds. Across different sub-measures, they are not independent. This is because each change in a part of an alignment in one category will cause other categories to change accordingly. For example, if a new mapping is found in $A\odot$ or $A\otimes$, then the number of mappings in $A\oplus$ and $A\ominus$ will be reduced by one each, and vice versa.

(2) **Update Pitfalls:** OM systems are sensitive to changes in similarity thresholds and backend LLMs. For OV tasks, only *update* entities are sensitive to these changes, while *remain*, *add*, and *delete* are relatively stable across these hyperparameter settings. Poor *update* performance is common, while tuning the similarity threshold trades precision for recall without resolving the underlying difficulty.

(3) **Disputed False Mappings:** OM systems used for OV tasks can produce mappings that are false with respect to the ground truth reference, but that are true with respect to expert review. This can be caused by flawed ontology design choices, ambiguous "equivalent" relationships, or inappropriate settings of the similarity threshold. This is difficult for OM systems to recognise and generally requires human effort.

5 OM4OV Optimisation

Having established the current pitfalls of OM4OV, we explore optimisations for the framework. Often, ontology creators provide cross-references to alternative ontologies to enhance interoperability and facilitate integration. For example, a cross-reference between the Mouse ontology and the Human ontology is provided in the OAEI Anatomy Track. Reusing the cross-references developed for OM tasks, we propose a novel mechanism to reduce the number of matching candidates and also to improve overall OV performance.

Figure 7 illustrates the cross-reference (CR) mechanism used in the OM4OV framework. We can see that, without using the cross-reference O_r, the matching candidates cover the range of $O \cup O'$. This number can be significantly reduced by removing prior matches (i.e. $O \cap O_r \cap O'$) and non-matches (i.e. $O \cap O_r \setminus O'$ and $O' \cap O_r \setminus O$). The prior matching will be incorporated into the final alignment, while the known non-matching regions will be removed entirely in the subsequent OV process. The OV process then only determines the posterior alignment.

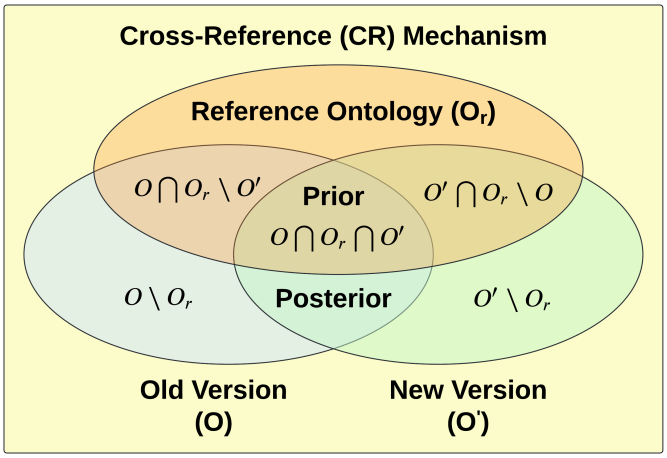


Figure 7 Illustration of the cross-reference (CR) mechanism.

In practice, the prior alignment usually contains a large number of *remain* entities and a small number of *update* entities. Matching performance is also enhanced by utilising these established mappings. Since prior matches are inferred from OM references validated by domain experts, they provide a solid ground truth for alignment within a specific domain. Removing known non-matches from the OV process also simplifies posterior alignment detection. The optimised OM4OV framework using the CR mechanism is defined as follows.

▶ **Definition 4.** *Given a reference ontology (O_r), an old version of an ontology (O) and a new version of the same ontology (O'), two cross-references between O and O_r (R_{or}) and between O' and O_r ($R_{o'r}$) are defined respectively as:*

$$\begin{aligned} R_{or} &= \{(e_1, e_3, c) \mid e_1 \in O, e_3 \in O_r, s \leq c \leq 1 \text{ and there is no } (e_1, e_3, c') \text{ with } c' > c\} \\ R_{o'r} &= \{(e_2, e_3, c) \mid e_2 \in O', e_3 \in O_r, s \leq c \leq 1 \text{ and there is no } (e_2, e_3, c') \text{ with } c' > c\} \end{aligned} \tag{5}$$

We can use R_{or} and $R_{o'r}$ to infer some known mappings between O and O' before performing OV. We call these mappings a *prior* alignment (A_π). After subsequently performing OV, we have our *posterior* alignment (A_{π^*}). In this setting, A_{match} in OV can be decomposed into two parts: $A_{match} = A_\pi \cup A_{\pi^*}$. A_π can be directly inferred from the two cross-references R_{or} and $R_{o'r}$. By assuming the equivalence relation as extracted from the two cross-references is transitive, we have

that $e_1 \in O$, $e_2 \in O'$, and $e_3 \in O_r$, if $e_1 \equiv e_3$ and $e_2 \equiv e_3$ then $e_1 \equiv e_2$. A_{π^*} aims to detect missing mappings from the cross-reference. None of these mappings would come from any pairwise intersection of O, O_r, and O' because $O \cap O_r \setminus O'$ and $O' \cap O_r \setminus O$ are pre-defined as non-matched entities, and the matched entities in $O \cap O_r \cap O'$ have already been captured in the $A(\pi)$. As a result, A_{π^*} can be defined within a smaller scope.

▶ **Definition 5.** *Prior alignment (A_π) and posterior alignment (A_{π^*}) are defined as:*

$$\begin{aligned} A_\pi &= R_{or} \cap R_{o'r} = \{(e_1, e_2, c) \mid e_1, e_2 \in O \cap O_r \cap O', s \leq c \leq 1\} \\ A_{\pi^*} &= \{(e_1, e_2, c) \mid e_1 \in O \setminus O_r, e_2 \in O' \setminus O_r, s \leq c \leq 1\} \end{aligned} \tag{6}$$

An ontology can have multiple cross-references available. In such cases, the *prior* reference becomes the union of all known cross-references (R_{or1} ... R_{orn}), and the ontology used in the *posterior* alignment (O_{ra}) becomes the union of all reference ontologies (O_{r1} ... O_{rn}).

▶ **Definition 6.** *A_π and A_{π^*} in multiple cross-references can be formulated as:*

$$\begin{aligned} A_\pi &= (R_{or1} \cap R_{o'r1}) \cup (R_{or2} \cap R_{o'r2}) \cup ... \cup (R_{orn} \cap R_{o'rn}) \\ A_{\pi^*} &= \{(e_1, e_2, c) \mid e_1 \in O \setminus O_{ra}, e_2 \in O' \setminus O_{ra}, c \geq s\} \text{ where } O_{ra} = O_{r1} \cup O_{r2} \cup ...O_{rn} \end{aligned} \tag{7}$$

Figure 8 compares the OV performance with and without the CR mechanism on the same OV testbed that we analysed in Section 4. We can see that the CR mechanism significantly enhances Agent-OV's ability to detect *update* entities and slightly improves its performance in detecting *add* and *delete* entities, with no effect on *remain* entities. More specifically, performance improvement is mainly attributed to recall for *update* entities, whereas it is more evident in precision for *add* and *delete* entities. Figure 9 compares the effects of different similarity thresholds for the Mouse ontology versioning. The results of the experiment show that the CR mechanism is less sensitive to the similarity threshold than the original OM4OV framework in Figure 6, suggesting that it can be helpful in scenarios where the optimal similarity threshold is unclear or difficult to determine.

6 OM4OV in Real-World Applications

Brick Schema [4] is a popular ontology used in the building industry. Introduced in 2016, Brick Schema has undergone several major version changes, including URI changes, schema language changes, and ontology harmonisation (combining Brick Schema with RealEstateCore [16] and Project Haystack [25]). We use the change logs from the official GitHub repository to provide near-ground-truth alignments for evaluation. We are not confident about the reliability of the manually-generated logs as we observe instances where the log entry for a change conflicts with the difference observed in the ontology itself. We demonstrate versioning performance over three distinct major version updates: from v1.1.0 to v1.4.0, v1.2.0 to v1.4.0, and v1.3.0 to v1.4.0. We have collapsed multiple human-identified minor version changes into the corresponding major version changes for our experiments to consider more complex version changes than are evident in successive minor versions. We filter out classes and properties that are used in Brick Schema conventions but that user experts determined to be unimportant for version tracking and hence have not been captured in change logs. For example, "Tag", "NodeShape", and "PropertyShape" are excluded from the matching candidates.

Figure 10 shows that our OM4OV implemented in the OM system is more precise than our previous OM-only system in tracking changes across different versions of Brick Schema. Similar trends are observed in skewed measurements, update pitfalls, and disputed false mappings. We are unable to empirically evaluate our cross-reference mechanism with Brick Schema. Unfortunately,

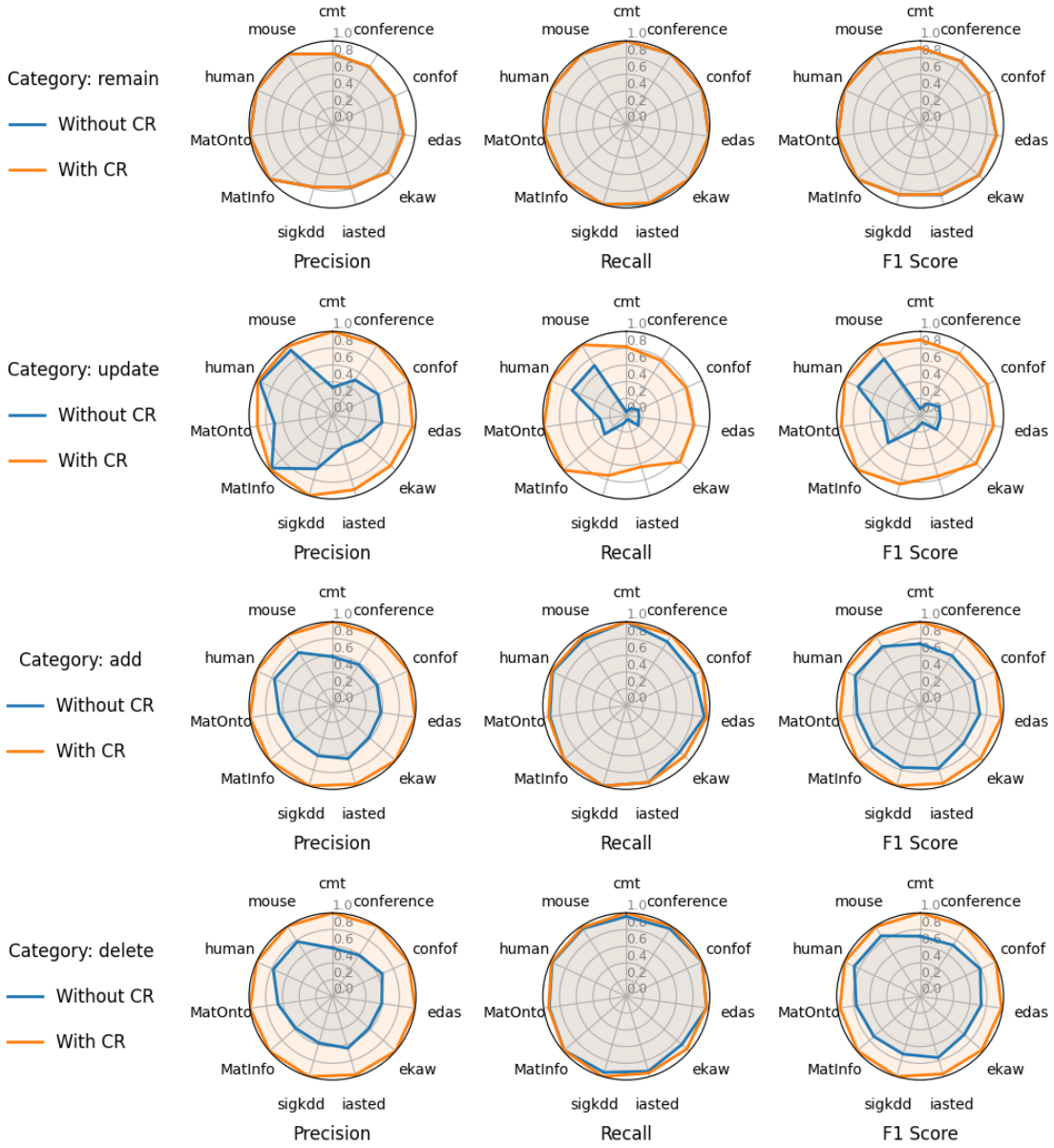


Figure 8 Evaluation of the cross-reference (CR) mechanism on the OV testbed. Each radar chart shows measurements (precision, recall, and F1 score) for a change category over 11 datasets analysed around the perimeter, both with (*orange*) and without (*blue*) the CR mechanism applied.

while Brick Schema offers high-level alignments with RealEstateCore and Project Haystack, these alignments are highly incomplete, addressing only few classes and properties. Further, they do not indicate versions of the ontologies to which they apply. We have been unable to locate an alternative ontology that offers the data needed for evaluating our cross-reference mechanism. We hope our work contributes to the quality improvement of OV so that building high-quality versioned reference alignments becomes standard industry practice. At that time, automated system updates with respect to ontology version changes would become feasible, and we could evaluate our cross-reference mechanism on real-world data.

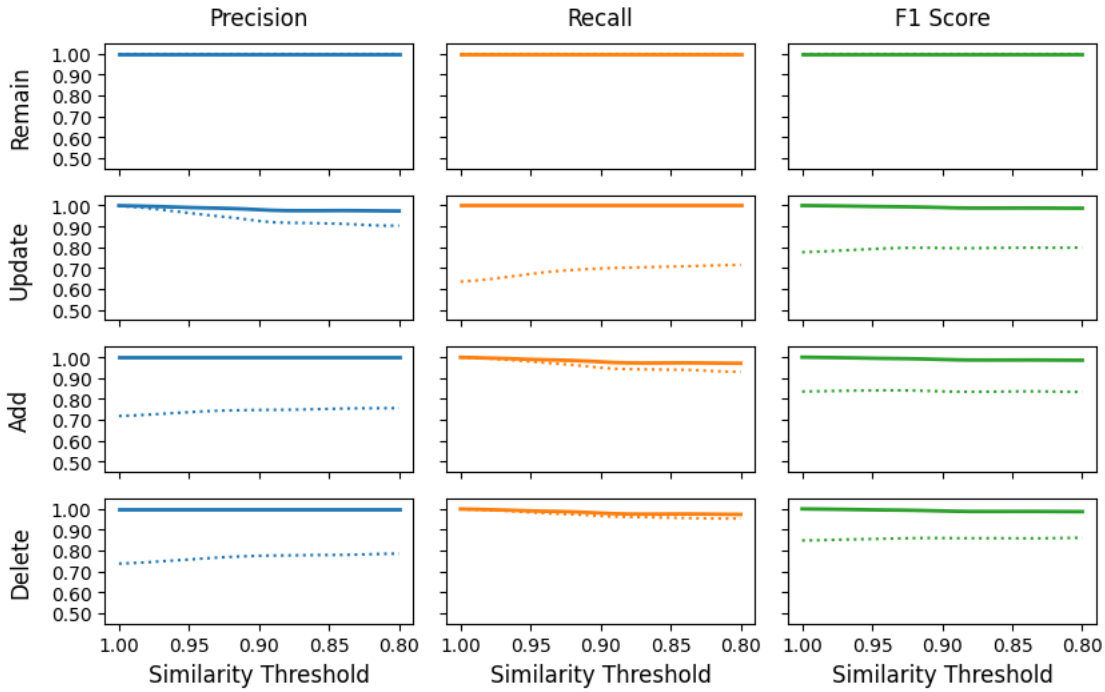


Figure 9 Evaluation of the cross-reference (CR) mechanism on different similarity thresholds. The solid lines contribute to OV performance with the CR mechanism, while the dotted lines contribute to OV performance without the CR mechanism. We apply a 1-D Gaussian filter for smoothing. The standard deviation for a Gaussian kernel is 1 (i.e. $sigma = 1$).

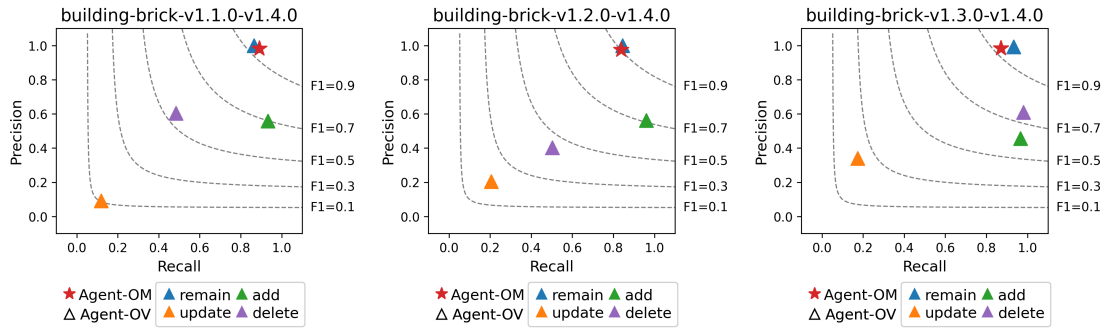


Figure 10 Evaluation of the OM4OV framework tracking Brick Schema version changes.

7 Conclusion

In this paper, we systematically analyse similarities and differences between OM and OV tasks and provide formal definitions and task formulations for OM4OV. The experimental results from Agent-OM and Agent-OV demonstrate that the OM4OV framework is workable, though it presents several limitations. To tackle these issues, we propose a novel optimisation method that uses a cross-reference (CR) mechanism to leverage reference alignments from OM for OV. This method (1) overcomes several pitfalls in using OM for OV tasks, (2) significantly reduces the number of matching candidates, and (3) improves overall OV performance. We showcase the use of our OM4OV framework in tracking version changes in a widely used ontology for the building industry.

We observe three opportunities for future work. (1) While OM has a universal evaluation measure, our proposed measures for OV have four sub-measures. We plan to investigate a merged formula for OV sub-measures in the future, for example, using a harmonic mean to combine sub-measures. (2) We limit the semantic analysis to noting changes in 1-hop subgraph relations that can be either ontology keywords (such as domain, range, and subclass) or domain concepts as object properties. A more complete analysis of structural evolution could employ description-logic-based reasoning for subsumption checking, consistency validation, and detection of semantically equivalent expressions with only syntactic differences. This kind of analysis has not been previously explored in OM and is not addressed in this study, but could be worthwhile further work. (3) We plan to apply the OM4OV framework in emerging domain ontologies that demonstrate regular changes over time. We are working with the Brick Schema development team to encourage adoption of our OM4OV framework.

Statement on Supplementary Materials and Additional Resources

The following supplementary materials belong to this paper:

(S1) *Software (OM4OV and Agent-OV):* `https://github.com/qzc438/ontology-versioning`
This artifact contains source code for OM4OV and Agent-OV, as well as the versioning datasets mentioned in Section 4.1.

(S2) *Software (Brick Schema Version Track):* `https://github.com/qzc438/brick-version-track`
This artifact contains the source code for tracking Brick Schema [4] version changes mentioned in Section 6.

The following resources were used in the mentioned parts of the paper:

(R1) *Software (Agent-OM) [36]:* `https://github.com/qzc438/ontology-llm`
This artifact contains the source code for Agent-OM [37] mentioned in Sections 4.2 and 4.4.

(R2) *Dataset (OAEI Datasets) [26]:* `https://dwslab.github.io/melt/track-repository`
This artifact shows the Matching EvaLuation Toolkit (MELT) [18] repository to download the original OAEI datasets for (S1) (retrieved April 1, 2025). According to the OAEI data policy, "OAEI results and datasets, are publicly available, but subject to a use policy similar to the one defined by NIST for TREC. These rules apply to anyone using these data." Please find more details from the official website: `https://oaei.ontologymatching.org/doc/oaei-deontology.2.html` (retrieved April 1, 2025).

(R3) *Dataset (Brick Schema Change Logs) [6]:* `https://github.com/BrickSchema/Brick/releases`
This artifact shows the link to the Brick Schema [4] official repository to download the change logs for (S2) (retrieved April 1, 2025).

References

1 Alibaba Qwen Team. Qwen Models, n.d. URL: `https://qwenlm.github.io`.

2 Anthropic. Claude Models, n.d. URL: `https://docs.anthropic.com/en/docs/about-claude/models`.

3 Toshihiro Ashino. Materials ontology: An infrastructure for exchanging materials information and knowledge. *Data Science Journal*, 9:54–61, 2010. `doi:10.2481/dsj.008-041`.

4 Bharathan Balaji, Arka Bhattacharya, Gabriel Fierro, Jingkun Gao, Joshua Gluck, Dezhi Hong, Aslak Johansen, Jason Koh, Joern Ploennigs, Yuvraj Agarwal, Mario Berges, David Culler, Rajesh Gupta, Mikkel Baun Kjærgaard, Mani Srivastava, and Kamin Whitehouse. Brick: Towards a unified metadata schema for buildings. In *Proceedings of the 3rd ACM International Conference on Systems for Energy-Efficient Built Environments*, pages 41–50, Palo Alto, CA, USA, 2016. ACM. `doi:10.1145/2993422.2993577`.

5 Conrad Bock, Achille Fokoue, Peter Haase, Rinke Hoekstra, Ian Horrocks, Alan Ruttenberg, Uli Sattler, and Michael Smith. OWL 2 web ontology language, 2012. URL: `https://www.w3.org/TR/owl2-syntax/`.

6 Brick Schema Development Team. Brick Schema change logs. Last accessed: April 1, 2025. URL: https://github.com/BrickSchema/Brick/releases.

7 Silvio Domingos Cardoso, Marcos Da Silveira, and Cédric Pruski. Construction and exploitation of an historical knowledge graph to deal with the evolution of ontologies. *Knowledge-Based Systems*, 194:105508, 2020. doi:10.1016/j.knosys.2020.105508.

8 Silvana Castano, Alfio Ferrara, and Stefano Montanelli. A matchmaking-based ontology evolution methodology. In *Proceedings of the Open Interop Workshop on Enterprise Modelling and Ontologies for Interoperability, Co-located with CAiSE'06 Conference*, volume 200, Luxembourg, 2006. CEUR-WS.org. URL: https://ceur-ws.org/Vol-200/02.pdf.

9 European Materials Modelling Council. EMMO, n.d. URL: https://github.com/emmo-repo/EMMO.

10 Jérôme Euzenat. Semantic precision and recall for ontology alignment evaluation. In *Proceedings of the 20th international joint conference on Artificial intelligence*, pages 348–353, Hyderabad, India, 2007. Morgan Kaufmann. URL: http://ijcai.org/Proceedings/07/Papers/054.pdf.

11 Jérôme Euzenat and Pavel Shvaiko. *Ontology Matching (2nd ed.)*. Springer, Berlin, Germany, 2013. doi:10.1007/978-3-642-38721-0.

12 Google DeepMind. Google Gemma Models, n.d. URL: https://ai.google.dev/gemma.

13 Anika Gross, Michael Hartung, Andreas Thor, and Erhard Rahm. How do computed ontology mappings evolve? - a case study for life science ontologies. In *Proceedings of the 2nd Joint Workshop on Knowledge Evolution and Ontology Dynamics in conjunction with the 11th International Semantic Web Conference (ISWC 2012)*, volume 890, Boston, MA, USA, 2012. CEUR-WS.org. URL: https://ceur-ws.org/Vol-890/paper1.pdf.

14 Thomas R. Gruber. A translational approach to portable ontologies. *Knowledge Acquisition*, 5(2):199–220, 1993. doi:10.1006/knac.1993.1008.

15 Armin Haller and Axel Polleres. Are we better off with just one ontology on the web? *Semantic Web*, 11(1):87–99, 2020. doi:10.3233/SW-190379.

16 Karl Hammar, Erik Oskar Wallin, Per Karlberg, and David Hälleberg. The RealEstateCore ontology. In *The Semantic Web - ISWC 2019*, pages 130–145, Auckland, New Zealand, 2019. Springer. doi:10.1007/978-3-030-30796-7_9.

17 Jeff Heflin and James Hendler. Dynamic ontologies on the web. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence*, pages 443–449, Austin, TX, USA, 2000. AAAI Press. URL: https://aaai.org/papers/00443-AAAI00-068-dynamic-ontologies-on-the-web/.

18 Sven Hertling, Jan Portisch, and Heiko Paulheim. MELT – matching evaluation toolkit. In *Semantic Systems. The Power of AI and Knowledge Graphs*, volume 11702, pages 231–245, Karlsruhe, Germany, 2019. Springer. doi:10.1007/978-3-030-33220-4_17.

19 iNovex IRAD. Ontologies for the MatOnto project, n.d. URL: https://github.com/inovexcorp/MatOnto-Ontologies.

20 Michel Klein and Dieter Fensel. Ontology versioning on the semantic web. In *Proceedings of the First International Conference on Semantic Web Working*, pages 75–91, Stanford University, CA, USA, 2001. CEUR-WS.org. URL: https://dl.acm.org/doi/10.5555/2956602.2956610.

21 Michel Klein, Dieter Fensel, Atanas Kiryakov, and Damyan Ognyanov. Ontology versioning and change detection on the web. In *Knowledge Engineering and Knowledge Management: Ontologies and the Semantic Web*, pages 197–212, Sigüenza, Spain, 2002. Springer. doi:10.1007/3-540-45810-7_20.

22 Adrianna Kozierkiewicz and Marcin Pietranik. Triggering ontology alignment revalidation based on the degree of change significance on the ontology concept level. In *Business Information Systems*, pages 137–148, Seville, Spain, 2019. Springer. doi:10.1007/978-3-030-20485-3_11.

23 Adrianna Kozierkiewicz and Marcin Pietranik. Updating ontology alignment on the concept level based on ontology evolution. In *Advances in Databases and Information Systems*, pages 201–214, Bled, Slovenia, 2019. Springer. doi:10.1007/978-3-030-28730-6_13.

24 Adrianna Kozierkiewicz and Marcin Pietranik. Updating ontology alignment on the relation level based on ontology evolution. In *Proceedings of the 15th International Conference on Evaluation of Novel Approaches to Software Engineering*, pages 241–248, Prague, Czech Republic, 2020. SciTePress. doi:10.5220/0009142002410248.

25 John J. McGowan. Project haystack data standards. In *Energy and Analytics*, pages 237–243. River Publishers, Aalborg, Denmark, 2020. doi:10.1201/9781003151944-16.

26 MELT Team. OAEI datasets. Last accessed: April 1, 2025. URL: https://dwslab.github.io/melt/track-repository.

27 Meta. llama-3-8b, 2024. URL: https://huggingface.co/meta-llama/Meta-Llama-3-8B.

28 Meta. Meta Llama Models, n.d. URL: https://www.llama.com.

29 Natalya F. Noy, Sandhya Kunnatur, Michel Klein, and Mark A. Musen. Tracking changes during ontology evolution. In *The Semantic Web – ISWC 2004*, pages 259–273, Hiroshima, Japan, 2004. Springer. doi:10.1007/978-3-540-30475-3_19.

30 Natalya F. Noy and Mark A. Musen. PROMPTDIFF: A fixed-point algorithm for comparing ontology versions. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence*, pages 744–750, Edmonton, Canada, 2002. AAAI Press. URL: https://aaai.org/papers/00744-aaai02-112-promptdiff-a-fixed-point-algorithm-for-comparing-ontology-versions/.

31 Natalya F. Noy and Mark A. Musen. Ontology versioning in an ontology management framework. *IEEE Intelligent Systems*, 19(4):6–13, 2004. doi:10.1109/MIS.2004.33.

32 OAEI Community. Ontology Alignment Evaluation Initiative (OAEI), n.d. URL: https://oaei.ontologymatching.org.

33 OpenAI. OpenAI Models, n.d. URL: https://platform.openai.com/docs/models.

34 Marcin Pietranik and Adrianna Kozierkiewicz. Methods of managing the evolution of ontologies and their alignments. *Applied Intelligence*, 53(17):20382–20401, 2023. doi:10.1007/S10489-023-04545-0.

35 Peter Plessers and Olga De Troyer. Ontology change detection using a version log. In *The Semantic Web – ISWC 2005*, pages 578–592, Galway, Ireland, 2005. Springer. doi:10.1007/11574620_42.

36 Zhangcheng Qiang, Weiqing Wang, and Kerry Taylor. Agent-OM. Last accessed: April 1, 2025. URL: https://github.com/qzc438/ontology-llm.

37 Zhangcheng Qiang, Weiqing Wang, and Kerry Taylor. Agent-OM: Leveraging LLM agents for ontology matching. *Proceedings of the VLDB Endowment*, 18(3):516–529, 2024. doi:10.14778/3712221.3712222.

38 RDF-star Working Group. RDF 1.2 schema, 2024. URL: https://www.w3.org/TR/rdf12-schema/.

39 Najla Sassi, Wassim Jaziri, and Saad Alharbi. Supporting ontology adaptation and versioning based on a graph of relevance. *Journal of Experimental & Theoretical Artificial Intelligence*, 28(6):1035–1059, 2016. doi:10.1080/0952813X.2015.1056239.

40 Team GLM. ChatGLM: A family of large language models from GLM-130B to GLM-4 all tools, 2024. doi:10.48550/arXiv.2406.12793.

41 Ahmed Zahaf and Mimoun Malki. Alignment evolution under ontology change. *International Journal of Information Technology and Web Engineering*, 11(2):14–38, 2016. doi:10.4018/IJITWE.2016040102.

42 Abir Zekri, Zouhaier Brahmia, Fabio Grandi, and Rafik Bouaziz. τOWL: A systematic approach to temporal versioning of semantic web ontologies. *Journal on Data Semantics*, 5(3):141–163, 2016. doi:10.1007/s13740-016-0066-3.