Computation of fuzzy joins over large collections of JSON data using semantic similarity

Alan Petit[1,*], Matthew Damigos[2], Laurent D'Orazio[3] and Eleftherios Kalogeros[2]

[1]*Polytech Nantes, Nantes Université, Nantes, France*

[2]*Laboratory on Digital Libraries and Electronic Publishing, Department of Archives, Library Science & Museology, Ionian University, Corfu, Greece*

[3]*Univ Rennes, CNRS, IRISA, Lannion, France*

Abstract

In recent years, fuzzy joins have emerged as a critical tool in data integration, data cleaning, and similarity-based retrieval tasks, especially when dealing with noisy or heterogeneous data. Although fuzzy joins have been widely explored for relational data, existing approaches typically focus on textual similarity, matching records based on certain text values and text distances. Apart from relational data, syntactic similarity of XML and JSON documents has also been investigated, where the matching documents are compared over their hierarchical structure. These methods often struggle when confronted with variations in element names, formats, or hierarchical layouts. In contrast, the semantic similarity of semi-structured data, such as JSON documents, remains underexplored, despite its importance in capturing the true meaning conveyed by document values. This paper addresses the problem of performing fuzzy joins between JSON documents by comparing their semantic content rather than their structural form. We propose and describe several methods based on document embeddings, aiming to capture the meaning of a document from its values and contextual relationships. The proposed techniques offer a promising foundation for semantic joining tasks in semi-structured data environments.

Keywords

Fuzzy join, JSON similarity, semantic similarity, embeddings, structural comparison, distributed computation

1. Introduction

The growing volume and heterogeneity of semi-structured data, particularly JSON (JavaScript Object Notation) documents, poses significant challenges for data integration, data cleaning, similarity-based information retrieval and analysis tasks. In particular, organizations need more and more to integrate, deduplicate, and correlate JSON documents that describe the same real-world entities or events, originating from different sources and production pipelines. Such documents are rarely uniform. They may use different key names (e.g., author vs. creator), different value conventions and different nesting schemes, resulting a variety of differences in both structure and content. As a result, conventional equality joins, which rely on strict equality between attributes, fail to identify such correspondences. The problem becomes more challenging when the volume of the input datasets is large.

Fuzzy joins address this limitation by replacing strict equality with a similarity measure; i.e., returning pairs of documents whose similarity exceeds a predefined threshold [1]. While fuzzy-join techniques have been extensively studied for relational data and for text-centric matching (e.g., edit distance, Jaccard, cosine over token sets) [1, 2, 3, 4, 5, 6, 7, 8], to evaluate fuzzy joins over collections of semi-structured data is not straightforward since not only the text-values but also the hierarchical structure need to be taken into account. Since JSON data is self-descriptive, multiple structures of the same information might be found. To overcome this issue, computing structural similarity has been extensively investigated in the past for tree structures (e.g., [9, 10, 11]), XML documents (e.g., [12, 13]), and graphs (e.g., [14, 15]). In

DOLAP 2026: 28th International Workshop on Design, Optimization, Languages and Analytical Processing of Big Data, co-located with EDBT/ICDT 2026, March 24, 2026, Tampere, Finland

[*]Corresponding author.

alan.petit04@gmail.com (A. Petit); mgdamigos@ionio.gr (M. Damigos); laurent.dorazio@univ-rennes.fr (L. D'Orazio); kalogero@ionio.gr (E. Kalogeros)

0000-0003-0431-1482 (M. Damigos); 0000-0001-8614-1848 (L. D'Orazio); 0000-0001-8080-6343 (E. Kalogeros)

© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR-WS.org/Vol-4186/short3.pdf

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073

published 2026-03-25

the last decade, structural similarity over JSON documents has also attracted attention [16, 17, 18, 19], where the main challenge is to support both ordered (i.e., arrays) and unordered (i.e., objects) sets of values. These approaches [16, 17, 18, 19] focus on variations of tree edit distance to measure structural similarity, and focus on single-machine computation. Taking into account both structure and content of JSON documents, however, is still an open problem.

In this work, we present a distributed evaluation method for computing fuzzy joins over large collections of JSON data, handling similarity over both content and structure. The proposed approach uses methods for converting JSON documents first to text and then to document embeddings, aiming to preserve both content and structure through descriptive representations of the information encoded in the JSON documents. In particular, the main contributions of this paper are summarized as follows: (1) we propose a novel approach for evaluating fuzzy joins over collections of JSON documents, by using a two-step transformation of each document to a document embedding that captures the meaning of the information encoded in the JSON documents, (2) present five different transformation approaches of a document to its text-representation, where the four of them use specialized prompts posed on an LLM, and (3) present scalable MapReduce algorithms for computing the fuzzy joins over large collections of generated embeddings, avoiding shuffling the JSON documents during join computation.

2. Preliminary Definitions

This paper focuses on JSON (JavaScript Object Notation) [20] documents. A JSON document is either a possibly empty object (unordered list enclosed in curly brackets { }) of key-value pairs or a possibly empty array of values (ordered list enclosed into square brackets [ ]), where (a) the values could be either objects or arrays, or primitive values (i.e., string, number, boolean, or null), and (b) the keys are strings. The keys are unique within an object. A **JSON collection** (or simply, a *collection*) is a multiset/bag of JSON documents, which do not necessarily share the same structure.

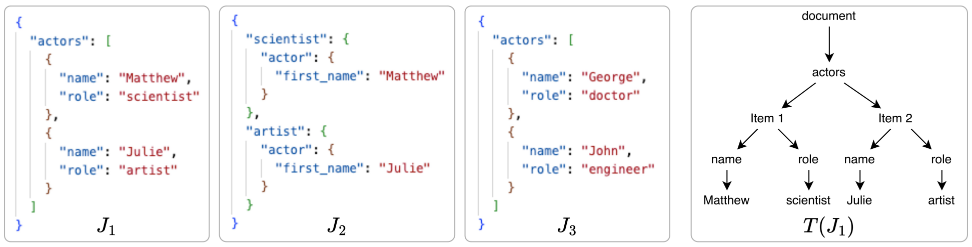
To process large JSON collections, we use a distributed processing framework. Although Apache Spark [21] is a widely-used, high-performance distributed processing engine, we present our algorithms in the MapReduce [22] model, which can also be implemented straightforwardly in Spark. This model lets users design processing pipelines by defining Map and Reduce functions over key-value pairs. The Map function takes a pair (k_1, v_1) and emits intermediate pairs (k_2, v_2): $(k_1, v_1) \xrightarrow{map} (k_2, v_2)$. The intermediate values with the same key k_2 are grouped since they are transmitted to the same compute node, and the Reduce function applies the predefined computation to each group: $(k_2, [v_2, \ldots]) \xrightarrow{reduce} (k_3, v_3)$.

Large Language Models (LLMs) [23], such as GPT (Generative Pre-trained Transformer) [24] and its subsequent versions (e.g., LLaMA [25]), are sophisticated neural architectures developed using huge collections of text data to forecast and produce coherent, contextually appropriate natural language, typically using transformer architectures [26]. Effective utilization of LLMs requires prompt engineering [27], i.e., designing LLM inputs (a.k.a. prompts) that guide the model's behavior (e.g., zero-shot prompting [28], chain-of-thought prompting [29]) toward a desired output. Apart from text generation, LLMs rely on and produce dense vector representations of linguistic elements, called *embeddings*, mapping elements (e.g., words [30], sentences, or documents [31, 32]) into points within a continuous, low-dimensional vector space so that geometric relationships (e.g., cosine similarity) represent semantic similarity [33]. Transformer-based models produce contextual token embeddings, while sentence/document embeddings map variable-length text spans into fixed-dimensional vectors that are directly comparable via standard similarity measures [34].

3. Fuzzy join based on semantic similarity

In this section, we investigate the problem of distributed computation of a fuzzy join over two JSON collections using semantic similarity. In general, a fuzzy join is a join operation applied over two collections $\mathcal{C}_1, \mathcal{C}_2$ of records, and its result is a new collection containing the pairs of records from $\mathcal{C}_1$,

$\mathcal{C}_2$ that are similar according to a given similarity measure and a threshold. In our setting, the joining collections include JSON documents. Unlike prior fuzzy join methods that focus on textual similarity over selected values or on structural similarity, we assess JSON-record similarity by jointly considering content (keys and values) and structure. For example, consider the JSON collections $\mathcal{C}_1 = \{J_1\}$ and $\mathcal{C}_2 = \{J_2, J_3\}$, where J_1, J_2, J_3 are illustrated in Figure 1. Intuitively, although J_1, J_3 share the same structure, and J_1, J_2 have different structures and vocabularies, we would like the joining result to return only the pair (J_1, J_2); since only this pair describes the same real-world entity (i.e., the documents are semantically highly-similar). It is easy to understand that traditional fuzzy joining approaches relying on structural similarity fail to capture the underlying semantic meaning of these documents.



```
{
  "actors": [
    {
      "name": "Matthew",
      "role": "scientist"
    },
    {
      "name": "Julie",
      "role": "artist"
    }
  ]
}
```

J_1

```
{
  "scientist": {
    "actor": {
      "first_name": "Matthew"
    }
  },
  "artist": {
    "actor": {
      "first_name": "Julie"
    }
  }
}
```

J_2

```
{
  "actors": [
    {
      "name": "George",
      "role": "doctor"
    },
    {
      "name": "John",
      "role": "engineer"
    }
  ]
}
```

J_3

Figure 1: Example - Semantically similar and dissimilar JSON documents

To address this limitation, our approach, named **FLESHJ-Join**[1], focuses on transforming each JSON document in the joining collections into a dense vector representation (embedding) that encodes its semantic content and structure, and then to join collections by comparing them using vector-space similarity measure [35] (e.g., cosine similarity, $sim_{cos}(\vec{V_1}, \vec{V_2}) = \frac{\vec{V_1} \cdot \vec{V_2}}{\|\vec{V_1}\| \cdot \|\vec{V_2}\|}$). Figure 2 illustrates an overview of the proposed semantic fuzzy-join framework, which is organised into an offline phase and an online phase. In the offline phase, for each JSON document included in each collection, we initially generate a text-representation that preserves the JSON semantics (content and structure), and then encode this representation into a fixed-dimensional vector space. The collections $\mathcal{T}_1$ and $\mathcal{T}_2$ include the text-representations of the documents in $\mathcal{C}_1$ and $\mathcal{C}_2$, and the collections $\mathcal{E}_1$ and $\mathcal{E}_2$ contain the corresponding embeddings, respectively. In the online phase, the precomputed embeddings in $\mathcal{E}_1$ and $\mathcal{E}_2$ are fed into a fuzzy-join execution module, which computes semantic similarity scores (e.g., cosine similarity) between vectors and identifies pairs whose similarity exceeds a given threshold.

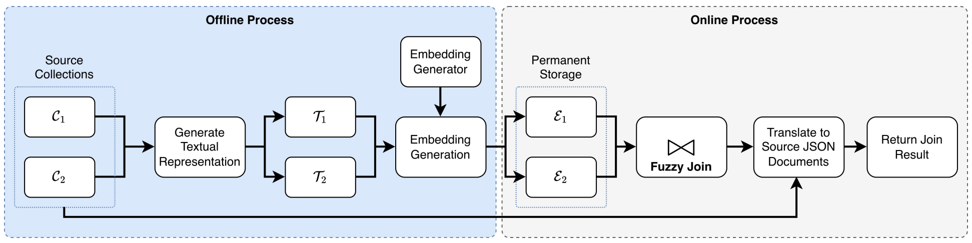


Figure 2: Overall processing pipeline of FLESHJ-Join

Formally, consider two collections of JSON documents C_1 and C_2, so that each document $J_i \in C_1 \cup C_2$ is associated with a unique identifier $id(J_i)$. Suppose two transformations t_1, t_2 transforming a JSON document to one or more text representations, and a text to an embedding, respectively. The FLESHJ-Join is defined as follows, where τ is a given similarity threshold: $C_1 \bowtie_{\geq\tau} C_2 = \{[J_1, J_2] \mid J_1 \in C_1, J_2 \in C_2,\ sim_{cos}(t_2(t_1(J_1)), t_2(t_1(J_2))) \geq \tau\}$. Our objective is to efficiently compute $C_1 \bowtie_{\geq\tau} C_2$ by maximizing the semantic accuracy (assessed using standard information retrieval metrics, such as precision, recall and $F1$ score, over a gold-standard set of matching pairs); i.e., if $\mathcal{O}$ is the desired output, we want $(C_1 \bowtie_{\geq\tau} C_2) - \mathcal{O} \simeq \emptyset$, while minimizing both false positives and false negatives.

[1]stands for Fuzzy LLM-based Embedding Similarity for Heterogeneous JSON

3.1. Transforming JSON documents into text representations and embeddings

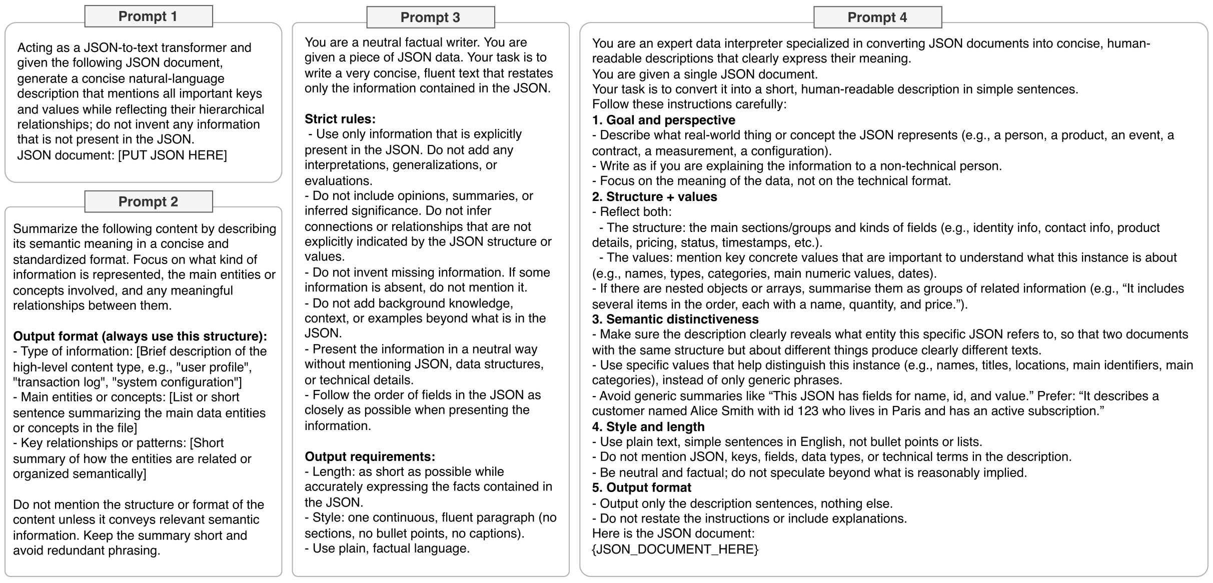
The primary challenge lies in designing scalable and effective methods for representing JSON documents through natural-language text. In this context, the first step consists in transforming each JSON document $J \in C_1 \cup C_2$ into a natural-language text-representation $t_1(J) \in \mathcal{T}$, using a transformation function $t_1 : C_1 \cup C_2 \rightarrow \mathcal{T}_1 \cup \mathcal{T}_2$. The goal of this transformation is twofold: (i) to preserve the semantic content of the original document (i.e., what real-world entity or concept it describes), and (ii) to encode enough information about the document's structure so that produced document embeddings can capture similarities that depend on both content and structure.

To generate the representative text from a certain JSON document, we consider two main methods. The first method simply generates the text by following a certain text-generation rules. The second one uses an LLM to generate this text-representation. In both cases, such a transformation is applied independently to each document, which makes it suitable for large-scale, parallel processing frameworks (e.g., Spark, or MapReduce).

In the first approach, we initially model each JSON document J as a tree $T(J) = (V, E, r, \ell)$ rooted at a special *document* node representing the entire JSON document, so that for each key and primitive value of the document, a tree node is considered. In addition, for each object element in a JSON array at position i, we create an item node labeled as "Item i". Edges are defined as parent–child links from each enclosing JSON construct to its component: from an object (or the document root) to its key nodes, from an array node (typically a key node) to each "Item i" node, and from each enclosing key/item node to its primitive value node. To construct such a tree, we follow a top-down approach, starting from the document node. Note that the nested arrays of primitive values are encoded using their flattened form; e.g., the array $[[''a'', ''b''], ''c'']$ is similarly treated as the array $[''a'', ''b'', ''c'']$. For example, in Figure 1, the tree-representation of document J_1 is the tree $T(J_1)$.

Based on the tree representation $T(J)$ of a JSON document J, we define text-generation rules that convert each node and its children into short declarative sentences. Traversing $T(J)$ from the root, for each internal node n we generate *"[n] has fields* $k_1, \ldots, k_r$*"* (or *"document is described by* $k_1, \ldots, k_r$*"* for the root), where $k_1, \ldots, k_r$ are the children of n. For nodes with leaf children, we generate *"[n] has values:* $x_1, \ldots, x_m$*"*, and when there is a single leaf value v we use *"The value of [n] is [v]"*. For example, for $T(J_1)$ we obtain: *"document is described by* $actors$*.* $actors$ *has fields* $Item1$, $Item2$*.* $Item1$ *has fields* $name, role$*. The value of* $name$ *is* $Matthew$*. The value of* $role$ *is* $scientist$*.* $Item2$ *has fields* $name, role$*. The value of* $name$ *is* $Julie$*. The value of* $role$ *is* $artist$*"*. Since multiple nodes might have the same key, we could use the reachable (root-to-node) path of each node from the root, instead of its single label.

The second approach uses an LLM to convert each JSON document into a concise text-representation that captures both structure and content, with the goal of expressing the document's meaning. We study four zero-shot, role-based prompts (Figure 3) that enforce faithfulness (no hallucinated keys/values) and a standardised, fluent output for FLESHJ-Join: Prompt 1 produces a compact paragraph reflecting important keys, values, and nesting; Prompt 2 generates a higher-level semantic summary (type of information, main entities, and relationships) while avoiding technical format details; Prompt 3 follows a similar style but restricts the output to information explicitly present in the JSON; and Prompt 4 uses process-guided constraints to generate a few simple, consistent sentences, improving embedding stability and semantic distinctiveness. These prompts define JSON-to-text transformations (t_1), span the trade-off between structural fidelity (Prompts 1/3) and higher-level semantics/readability (Prompts 2/4) and can be selected or combined per document. Prompt 1 and Prompt 3 are more tightly coupled to the original structure and values, whereas Prompt 2 and Prompt 4 focus on higher-level semantics and human interpretability. In our framework, any of these prompts (or combinations of them) can be used to generate text-representations of a JSON document. For example, the text-representation of J_1 depicted in Figure 1 is *"This document describes two actors. One actor is Matthew, whose role is a scientist. The other actor is Julie, whose role is an artist. It identifies each actor by name and role."*, which has been generated using Prompt 4. As you can see both the structure and content of J_1 are captured in this text.



Prompt 1

Acting as a JSON-to-text transformer and given the following JSON document, generate a concise natural-language description that mentions all important keys and values while reflecting their hierarchical relationships; do not invent any information that is not present in the JSON.
JSON document: [PUT JSON HERE]

Prompt 2

Summarize the following content by describing its semantic meaning in a concise and standardized format. Focus on what kind of information is represented, the main entities or concepts involved, and any meaningful relationships between them.

Output format (always use this structure):
- Type of information: [Brief description of the high-level content type, e.g., "user profile", "transaction log", "system configuration"]
- Main entities or concepts: [List or short sentence summarizing the main data entities or concepts in the file]
- Key relationships or patterns: [Short summary of how the entities are related or organized semantically]

Do not mention the structure or format of the content unless it conveys relevant semantic information. Keep the summary short and avoid redundant phrasing.

Prompt 3

You are a neutral factual writer. You are given a piece of JSON data. Your task is to write a very concise, fluent text that restates only the information contained in the JSON.

Strict rules:
- Use only information that is explicitly present in the JSON. Do not add any interpretations, generalizations, or evaluations.
- Do not include opinions, summaries, or inferred significance. Do not infer connections or relationships that are not explicitly indicated by the JSON structure or values.
- Do not invent missing information. If some information is absent, do not mention it.
- Do not add background knowledge, context, or examples beyond what is in the JSON.
- Present the information in a neutral way without mentioning JSON, data structures, or technical details.
- Follow the order of fields in the JSON as closely as possible when presenting the information.

Output requirements:
- Length: as short as possible while accurately expressing the facts contained in the JSON.
- Style: one continuous, fluent paragraph (no sections, no bullet points, no captions).
- Use plain, factual language.

Prompt 4

You are an expert data interpreter specialized in converting JSON documents into concise, human-readable descriptions that clearly express their meaning.
You are given a single JSON document.
Your task is to convert it into a short, human-readable description in simple sentences.
Follow these instructions carefully:
1. Goal and perspective
- Describe what real-world thing or concept the JSON represents (e.g., a person, a product, an event, a contract, a measurement, a configuration).
- Write as if you are explaining the information to a non-technical person.
- Focus on the meaning of the data, not on the technical format.
2. Structure + values
- Reflect both:
  - The structure: the main sections/groups and kinds of fields (e.g., identity info, contact info, product details, pricing, status, timestamps, etc.).
  - The values: mention key concrete values that are important to understand what this instance is about (e.g., names, types, categories, main numeric values, dates).
- If there are nested objects or arrays, summarise them as groups of related information (e.g., "It includes several items in the order, each with a name, quantity, and price.").
3. Semantic distinctiveness
- Make sure the description clearly reveals what entity this specific JSON refers to, so that two documents with the same structure but about different things produce clearly different texts.
- Use specific values that help distinguish this instance (e.g., names, titles, locations, main identifiers, main categories), instead of only generic phrases.
- Avoid generic summaries like "This JSON has fields for name, id, and value." Prefer: "It describes a customer named Alice Smith with id 123 who lives in Paris and has an active subscription."
4. Style and length
- Use plain text, simple sentences in English, not bullet points or lists.
- Do not mention JSON, keys, fields, data types, or technical terms in the description.
- Be neutral and factual; do not speculate beyond what is reasonably implied.
5. Output format
- Output only the description sentences, nothing else.
- Do not restate the instructions or include explanations.
Here is the JSON document:
{JSON_DOCUMENT_HERE}

Figure 3: Prompts transforming a JSON document to a text-representation.

3.2. Joining embeddings in a distributed environment

This section presents distributed evaluation techniques for fuzzy joins over embeddings. Once the embedding collections $\mathcal{E}_1, \mathcal{E}_2$ have been properly constructed, with $\mathcal{E}_i = \{(id(J), \vec{e}) \mid J \in C_i,\ \vec{e} = t_2(t_1(J))\}$ for $i \in \{1, 2\}$, FLESHJ-Join aims to find all pairs (J_1, J_2) such that $sim_{cos}(\vec{e}_1, \vec{e}_2) \geq \tau$, where $\vec{e}_i$ represents $J_i \in \mathcal{C}_i$. A naïve all-pairs evaluation requires $\mathcal{O}(|\mathcal{E}_1||\mathcal{E}_2|)$ comparisons and is infeasible for large JSON collections; moreover, in distributed settings we must also limit communication cost and balance work. Using MapReduce notation, we outline blocking strategies (e.g., LSH) to reduce communication and computation, without aiming for an exhaustive optimization study [36, 7, 37].

Let us initially model the input as a unified multiset of pairs $< id(J), (\vec{e}_J, C_J) >$, where $C_J \in \{C_1, C_2\}$. A generic MapReduce job for producing candidate pairs can be written as:
$< id(J), (\vec{e}_J, C_J) > \xrightarrow{\text{Map}} \{< k, (id(J), \vec{e}_J, C_J) > \mid k \in K(J)\}$
$< k, \{(id(J), \vec{e}_J, C_J)\} > \xrightarrow{\text{Reduce}} \{(id(J_1), id(J_2)) \mid J_1 \in C_1,\ J_2 \in C_2,\ sim_{cos}(\vec{e}_1, \vec{e}_2) \geq \tau\}$,
where $K(J)$ is a set of blocking keys determining replication and communication cost. Each reducer receives embeddings sharing a key k and evaluates a local join between the subsets from C_1 and C_2.

We consider two representative options for computing $K(J)$. (1) *Matrix bucketing* (baseline): embeddings from $\mathcal{E}_1$ and $\mathcal{E}_2$ are hashed to rows and columns of an $n \times n$ matrix, and each reducer processes a cell bucket. This guarantees full coverage but can incur high replication [38, 39]. If r is the number of reducers, it is recommended to select $n \approx \sqrt{2r}$. (2) *LSH blocking*: for cosine similarity, let h_ℓ ($\ell = 1, \ldots, L$) be independent LSH functions [40, 41, 42] and define $K(J) = \{h_\ell(\vec{e}_J) \mid \ell = 1, \ldots, L\}$. The map function emits one pair per hash value, while each reducer compares only vectors co-assigned to the same bucket, followed by optional duplicate elimination across buckets. Matrix bucketing is deterministic but expensive in communication, while LSH trades deterministic coverage for probabilistic recall with reduced shuffle; increasing L improves recall at the cost of additional replication.

4. Preliminary Evaluation

This section presents a preliminary evaluation focusing on the generation of text-representations from JSON documents. To evaluate our approach, since no publicly available dataset is available, we generated a dataset [2] of ∼200K JSON documents using ChatGPT[3], ensuring that ≈

[2]The dataset is available here: https://github.com/DLIB-Ionian-University/fuzzyjoin-json-dataset
[3]OpenAI ChatGPT, https://chatgpt.com/

5% of the documents form pairs with at least 80% similarity in content and/or structure (including similar content under different structures, and similar structure with different content). From this dataset, we evaluated 31 documents (9 matching pairs with similarity $\geq 80\%$), manually validated the similarity estimates (and computing precision, recall, and accuracy). For each prompt, we generated text representations (Meta Llama 3.3-70B-Instruct), embedded them (all-MiniLM-L6-v2), and compared the 220 cross-collection pairs with cosine similarity; Table 1 reports precision, recall, and accuracy. Three prompts achieve accuracy $\geq 95\%$; Prompt 3 yields the best precision/recall trade-off, with Prompt 4 close. Metrics are computed over pairs with similarity $\geq 80\%$, where a produced pair is counted as correct if it is estimated to have similarity $\geq 80\%$ and its computed cosine similarity is also $\geq 80\%$.

Prompt	**Precision**	**Recall**	**Accuracy**
Prompt 1	47%	89%	95%
Prompt 2	13%	100%	74%
Prompt 3	82%	100%	99%
Prompt 4	64%	100%	98%

Table 1: Accuracy, precision, and recall.

5. Related Work

Existing work on fuzzy joins in distributed environments has largely focused on structural or token-based similarity measures [43, 44, 45, 38, 46], often combined with optimization techniques such as filtering, prefix indexing or partition-based pruning. Approaches leveraging MapReduce and its derivatives [38, 47, 48] have enabled large-scale execution, with improvements including Bloom filter-based pruning, ball hashing and substring splitting [47] to reduce candidate comparisons. While these methods are effective for structure-oriented matching, for example when using Hamming or edit distance on flattened keys, they struggle to capture cases where two JSON documents are structurally different but semantically equivalent. In the context of assessing structural similarity in semi-structured data (e.g., JSON documents), it is worth mentioning JSONVec [49] and ORIGAMI [50], both of which propose methods for encoding JSON structures into embeddings. Recent advances in semantic similarity, such as sentence embeddings and large language models (LLMs), offer new opportunities, but integrating them efficiently into a distributed fuzzy join pipeline introduces new computational and communication bottlenecks. Moreover, research on semantic similarity in order to create clusters of files has been done [51, 52, 53, 54], and shows that semantic similarity between JSON files is already useful for clustering; we therefore investigate how to repurpose these semantic signals to perform fuzzy joins with higher recall and precision. Computing such a join is non-trivial, especially when performed at scale. In addition to the inherent complexity of comparing high-dimensional semantic embeddings, the integration of language models into a distributed computation pipeline introduces additional challenges, including network overhead, fault tolerance, load balancing, and the serialization of model calls across the cluster. These factors make the deployment of fuzzy join operations in real-world distributed systems particularly demanding.

6. Conclusion and Future Work

In this work, we presented a distributed semantic fuzzy join for JSON collections that (1) textualizes each document to preserve content and structure (rule- or prompt-based), (2) encodes the text as embeddings, and (3) applies a scalable similarity-join pipeline over vectors. Future work will expand evaluation on larger datasets, study prompt/embedding choices, investigate extensions of JSONVec/ORIGAMI [49, 50] and study distributed cost optimizations (e.g., batching/caching model calls).

Declaration on Generative AI

During the preparation of this work, the authors used GPT in order to: Grammar and spelling check. After using these tool/service, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] Y. Jiang, G. Li, J. Feng, W.-S. Li, String similarity joins: An experimental evaluation, Proc. VLDB Endow. 7 (2014) 625–636.

[2] R. Vernica, M. J. Carey, C. Li, Efficient Parallel Set-similarity Joins Using MapReduce, in: SIGMOD, 2010, pp. 495–506.

[3] D. Deng, G. Li, S. Hao, J. Wang, J. Feng, Massjoin: A mapreduce-based method for scalable string similarity joins, in: ICDE, 2014, pp. 340–351.

[4] X. Wang, D. Sun, Qjoin: A q-sample-based method for large-scale string similarity joins, in: ISCID, 2018, pp. 45–48.

[5] T.-T.-Q. Tran, T.-C. Phan, A. Laurent, L. d' Orazio, Improving hamming distance-based fuzzy join in mapreduce using bloom filters, in: FUZZ-IEEE, 2018, pp. 1–7.

[6] T. Thi-To-Quyen, P. Thuong-Cang, A. Laurent, L. d'Orazio, Optimization for large-scale fuzzy joins using fuzzy filters in mapreduce, in: FUZZ-IEEE, IEEE, 2020, pp. 1–8.

[7] F. N. Afrati, A. D. Sarma, D. Menestrina, A. Parameswaran, J. D. Ullman, Fuzzy Joins Using MapReduce, in: ICDE, 2012, pp. 498–509.

[8] M. Yu, G. Li, D. Deng, J. Feng, String similarity search and join: a survey, Frontiers of Computer Science 10 (2016) 399–417.

[9] M. Pawlik, N. Augsten, Tree edit distance: Robust and memory-efficient, Information Systems 56 (2016) 157–173.

[10] R. Yang, P. Kalnis, A. K. Tung, Similarity evaluation on tree-structured data, in: SIGMOD, 2005, pp. 754–765.

[11] L. Wang, K. Zhang, Space efficient algorithms for ordered tree comparison, Algorithmica 51 (2008) 283–297.

[12] J. Tekli, R. Chbeir, K. Yetongnon, An overview on XML similarity: Background, current trends and future directions, Computer science review 3 (2009) 151–173.

[13] G. Cobena, S. Abiteboul, A. Marian, Detecting changes in XML documents, in: ICDE, IEEE, 2002, pp. 41–52.

[14] C. Donnat, S. Holmes, Tracking network dynamics: A survey of distances and similarity metrics, arXiv preprint arXiv:1801.07351 (2018).

[15] D. Koutra, A. Parikh, A. Ramdas, J. Xiang, Algorithms for graph similarity and subgraph matching, in: Proc. Ecol. inference conf, volume 17, Citeseer, 2011.

[16] D. U. Priya, P. S. Thilagam, JSON document clustering based on schema embeddings, Journal of Information Science (2022).

[17] T. Hütter, N. Augsten, C. M. Kirsch, M. J. Carey, C. Li, JEDI: These aren't the JSON documents you're looking for?, in: SIGMOD, 2022, pp. 1584–1597.

[18] E. Ben Hadj Yahia, J.-R. Falleri, L. Réveillère, Polly: A language-based approach for custom change detection of web service data, in: ICSOC, Springer, 2017, pp. 430–444.

[19] H. Cao, J.-R. Falleri, X. Blanc, L. Zhang, JSON patch for turning a pull REST API into a push, in: ICSOC, Springer, 2016, pp. 435–449.

[20] T. Bray, The javascript object notation (JSON) data interchange format, RFC 7158 (2014) 1–16.

[21] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, I. Stoica, Spark: Cluster computing with working sets, in: Proceedings of the USENIX Conference on Hot Topics in Cloud Computing (HotCloud), 2010, pp. 10–10.

[22] J. Dean, S. Ghemawat, MapReduce: Simplified data processing on large clusters, in: Proceedings of the Symposium on Operating Systems Design and Implementation (OSDI), 2004, pp. 137–150.

[23] W. X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, Y. Min, B. Zhang, J. Zhang, Z. Dong, et al., A survey of large language models, arXiv preprint arXiv:2303.18223 1 (2023).

[24] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever, et al., Improving language understanding by generative pre-training (2018).

[25] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, et al., Llama: Open and efficient foundation language models, arXiv preprint

arXiv:2302.13971 (2023).

[26] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, I. Polosukhin, Attention is all you need, Advances in neural information processing systems 30 (2017).

[27] P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, G. Neubig, Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing, ACM computing surveys 55 (2023) 1–35.

[28] J. Wei, M. Bosma, V. Y. Zhao, K. Guu, A. W. Yu, B. Lester, N. Du, A. M. Dai, Q. V. Le, Finetuned language models are zero-shot learners, arXiv preprint arXiv:2109.01652 (2021).

[29] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou, et al., Chain-of-thought prompting elicits reasoning in large language models, Advances in neural information processing systems 35 (2022) 24824–24837.

[30] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, J. Dean, Distributed representations of words and phrases and their compositionality, Advances in neural information processing systems 26 (2013).

[31] Z. Lin, M. Feng, C. N. d. Santos, M. Yu, B. Xiang, B. Zhou, Y. Bengio, A structured self-attentive sentence embedding, arXiv preprint arXiv:1703.03130 (2017).

[32] N. Reimers, I. Gurevych, Sentence-bert: Sentence embeddings using siamese bert-networks, arXiv preprint arXiv:1908.10084 (2019).

[33] Y. Bengio, R. Ducharme, P. Vincent, C. Jauvin, A neural probabilistic language model, Journal of machine learning research 3 (2003) 1137–1155.

[34] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, Bert: Pre-training of deep bidirectional transformers for language understanding, in: Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers), 2019, pp. 4171–4186.

[35] W. H. Gomaa, A. A. Fahmy, et al., A survey of text similarity approaches, international journal of Computer Applications 68 (2013) 13–18.

[36] B. Yang, H. J. Kim, J. Shim, D. Lee, S.-g. Lee, Fast and scalable vector similarity joins with mapreduce, Journal of Intelligent Information Systems 46 (2016) 473–497.

[37] R. Baraglia, G. D. F. Morales, C. Lucchese, Document similarity self-join with mapreduce, in: 2010 IEEE International Conference on data mining, IEEE, 2010, pp. 731–736.

[38] R. Uhartegaray, L. D'Orazio, M. Damigos, E. Kalogeros, Scalable computation of fuzzy joins over large collections of json data, IEEE International Conference on Fuzzy Systems (FUZZ-IEEE) (2023).

[39] A. Okcan, M. Riedewald, Processing theta-joins using mapreduce, in: Proceedings of the 2011 ACM SIGMOD International Conference on Management of data, 2011, pp. 949–960.

[40] M. S. Charikar, Similarity estimation techniques from rounding algorithms, in: Proceedings of the thiry-fourth annual ACM symposium on Theory of computing, 2002, pp. 380–388.

[41] A. Andoni, P. Indyk, T. Laarhoven, I. Razenshteyn, L. Schmidt, Practical and optimal lsh for angular distance, Advances in neural information processing systems 28 (2015).

[42] X. Hu, Y. Tao, K. Yi, Output-optimal parallel algorithms for similarity joins, in: Proceedings of the ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, 2017, pp. 79–90.

[43] T. Hütter, N. Augsten, C. M. Kirsch, M. J. Carey, C. Li, JEDI: These aren't the JSON documents you're looking for..., SIGMOD '22 (2022).

[44] A. Sevim, A. Eldawy, E. P. Carman, M. J. Carey, V. J. Tsotras, Fudj: Flexible user-defined distributed joins, IEEE International Conference on Fuzzy Systems (2024).

[45] T. Mizokami, S. Bou, T. Amagasa, Subtree similarity search based on structure and text, in: Big Data Analytics and Knowledge Discovery, 2024.

[46] M. Yu, G. Li, D. Deng, J. Feng, String similarity search and join : A survey, Frontiers of Computer Science (2015).

[47] T.-T.-Q. Tran, T.-C. Phan, A. Laurent, L. D'Orazio, Optimization for large-scale fuzzy joins using fuzzy filters in mapreduce, IEEE International Conference on Fuzzy Systems (2020).

[48] A. Drissi, L. d'Orazio, M. Damigos, Scalable structural similarity analysis of json documents using mapreduce, IEEE International Conference on Fuzzy Systems, Jul 2025 (2025).

[49] W. Woof, K. Chen, A framework for end-to-end learning on semantic tree-structured data, arXiv preprint arXiv:2002.05707 (2020).

[50] T. Rückstieß, A. Huang, R. Vujanic, ORIGAMI: A generative transformer architecture for predictions from semi-structured data, arXiv preprint arXiv:2412.17348 (2024).

[51] U. P. Dharmaraj, P. S. Thilagam, JSON document clustering based on schema embeddings, Journal of information science (2022).

[52] U. P. Dharmaraj, P. S. Thilagam, JSON document clustering based on structural similarity and semantic fusion, in: Proceedings of International Conference on Computational Intelligence and Data Engineering, 2023.

[53] D. Chandrasekaran, V. Mago, Evolution of semantic similarity - a survey, ACM Computing Surveys (2021).

[54] D. R. Santana, P. H. S. Lima, L. A. Ribeiro, EM-Join: Efficient entity matching using embedding-based similarity join, in: Proceedings of the International Conference on Enterprise Information Systems, 2025.