LLM4Schema.org: Generating Schema.org Markups with Large Language Models

Minh-Hoang Dang [a,*], Thi Hoang Thi Pham [a], Pascal Molli [a], Hala Skaf-Molli [a] and Alban Gaignard [b]
[a] *Laboratoire des Sciences du Numérique de Nantes (LS2N), Nantes Université, 44000 Nantes, France*
E-mails: minh-hoang.dang@univ-nantes.fr, thi-hoang-thi.pham@univ-nantes.fr, pascal.molli@univ-nantes.fr, hala.skaf@univ-nantes.fr
[b] *Nantes Université, CNRS, INSERM, l'Institut du Thorax F-44000 Nantes, France*
E-mail: alban.gaignard@univ-nantes.fr

Abstract.
The integration of Schema.org markup in web pages has resulted in the creation of billions of RDF triples, yet approximately 75% of all web pages still lack these essential markups. Large Language Models (LLMs) offer a potential solution by automatically generating the missing Schema.org markups. However, the accuracy and reliability of LLM-generated markups compared to human annotators remain uncertain. This paper introduces LLM4Schema.org, a novel approach to evaluate the performance of LLMs in generating Schema.org markups. Our study identifies that 40-50% of the markup produced by GPT-3.5 and GPT-4 are either invalid, non-factual, or non-compliant with the Schema.org ontology. We show that these errors can be identified and removed using specialized agents powered by LLMs. Once errors are filtered out, GPT-4 outperforms human annotators in generating accurate and comprehensive Schema.org markups. Both GPT-3.5 and GPT-4 are capable of making improvements in areas where human annotators fall short. LLM4Schema.org highlights the potential and challenges of using LLMs for semantic annotation, emphasizing the importance of curation to achieve reliable results.

Keywords: Schema.org, Large Language Model, Knowledge Graph Construction

1. Introduction

"The price of this book is 30€, this product is in stock, that recipe will take 30 minutes, this job is full-time." Search engines use the Schema.org markup embedded in web pages to better understand their content and enrich search results with precise information. This markup can be authored directly by humans when writing their web pages or indirectly generated by software such as Web Content Management Systems configured to produce Schema.org markup [1]. Schema.org markup adheres to the Schema.org ontology, which consists of 806 types and over 1400 properties to describe entities such as people, organizations, creative works, events . . . The nature of entities and information differs significantly when comparing Schema.org markup to a public knowledge graph like Wikidata. For example, Wikidata has approximately 43K books registered, whereas Schema.org markup encompasses over 3,5M books [10]. While Wikidata tends to describe well-known books, people, and organizations, Schema.org describes any book, any shop, or any individual, providing a broader scope. Additionally, Wikidata includes encyclopedic information (e.g., a book's literary genre), while Schema.org includes specific information (e.g., the price, the vendor,

*Corresponding author. E-mail: minh-hoang.dang@univ-nantes.fr.
[1]Through the paper, we refer to this as human markup

1570-0844/$35.00 © 0 – IOS Press. All rights reserved.

etc.). Thus, Schema.org markup can offer valuable information absent in public knowledge graphs, contributing to more comprehensive and diverse knowledge. Ensuring the availability of such knowledge is crucial to maintaining knowledge graphs that reflect the wealth of information available on the web.

Currently, semantic annotations are present in 41% of the world's web pages, and 25% specifically using Schema.org markup [7, 10]. This leaves 75% of web pages without Schema.org markup and, thus, lacking structured data about their content. Large Language Models (LLMs) can generate Schema.org markup from text [22], essentially "reading" the web as text to generate the missing Schema.org markup. However, ensuring that the Schema.org markup is generated correctly and comprehensively, regardless of the types of entities involved or the web page's text, is a critical question.

Previous researches [15, 24, 26, 34] have explored using LLMs to generate RDF triples from text according to ontology, but these studies have focused on small ontologies and brief sentences from the text benchmarks. Consequently, there is a gap in understanding the performance of LLMs on real, potentially lengthy web pages with a real ontology like Schema.org, which includes 806 types and 1476 properties[2].

To address this gap, we propose LLM4Schema.org, a novel approach to *assess whether LLMs can generate better Schema.org markup than humans.* This comparison poses scientific challenges: LLMs might generate more complete markup through hallucination, i.e., fabricating data, while humans might introduce markup from external sources.

We propose that humans and LLMs start annotating from the same input: the web page's text to enable fair comparison. Only correct Schema.org markup will be considered, i.e., syntactically and semantically valid Schema.org markup grounded in the text. Under these conditions, the more complete Schema.org markup wins the match.

The paper presents the following scientific contributions:

1. We created a diversified corpus of webpages containing Schema.org markup by sampling WebDataCommons [7, 10].
2. We developed three agents to curate Schema.org markup : 1) *The Validity agent* checks if the markup is valid, ensuring syntactic conformance. 2) *The Factuality agent* verifies that every markup statement can be accurately extracted from the text of the web page. 3) *The Compliance agent* ensures semantic conformance, i.e., that each markup value complies with the expected values declared in the Schema.org documentation.
3. To compare the performance of humans vs LLMs in the task of annotating real web pages with Schema.org markup, we propose MIMR, a metric to evaluate the completeness of two verified markups. We devised a simple yet effective metric by comparing them to an ideal merged markup. We evaluated the accuracy of this metric with a user-study.
4. Lastly, we constructed a comprehensive pipeline enabling a fair comparison between LLMs and Humans for our corpus. Thanks to this pipeline, we estimate that approximately 40-50% of the markup produced by GPT-3.5 or GPT-4 from the text with state-of-art prompts is either non-factual or non-compliant. Without curation, the LLM-generated markup is clearly not reliable. After curation, we estimate that GPT3-curated markup is worse than Human-generated markup, but GPT4-curated markup is better than the one produced by humans.

This paper is organized as follows: Section 3 details the methodology of LLM4Schema.org for comparing Humans and LLMs Schema.org markups. Section 4 presents our experimental study. Section 5 explains the positioning of this work compared to related works. Section 6 concludes and outlines future work.

2. Background and motivations

Schema.org [3] is a lightweight ontology that includes 806 types, 1476 properties, and 14 Datatypes [14]. It enables the description of various entities such as a person, a place, a product, an event ... Schema.org markup can be embedded in web pages using different formats; in this paper, we focus on the JSON-LD format [4].

[2]https://schema.org/docs/schemas.html, as by 23 June 2024
[3]https://schema.org/docs/schemas.html
[4]https://www.w3.org/TR/json-ld11/

```
<!DOCTYPE html>
<html lang="fr">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-
        width,␣initial-scale=1.0">
   <title>Classic American Apple Pie</title>
   <script type="application/ld+json">
{ "@context": "http://schema.org/",
  "@type": "Recipe",
  "name": "Simple␣Apple␣Pie",
  "recipeCategory": "Dessert",
  "recipeCuisine": "American",
  "recipeIngredient": [
    "6␣medium␣apples␣(Granny␣Smith␣and␣Honeycrisp␣
         mix)",
    "3/4␣cup␣sugar",
    "2␣tablespoons␣all-purpose␣flour",
    "1␣teaspoon␣ground␣cinnamon",
    "The␣Eiffel␣Tower",
    "2␣tablespoons␣unsalted␣butter",
    "2␣premade␣pie␣crusts",
    "1␣beaten␣egg␣(optional␣for␣brushing)",
    "A␣little␣sugar␣(optional␣for␣sprinkling)"
  ],}
   </script>
</head>
<body>
Preheat your oven to 375F (190C). Peel, core, and
     slice 6 medium apples (a mix of Granny Smith
     and Honeycrisp works well for flavor and
     texture). In a large bowl, mix the apple
     slices with 3/4 cup of sugar, 2 tablespoons of
      all-purpose flour, 1 teaspoon of ground
     cinnamon, and a pinch of salt. Roll out one
     premade pie crust and place it in a 9-inch pie
      dish. Trim the edge to 1/2 inch over the rim
     of the dish. Fill the crust with the apple
     mixture, and dot with 2 tablespoons of
     unsalted butter cut into small pieces. Roll
     out the second pie crust and place it over the
      filling. Trim, seal, and flute or crimp the
     edges. Cut slits in the top crust to allow
     steam to escape. Optional: For a golden crust,
      brush the top with abeaten egg and sprinkle
     with a little sugar. Enjoy your homemade
     american dessert apple pie!
</body>
</html>
```

Fig. 1. The Apple HTML web page

```
_:b0 <http://schema.org/name> "Simple Apple Pie" .
_:b0 <http://schema.org/recipeCategory> "Dessert" .
_:b0 <http://schema.org/recipeCuisine> "American" .
_:b0 <http://schema.org/recipeIngredient> "1 beaten egg (optional for brushing)" .
_:b0 <http://schema.org/recipeIngredient> "1 teaspoon ground cinnamon" .
_:b0 <http://schema.org/recipeIngredient> "2 premade pie crusts" .
_:b0 <http://schema.org/recipeIngredient> "2 tablespoons all-purpose flour" .
_:b0 <http://schema.org/recipeIngredient> "2 tablespoons unsalted butter" .
_:b0 <http://schema.org/recipeIngredient> "3/4 cup sugar" .
_:b0 <http://schema.org/recipeIngredient> "6 medium apples (Granny Smith and Honeycrisp mix)" .
_:b0 <http://schema.org/recipeIngredient> "A little sugar (optional for sprinkling)" .
_:b0 <http://schema.org/recipeIngredient> "The Eiffel Tower" .
_:b0 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://schema.org/Recipe> .
```

Fig. 2. RDF triples of the Apple Pie Markup of Figure 1

Figure 1 presents a simple example of an HTML web page describing an Apple Pie recipe. The page includes Schema.org markup in JSON-LD format between the tags `<script>`, `</script>`. This markup instantiates the `Recipe` Schema.org type to describe the content of the `<body>` section of the web page. The Schema.org markup is formatted using JSON-LD [5]. JSON-LD is a compact representation of an RDF graph using the types and properties defined in the Schema.org ontology. This markup is equivalent to the RDF representation in Figure 2.

The JSON-LD markup describes a Recipe entity along with its ingredients. For simplicity, the apple pie webpage includes one entity of one type (`Recipe`). Typically, a single web page may include many entities, and each entity can belong to multiple Schema.org types.

To enhance understanding, we will introduce the following simple definitions.

Definition 1 (Markup property and value)**.** *Given a set of triples of markup $\mathcal{T}$, let a triple $t \in \mathcal{T}$ be defined as $t = (S, P, O)$, P refers to the markup property and O to the markup value.*

Definition 2 (Markup entity)**.** *Given an RDF graph G, a markup entity, denoted by its subject s, is the set of triples that contains at least one Schema.org type and all triples reachable from s.*

[5]https://www.w3.org/TR/json-ld11/

```
  "@context": "https://schema.org",
  "@type": "Recipe",
  "name": "Homemade American Dessert Apple Pie",
  "recipeIngredient": [
    "6 medium apples (Granny Smith and Honeycrisp)",
    "3/4 cup sugar",
    "2 tablespoons all-purpose flour",
    "1 teaspoon ground cinnamon",
    "Pinch of salt",
    "2 premade pie crusts",
    "2 tablespoons unsalted butter" ],
  "recipeInstructions": [
    { "@type": "HowToStep",
      "text": "Preheat your oven to 375F (190C)." },
    { "@type": "HowToStep",
      "text": "Peel, core, and slice the apples."},
    { "@type": "HowToStep",
      "text": "In a large bowl, mix the apple slices with
          ↪ sugar, flour, cinnamon, and salt."},
    { "@type": "HowToStep",
      "text": "Roll out one premade pie crust and place it in
          ↪  a 9-inch pie dish. Trim the edge to 1/2 inch
          ↪ over the rim of the dish."},
    {"@type": "HowToStep",
      "text": "Fill the crust with the apple mixture, and dot
          ↪  with unsalted butter cut into small pieces."
          ↪ },
    {"@type": "HowToStep",
      "text": "Roll out the second pie crust and place it
          ↪ over the filling. Trim, seal, and flute or
          ↪ crimp the edges."},
    {"@type": "HowToStep",
      "text": "Cut slits in the top crust to allow steam to
          ↪ escape."},
    { "@type": "HowToStep",
      "text": "Optional: For a golden crust, brush the top
          ↪ with a beaten egg and sprinkle with a little
          ↪ sugar."},
    {"@type": "HowToStep",
      "text": "Enjoy your homemade American dessert apple pie
          ↪ !"} ],
  "recipeCategory": "Dessert",
  "recipeCuisine": "American",
  "nutrition": {
    "@type": "NutritionInformation", {
    "calories": "Varies",
    "servingSize": "1 slice",
    "fatContent": "Varies",
    "carbohydrateContent": "Varies",
    "proteinContent": "Varies" }}
```

Fig. 3. GPT3-5 Generated Schema.org markup from the Apple pie text of Figure 1

In our example in Figure 2, there is one markup entity denoted by the blank node `_:b0` that contain all triples presented in Figure 2.

Given the text of a web page, an LLM can generate a schema.org markup with a simple prompt such as:

```
Given the text above, please generate a schema.org markup in JSON-LD where
all elements adhere to the text.
```

With this prompt, GPT-3.5 produces the Schema.org markup shown in Figure 3. In this simple example, GPT-3.5 generates a fairly good markup, but the nutrition information at the end of the markup is not grounded in the text. As such, the primary challenge lies not in generating Schema.org markup using a large language model (LLM) but in assessing the *accuracy* and *completeness* of the generated markup relative to the webpage's content. In our simple motivating example, although the LLM's markup appears comprehensive compared to the recipe text, it lacks correctness.

Evaluating the correctness and the completeness of schema markup given a web page traditionally requires a ground truth. However, such ground truth does not exist and might be difficult to build. Indeed, building such a ground truth requires to consider the 806 types of Schema.org, and thousands of properties. In the example of Figure 3, we can observe that each `howToStep` is a typed entity referenced by the `Recipe` entity. As a markup is linked to text, we must consider a representative corpus of webpages with different sizes, maybe in different languages. For each page, determining if the markup is complete compared to the text is difficult, even for a human. Related works [15, 24, 34] focused on small ontologies, where the text is just a sentence and the ground truth a small set of triples. Scaling to full web pages with a large lightweight ontology raises our research question: How can we evaluate the correctness and completion of generated markup? More generally, can we trust the LLM in the task of generating Schema.org markup for web pages?

3. LLM4Schema.org Overview

To evaluate the quality of LLM-generated markup, LLM4Schema.org aims to answer a straightforward question: *Do LLMs produce better Schema.org markup than humans for real web pages?* The WebDataCommon project [7] gives access to a corpus of 1 billion real web pages that already embed Schema.org markup in JSON-LD. By comparing the Schema.org markup generated by an LLM for these pages to the existing human-generated markup, we can assess which is more complete for any sample of the corpus.

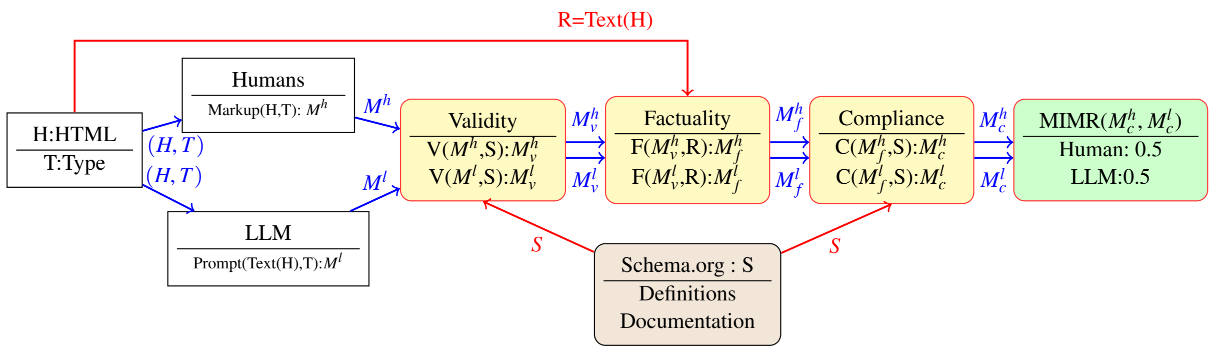


Fig. 4. LLM4Schema.org overall pipeline

```
{
  "@context": "http://schema.org/",
  "@type": "Recipe",
  "name": "Simple Apple Pie",
  "recipeCategory": "Dessert",
  "recipeCuisine": "American",
  "recipeIngredient": [
    "6 medium apples (Granny Smith and Honeycrisp
        ↪ mix)",
    "3/4 cup sugar",
    "2 tablespoons all-purpose flour",
    "1 teaspoon ground cinnamon",
    ~~"The Eiffel Tower",~~
    "2 tablespoons unsalted butter",
    "2 premade pie crusts",
    "1 beaten egg (optional for brushing)",
    "A little sugar (optional for sprinkling)"
  ],
}
```

(a) Human Markup before curation.

```
{
  "@context": "http://schema.org/",
   "@type": "Recipe",
   "name": "Simple Apple Pie",
   ~~"recipeCategory": "Main Dish",~~
   ~~"cookoo": "PT50M",~~
   "recipeInstructions": [
     {
       "@type": "HowToStep",
        "text": "Preheat oven to 375F (190C)."
     },
     {
       ~~"@type": "HowToStep",~~
       ~~"text": "Roll out one premade pie crust and
           place it in ,a 10-inch pie dish. Trim
           the edge to 1/2 inch, over the rim of
           the dish."~~
     },
     {
       ~~"@type": "Dataset",~~
       ~~"text": "Peel, core, and slice apples."~~
     }
   ],
}
```

(b) LLM Markup before curation.

Fig. 5. Illustrative Schema.org Markups for the Apple Pie Recipe. Strikeout color: blue = non-compliant properties, red = non-valid properties, green = non-factual properties.

Establishing a fair comparison is challenging; evaluating the completeness of a markup should only consider correct markup. LLMs should not gain an advantage by generating hallucinations, and there is no guarantee that humans authored the correct markup for a webpage. Therefore, only valid and accurate markup can be used for comparison.

To ensure a fair comparison between LLM-generated and human-authored markup, we used the text of the web page as the common input for both. This means that any markup fragment for either LLMs or humans that is not grounded in the webpage's text should be removed. Grounding markup in text aligns with the general recommendations of both the Schema.org [6] and Google [7].

Figure 4 outlines the overall pipeline for comparing human-generated vs. LLM-generated Schema.org markups.

[6]https://schema.org/docs/gs.html#schemaorg_expected.
[7]https://developers.google.com/search/docs/appearance/structured-data/sd-policies#content.

The pipeline begins with a webpage with Schema.org markup in JSON-LD format sampled from the WebDataCommons project [7]. The method for extracting a representative sample from this corpus is explained in Section 4.3.

From this web page, we choose a type T belonging to the page's markup. For the Apple Pie example in Figure 1, there is only one type T = `Recipe`. The objective is to compare markup entities of type T for humans and LLMs:

For humans we select markup entities of type T noted M^h. An illustrative example is presented in Figure 5a for our motivating example of Apple Pie (cf Figure 1).

For LLMs we first extract the text of the webpage (the `<body>` text in Figure 1) and ask an LLM to generate the markup M^l for the given type T. The prompt we used is detailed in Section 3.1. An illustrative example of the output is presented in Figure 5b.

For both humans and LLMs, markups presented in Figures 5a and 5b, some errors have been intentionally introduced to illustrate subsequent steps.

As we consider neither LLM-generated nor human-generated markup to be inherently reliable, we subject them to the following evaluation procedures:

1) **Validity Agent** takes a markup and the schema.org ontology as input and verifies whether the markup is syntactically correct. It returns only a valid markup. This process is detailed in Section 3.2.
2) **Factuality Agent** takes a markup and a text as input. It verifies if the markup statements are grounded in the text, checking each markup property individually, and retaining only factual statements. This process is detailed in Section 3.3.
3) **Compliance Agent** takes a markup and the schema.org documentation as input. It verifies that each markup value complies with the property's expectations as defined in the Schema.org documentation. It returns a markup with only compliant statements. This process is detailed in Section 3.4.

We consider markups that pass these agents to be correct and compare them using a scoring metric $MIMR$ detailed in Section 3.5. The idea of the metric is to merge the Human and LLM markup and evaluate each contribution to this final markup. A score of 0.5/0.5 indicates that both humans and LLMs produced half of the final markup. The MIMR metric cannot determine if a markup is complete; it can just indicate if a markup is more complete than another markup.

3.1. Schema.org Markup Generation with LLMs

Given a webpage and a set of Schema.org types, we use LLMs to generate Schema.org markup by employing prompt engineering techniques [8]. We adapted the prompt of Text2KgBench[24] to fit our context. We use the following prompt (the template on the left, and an example of the instantiated template on the right):

```
Given the Schema.org type(s),
properties, content, please write
the Schema.org JSON-LD markup that
matches the content according to the
Schema.org type(s).
Only output JSON-LD markup.
The Schema.org types:
<types> [TYPES] </types>
Properties for [TYPES]:
<properties>[PROPS]</properties>
Example [INDEX] for [TYPE]:
- Example content:
<ex_content>[ECONTENT]</ex_content>
- Example markup:
<ex_markup>[EMARKUP]</ex_markup>
The content:<content>[CONTENT]</content>
```

```
Given the Schema.org type(s), properties, content, please
write the Schema.org JSON-LD markup that matches the content
according to the Schema.org type(s).
Only output JSON-LD markup.
The Schema.org types:
<types> Recipe </types>
Properties for Recipe:
<properties>cookTime, cookingMethod, nutrition,
RecipeCetagory, RecipeCuisine,...</properties>
Example 1 for Recipe:
- Example content:
<ex_content> Mom's World Famous Banana Bread By John Smith,
May 8, 2009
This classic banana bread recipe comes from my mom - the
walnuts add a nice texture and flavor to the banana bread.
Prep Time: 15 minutes
Cook time: 1 hour </ex_content>
 - Example markup:
<ex_markup> { "@context": "https://schema.org",
"@type": "Recipe",
"author": "John Smith",
"cookTime": "PT1H", "description": "This classic banana
bread recipe comes from my mom - the walnuts add a nice
texture and flavor to the banana bread.",} </ex_markup>
The content:<content>Preheat your oven to 375F (190C).
Peel, core, and slice 6 medium apple... Enjoy your homemade
american dessert apple pie! </content>
```

– `TYPES` represents the types of the entity. In our example, there is only one `Recipe`.
– `PROPS` represents the properties that can be filled for TYPES, including supertypes. For `Recipe`, the list of properties includes `cookTime`, `cookingMethod` etc, in addition to the properties of its supertypes `HowTo`, `CreativeWork`, and `Thing`.

INDEX represents the index of the example; there can be several examples for one type.
– `ECONTENT` and `EMARKUP` represent one example from the Schema.org documentation for a type. For our example, this corresponds to the example at the bottom of https://schema.org/Recipe. `ECONTENT` is the text of the example, and `EMARKUP` is the JSON-LD markup for the text of the example.
– `CONTENT` is the text of the page to annotate, i.e., the text of the web page shown in Figure 1.

To address the limitation of an LLM's context window, such as the 16,385 token context window and 4,096 token output limit of GPT-3.5-turbo, we divide the text TXT into chunks while maintaining a 10% overlap between adjacent chunks to preserve context. The goal is to minimize the number of chunks C and maximize the number of tokens in each chunk to ensure context preservation. The number of chunks is estimated using the following formula:

$$chunk\_tokens = context\_window - output\_tokens - |Tokens(prompt)|$$

$$C = \frac{|Tokens(TXT)|}{0.90 \times chunk\_tokens} \qquad \text{account for 10\% overlap}$$

Subsequently, we merge the JSON-LD output from each chunk to create the final markup for the text. For the sake of simplicity, a mock example of the results of LLM markup generation is illustrated in Figure 5b.

3.2. Schema.org Markup Validity Agent

Existing validation tools like Google's Rich Results Test [8] or Schema Markup Validator [9] typically identify invalid properties and values in the markup, prompting human annotator to make corrections. For instance, consider the LLM-generated markup shown in Figure 5b at the line 6: the property "cookoo" is not a valid property of `schema:Recipe`.

These tools suffer from two limitations: (1) they are designed for human interaction and lack an API for automation, (2) their validation rules are not transparent, making it difficult to understand their inner workings.

To address these limitations, we developed a simple validator based on shape constraints validation. Schema.org is intentionally designed to be "semantically imperfect and not formally strict" in order to facilitate the vocabulary development [30], as well as the adoption by the community [10]. As a result, we only check for the following rules:

(1) **The type must be defined in Schema.org ontology.** For example, `Recipe` is a valid type, but `Recette` is not.
(2) **The property must be defined in the type's ontology or inherited from parent types.** For example, `recipeIngredient` is a valid property for `Recipe`, but `cookoo` is not. Additionally, `name` is a valid property for `Recipe` because it is inherited from `Thing`.
(3) **The value must approximately match the expected property type.** For instance, the value of `recipe-Instruction` can be either `CreativeWork`, `Text` or `ListItem`. In our example, `HowToStep` entity (child type of `CreativeWork`), the string `"Preheat oven ..."` are valid, but the `Dataset` entity is not.

Based on these rules, we generated SHACL [11] shapes for the entire Schema.org ontology [12], following these simple steps for each shape: (1) Use OWL, RDFS terms for inference. (2) Propagate all properties from parent type to children types. (3) Close the shapes to enable reasoning under Closed World Assumption.

[8]https://search.google.com/structured-data/testing-tool
[9]https://validator.schema.org/
[10]https://schema.org/docs/datamodel.html#Conformance
[11]https://www.w3.org/TR/shacl/
[12]https://github.com/schemaorg/schemaorg/blob/main/data/releases/23.0/schemaorg-all-http.nt

```
schema1:Recipe a schema1:CreativeWork, schema1:HowTo, schema1:Thing, rdfs:Class, sh:NodeShape ;
   rdfs:label "Recipe" ;
   rdfs:subClassOf schema1:CreativeWork,
      schema1:HowTo,
      schema1:Thing ;
   sh:closed true ;
   sh:ignoredProperties ( rdf:type owl:sameAs ) ;
   sh:property schema1:CreativeWork-about,
      schema1:CreativeWork-video,
      schema1:CreativeWork-workExample,
      schema1:CreativeWork-workTranslation,
      schema1:HowTo-estimatedCost,
      schema1:HowTo-performTime,
      schema1:HowTo-prepTime,
      schema1:HowTo-yield,
      schema1:Recipe-cookTime,
      schema1:Recipe-recipeCategory,
      schema1:Recipe-recipeCuisine,
      schema1:Recipe-recipeIngredient,
      ...

schema1:Recipe-recipeIngredient a sh:PropertyShape ;
   rdfs:label "recipeIngredient" ;
   rdfs:comment "A single ingredient used in the recipe, e.g. sugar, flour or garlic." ;
   rdfs:subPropertyOf schema1:supply ;
   sh:or ( [ sh:class schema1:URL ] [ sh:datatype xsd:string ] [ sh:class schema1:XPathType ] [ sh:class
        schema1:Text ] [ sh:class schema1:PronounceableText ] [ sh:class schema1:CssSelectorType ] ) ;
   sh:path schema1:recipeIngredient .
```

Fig. 6. Example of SHACL shape for `Recipe` type (excerpt).

Figure 6 shows an excerpt of the resulting SHACL shape[13] for the `Recipe` type.

We use RDFLib pySHACL [14] to validate the markup against these shapes, subsequently removing any invalid property-value pairs from the markup.

3.3. *Schema.org Markup Factuality Agent*

The Factuality agent takes a text and a markup as inputs, verifying whether the properties and values mentioned in the markup are grounded in the webpage text. It is LLM based. While it may seem surprising to verify the output of an LLM with another LLM, the precision of LLMs varies depending on the task. The initial task in this paper is to generate a complete markup starting from a text, which is quite complex. The Factuality Agent's task is simpler; it merely verifies that each markup fragment is grounded in the text. We utilize the following prompt:

```
You are an expert in the semantic web and have deep knowledge about
writing Schema.org markup. You will be given a document, a question and
a series of hints. Your task is to give an answer using the hints. Reply
using only Yes or No.
Given the document below:
<content>[CONTENT]</content>
Hint 1: Check whether the [VALUE] is in the text.
Hint 2: Check whether the [VALUE] is [PROP].
Is [VALUE] a [PROP] of [TYPE]?
```

If the Factuality agent rejects a markup property, this signifies the detection of a hallucination in LLM-generated markup. In our context, there are two possible types of hallucination as defined in [17]:

1. **Intrinsic Hallucination**: The property-value-type triple contradicts the webpage content. For example, in the markup of Figure 5b, the how-to step of line 14 indicates a 10-inch pie, while it is a 9-inch pie in the text.

[13]https://github.com/GDD-Nantes/LLM4SchemaOrg/tree/main/schemaorg/shacl
[14]https://github.com/RDFLib/pySHACL.git

2. **Extrinsic Hallucination**: The property-value-type triple cannot be supported or contradicted by the webpage content. For example, in the markup of Figure 5b, the pair `{"text": "Main Dish"}` (line 5) is not supported by the webpage content.

For each property in the markup, we instantiate the prompt template as follows:

```
Given the document below:
<content>Preheat your oven to 375F (190C). Peel, core, and slice 6 medium
apple... Enjoy your homemade american dessert apple pie! </content>
Hint 1: Check whether the [Main Dish] is in the text.
Hint 2: Check whether the [Main Dish] is [RecipeCategory].
Is [MainDish] a [RecipeCategory] of [Recipe]?
```

Because the prompt includes the entire web page text, it may encounter the token limit in LLMs, where the text's size may exceed the maximum token count supported by LLMs. To address this issue, we employ a technique of text chunking with overlaps, similar to our method for markup generation (Section 3.1). We aim also to minimize the number of chunks C, i.e., maximize the number of tokens in each chunk to preserve context. The number of chunk is estimated using the modified formula from Section 3.1:

$$chunk\_tokens = max\_tokens - |\max_{i=1}^{P} Tokens(p_i)| \qquad \text{P is a set of prompts}$$

$$C = \frac{|Tokens(T)|}{0.90 \times chunk\_tokens} \qquad \text{account for 10\% overlap}$$

Note that the number of output tokens is omitted from the equation because the model only replies with "yes" or "no", making the number of output tokens negligible compared to other terms. The Factuality agent then validates each chunk, producing a boolean vector for each chunk. The final factuality score is computed as the element-wise logical OR of these vectors. Finally, we remove the invalid property-value-type triples from the markup.

3.4. Schema.org Markup Compliance Agent

The Compliance agent takes a markup as input and ensures that each property's value aligns with the ontology expectations outlined in the Schema.org documentation. For example, the expected value for `recipeIngredient` is defined as *"A single ingredient used in the recipe, e.g., sugar, flour, or garlic."* [15]. When analyzing the human-generated markup depicted in Figure 5a at line 12, it becomes evident that *"The Eiffel Tower"* does not constitute a complaint recipe ingredient. The Compliance agent is LLM-based; we rely on language comprehension of LLMs to verify that property values are compliant with Schema.org expectations described in the documentation. We use the following prompt:

```
You are an expert in the semantic web and have deep knowledge about
writing Schema.org markup. You will be given a JSON-LD markup and a
Schema.org definition for a property and your task is to assert whether
the markup align with the given definition. Answer Yes if the value aligns
with the definition; if not, answer No. Reply using only Yes or No.
Given the markup below for property [PROP]:
<markup>[MARKUP]</markup>
Given the property definition below for [PROP]:
<definition>[DEFINITION]</definition>
Does the value align with the property definition?
```

`[PROP]` is a placeholder for a property of the input markup, and `[DEFINITION]` is replaced by the expectation for this property in the Schema.org documentation.

For each property in the markup, we send a query to the LLM, for instance:

[15]https://schema.org/recipeIngredient

Algorithm 1: Compute MIMR Score between Human and LLM Markup

```
Data: Human markup $H$, LLM markup $L$
Result: $(score_h, score_l)$
1 $C_h \leftarrow \emptyset, C_l \leftarrow \emptyset, C_m \leftarrow \emptyset$ ;
2 foreach $p \in (H \vee L)$ do
3     $C_h[p] = |\{o|(s,p,o) \in H\}|$ ;
4 foreach $p \in (H \vee L)$ do
5     $C_l[p] = |\{o|(s,p,o) \in L\}|$ ;
6 foreach $p \in (C_h[p] \vee C_l[p])$ do
7     $C_m[p] = max(C_h[p], C_l[p])$ ;
8 $(score_h, score_l) \leftarrow (J_M(C_h, C_m), J_M(C_l, C_m))$ ;
```

	C_h	C_l	C_m
type	1	2	2
name	1	1	1
recipeCategory	1	0	1
recipeCuisine	1	0	1
recipeIngredient	8	0	8
recipeInstruction	0	1	1
text	0	1	1
TOTAL	12	5	15

Fig. 7. Computing Scores

```
Given the markup below for property [RecipeIngredient]:
<markup> recipeIngredient "the Eiffel Tower" </markup>
Given the property definition below for [RecipeIngredient]:
<definition> A single ingredient used in the recipe, e.g., sugar or
garlic. </definition>
Does the value align with the property definition?
```

This prompt is instantiated for each property-value pair in the markup. By analyzing the LLM responses, we identify and remove any non-compliant property-value pairs from the markup.

3.5. MIMR: Markup Ideal Match Ratio

We designed the MIMR metric to estimate the contribution of human and LLM-generated markups to an ideal merged markup. This metric calculates the percentage of markup contributed by humans/LLM in the final merged markup. For example, applying the MIMR metric to the curated markup presented in Figures 5a and 5b, the MIMR yields scores of $(0.8, 0.33)$, where 0.8 is the ratio contributed by humans and 0.33 by LLMs. In this case, humans win the match.

Algorithm 1 details how the MIMR metric is computed.

For each property p in either human markup H or LLM markup L, we count the number of the occurrences of the property and store these counts in $C_h[p]$ and $C_l[p]$ respectively.

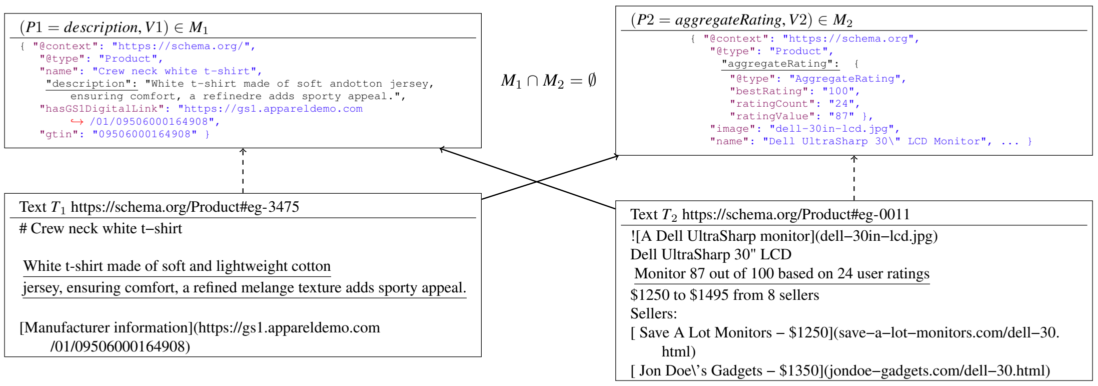
Then, for each property p in the union of $C_h[p]$ and $C_l[p]$, we determine the maximum count and store it in $C_m[p]$. We choose properties with larger counts to represent the ideal markup. The values of C_h, C_l, C_m for our motivating example are represented in Table 7.

The final scores $(score_h, score_l)$ are computed using the generalized Jaccard similarity coefficient J_M defined as:

$$J_M(C_a, C_b) = \frac{\sum_p min(C_a[p], C_b[p])}{\sum_p max(C_a[p], C_b[p])}$$

This allows us to evaluate the contribution of humans and LLMs to the merged markup. For our example, the resulting scores are : $score_h = \frac{12}{15} = 0.8$ and $score_l = \frac{5}{15} \approx 0.33$. This indicates that the human-generated markup is closer to the ideal merged markup I than the LLM-generated markup.

The MIMR metric remains coarse-grained and purely quantitative. Given two markup M_1, M_2, the MIMR metric might indicate $M_1 <_{MIMR} M_2$, however, from the qualitative perspective M_1 can be considered better than M_2. For instance, a markup with fewer properties might include properties that are considered more important. In our experiments, we assess the qualitative perspective of the MIMR metric using human evaluations.


$(P1 = description, V1) \in M_1$
$(P2 = aggregateRating, V2) \in M_2$
$M_1 \cap M_2 = \emptyset$
Text T_1 https://schema.org/Product#eg-3475
Text T_2 https://schema.org/Product#eg-0011

```
{ "@context": "https://schema.org/",
  "@type": "Product",
  "name": "Crew neck white t-shirt",
   "description": "White t-shirt made of soft andotton jersey,
      ensuring comfort, a refinedre adds sporty appeal.",
  "hasGS1DigitalLink": "https://gs1.appareldemo.com
      ↪ /01/09506000164908",
  "gtin": "09506000164908" }
```

```
{ "@context": "https://schema.org",
  "@type": "Product",
   "aggregateRating":  {
    "@type": "AggregateRating",
    "bestRating": "100",
    "ratingCount": "24",
    "ratingValue": "87" },
  "image": "dell-30in-lcd.jpg",
  "name": "Dell UltraSharp 30\" LCD Monitor", ... }
```

# Crew neck white t–shirt

White t-shirt made of soft and lightweight cotton
jersey, ensuring comfort, a refined melange texture adds sporty appeal.

[Manufacturer information](https://gs1.appareldemo.com
/01/09506000164908)

![A Dell UltraSharp monitor](dell–30in–lcd.jpg)
Dell UltraSharp 30" LCD
Monitor 87 out of 100 based on 24 user ratings
$1250 to $1495 from 8 sellers
Sellers:
[Save A Lot Monitors – $1250](save–a–lot–monitors.com/dell–30.
html)
[Jon Doe\'s Gadgets – $1350](jondoe–gadgets.com/dell–30.html)

Fig. 8. Example of ground truth generation for Extrinsic Hallucination.

4. Experiments

The experimental study aims to address the following questions:

1 How reliable is the Factuality agent? As the Factuality agent relies on LLM prompts, we have to evaluate the accuracy of the Factuality agent.
2 How reliable is the Compliance agent? As The Compliance agent relies on LLM prompts, we have to evaluate the accuracy of the compliance agent.
3 How can we obtain a representative sample of the Schema.org corpus?
4 Is the MIMR metric reliable for comparing markups? The MIMR metric is a purely quantitative metric. Between 2 markups, does humans choose the same one than the MIMR metric?
5 Are LLM-generated markup better than human-generated markup?

All experimental results and the code for reproducibility are available on the project website [16].

4.1. *How reliable is the Factuality agent?*

As the Factuality agent is based on an LLM, we must assess its accuracy in detecting the presence or absence of property-value-type triples in the input text.

4.1.1. *Ground truth dataset for Factuality agent*

We build a ground truth based on the many examples provided in the Schema.org documentation [17]. Each example associates a JSON-LD Schema.org markup M with a concise text snippet τ. Figure 8 illustrates two such examples denoted (M_1, τ_1) and (M_2, τ_2). As we rely on the official documentation of Schema.org, we assume that each property-value-type triple $(property, value, type)$ in M is present in τ. For example, the pair $(P_1 = \textit{"description"}, V_1 = \textit{"white t-shirt..."}, T_1 = Product)$ is found in τ_1, and this is true for all properties of M_1.

By reusing examples from the Schema.org documentation, we can generate 785 positive tests. To generate negative examples, we follow two principles:

Intrinsic Principle: For a pair $((P_i, V_i, T_i), \tau)$, we modify the numeric values from V_i to V_j, where $i \neq j$ to avoid matching these values in τ. Consequently, we generated $((P_i, V_j, T_i), \tau)$ such that (P_i, V_j, T_i) have no corresponding match in τ. We generated 498 negative examples from the 785 positive examples using this principle.

Extrinsic Principle: For two pairs $((P_1, V_1, T_1) \in M_1, \tau_1)$ and $(M_2 = (P_2, V_2, T_2) \in M_2, \tau_2)$ where M_1 and M_2 are disjoint, we create new pairs $((P_1, V_1, T_1), \tau_2)$ and $((P_2, V_2, T_2), \tau_1)$, as presented in Figure 8. When M_1 and M_2 are disjoint, there is no match for every property of M_1 in τ_2 and for every property of M_2 in τ_1. Thanks to this principle, we generated 630 negative examples from the 785 positive examples.

Table 1
Number of examples in the ground-truth dataset for Factuality agent

	Positive	**Negative**	**Total**
Factual (Intrinsic)	785	498	1283
Factual (Extrinsic)	785	630	1415

Table 2
Precision, Recall, and F1 of the Factuality agent

	Prec.	**Recall**	**F1**
Factual (Intrinsic)	0.866	0.938	0.901
Factual (Extrinsic)	0.945	0.945	0.945

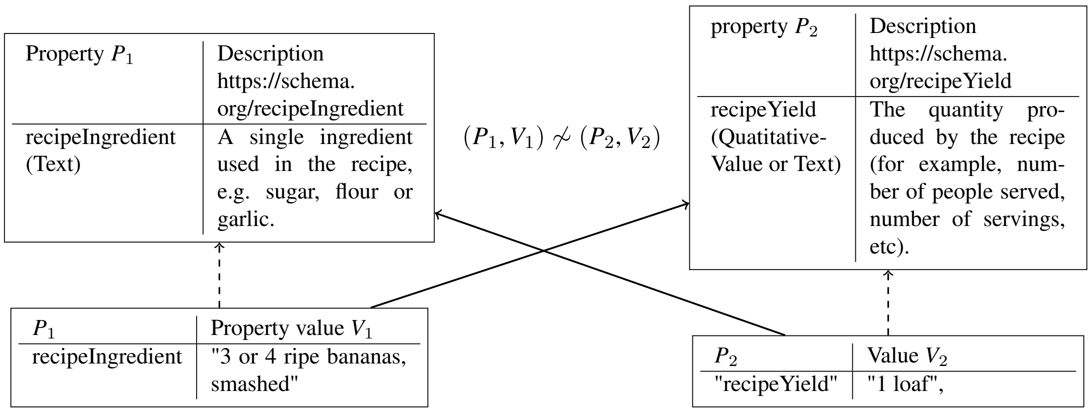

Property P_1
Description https://schema.org/recipeIngredient
recipeIngredient (Text)
A single ingredient used in the recipe, e.g. sugar, flour or garlic.
$(P_1, V_1) \not\sim (P_2, V_2)$
property P_2
Description https://schema.org/recipeYield
recipeYield (Quatitative-Value or Text)
The quantity produced by the recipe (for example, number of people served, number of servings, etc).
P_1
Property value V_1
recipeIngredient
"3 or 4 ripe bananas, smashed"
P_2
Value V_2
"recipeYield"
"1 loaf",

Fig. 9. Example of ground truth generation for Compliance agent.

The Factuality ground-truth consists of 785 positive examples, 498 negative intrinsic examples, and 630 negative extrinsic examples, as shown in Table 1. The ground-truth dataset is available on our GitHub repository[18].

4.1.2. Experimental results for the Factuality agent

We evaluate the precision, recall, and F1 score for the factuality agent using its ground-truth dataset described in the previous section.

For evaluation purposes, We used a quantized version [19] of Mixtral-8x7B-Instruct [18] to mitigate evaluation costs associated with using using GPT-3 or GPT-4. We set the temperature to 0.0 when prompted to ensure a deterministic response from the model. The agent produces a numerical value between 0 and 1, where 1 indicates full factual adherence. We classify property-value-type triples as `VALID` if their score is $\geqslant$ `threshold`, and `INVALID` otherwise. Naturally, the choice of threshold significantly impacts evaluation outcomes. We opted for a threshold of 0.5, aligning with the interpretation of SelfCheckGPT-Prompt [20], where property-value-type triples may exhibit some factual inconsistencies but are overall factually correct. We refer the reader to Appendix F for more details on the implementation of the Factuality agent.

Table 2 presents the precision, recall, and F1 score of the Factuality agent. The Factuality agent obtains high F1 scores in intrinsic and extrinsic test cases. Higher precision for extrinsic cases indicates that detecting hallucinations with evidence in the text is more challenging, as the task requires a deeper understanding of the text, leading to higher false positives (see Appendix A).

4.2. How reliable is the Compliance agent?

Given that the Compliance agent is based on an LLM, it is crucial to assess its accuracy in determining whether a property value complies with the property expectations specified in the Schema.org documentation. Similarly to the Factuality agent, this evaluation relies on the many examples provided in the Schema.org documentation.

[16]https://github.com/GDD-Nantes/LLM4SchemaOrg/
[17]https://github.com/schemaorg/schemaorg/blob/main/data/releases/23.0/schemaorg-all-examples.txt
[18]https://github.com/GDD-Nantes/LLM4SchemaOrg/tree/main/schemaorg/examples/factual
[19]https://huggingface.co/TheBloke/Mixtral-8x7B-Instruct-v0.1-GGUF

Table 3
Statistics of the ground truth for Compliance agent

	Positive	Negative	Total
Compliance	785	147	932

Table 4
Precision, Recall, and F1 of the Compliance agent

	Prec.	Recall	F1
Compliance	0.9	0.914	0.907

4.2.1. *Ground truth dataset for Compliance agent*

From the examples in the Schema.org documentation, we consider pairs (P, V) where P is the description of the property in natural language, and V is a value available in Schema.org examples for the property P. According to the Schema.org documentation, we consider that a value V is compliant with the description of the property P.

Figure 9 illustrates two compliance pairs: one for the `recipeIngredient` property and the other for `recipeYield` property. In these examples, V_1 is compliant with P_1 and V_2 is compliant with P_2.

In addition to the positive examples, we generate negative examples by swapping values between textual properties that are semantically distant. For example, V_1 is considered not compliant with P_2 and V_2 is considered not compliant with P_1. To ensure that (P_1, V_1) is distant from (P_2, V_2) during negative test generation, we measure the semantic distance between their embeddings using Spacy [20] and retain pairs with the largest distances as negative examples.

As described in Table 3, we generated 932 tests with positive and negative examples. The ground truth dataset is available on our GitHub repository [21].

4.2.2. *Experimental results of the Compliance agent*

We evaluated the Compliance agent's precision, recall, and F1 score using its ground truth datasets. As for the Factuality agent, we used a quantized version [22] of Mixtral-8x7B-Instruct [18]. The outputs of the agent are numbers between 0 and 1, where 1 indicates full compliance of property-value pairs. We assign the label `VALID` to the property-value pairs with a score $\geqslant$ `threshold`, and `INVALID` otherwise. Naturally, the results are sensitive to the choice of the threshold. As before, we chose the threshold of 0.5 following the interpretation of SelfCheckGPT-Prompt. We refer the reader to Appendix F for more details on the implementation of the Compliance agent.

Table 4 presents the precision, recall, and F1 score of the Compliance agent. High Precision and Recall indicate that the Compliance agent performs well on positive and negative tests (See Appendix B). A high F1 score also means we can safely integrate the Compliance agent into our pipeline to evaluate the quality of the generated markup.

4.3. *Sampling over WebDataCommons*

For practical reasons, we consider a representative sampled subset of the corpus consisting of 877 million web pages visited in 2022 that feature Schema.org markup in JSON-LD format. This corpus is extracted from the CommonCrawl of October 2022 dataset [23], which contains 3.15 billion web pages. Of these, 1.5 billion pages contain structured data [24], and 877M include Schema.org markup in JSON-LD format.

The corpus is available in RDF as quad files using the format *subject*, *predicate*, *object*, *URL*, where *URL* is the *URL* of the visited web page, *predicate* is a Schema.org predicate, and the subject can be either a URL or a Blank node.

Sampling over these 877 million pages is challenging because some Schema.org types/properties are much more frequent and better described than others [10]. Sampling only frequent Schema.org types or well-described entities may bias results in favor of LLMs.

To create representative samples, we rely on ideas explored by Schema.org observatory [10]. Schema.org observatory has computed characteristic sets for WebDataCommons released in October 2021. Characteristic sets (C-sets)

[20]https://spacy.io/
[21]https://github.com/GDD-Nantes/LLM4SchemaOrg/tree/main/schemaorg/examples/compliance
[22]https://huggingface.co/TheBloke/Mixtral-8x7B-Instruct-v0.1-GGUF
[23]https://data.commoncrawl.org/crawl-data/CC-MAIN-2022-40/index.html
[24]http://www.webdatacommons.org/structureddata/#results-2022-1

Table 5
Example of C-sets from WebDataCommons 2022.

C-set	#instances	#properties	URLs
address, alternateName, description, image, logo, name, sameAs, telephone, url, isa:<NewsMediaOrganization>	36	10	https://sports442.com, http://www.bet88news.net
areaServed, contactPoint, description, image, knowsAbout, legalName, logo, name, telephone, url, isa:<Organization>	1	6	https://mridt.com/
aggregateRating, description, isSimilarTo, model, name, offers, isa:<Product>	1	7	https://www.destination-fougeres.bzh/loisirs/cinema-le-club/, https://www.ardennes.com/activite-sportive/balade-a-cheval/
address, contactPoint, description, name, openingHours, url, isa:<EmergencyService>	3	7	https://www.annarbortow.com/, https://www.annarbortow.com/about.html, https://www.annarbortow.com/services.html
name, sameAs, isa:<HowToSection>	6	3	https://breadboozebacon.com/tag/pasta/#id

are sets of properties that are shared by entities across web pages, revealing how humans combine properties to describe web entities. We extend this work by adding a web page source (URL) for each C-set, allowing us to extract the textual content and human-generated JSON-LD markup for each C-set. Formally, a C-set for a subject s in a quad s, p, o, u is defined as:

$$S_C(s) = \{p | \exists o : (s, p, o, u) \in D\}$$

where (s, p, o, u) represents quads in the dataset D. For example, applying this definition using the markup in Figure 5a yields a C-set of length 5: { `isa:Recipe`, `name`, `recipeCategory`, `recipeCuisine`, `recipeIngredient` }.

For the corpus of 877 million pages, we computed C-sets using Equation 4.3. Within each C-set, we removed invalid properties and types (outside Schema.org, typographic errors), resulting in a total of 1.2 million C-sets. Table 5 shows some examples of C-sets from WebDataCommons 2023. The entire C-sets dataset is available on Zenodo [25].

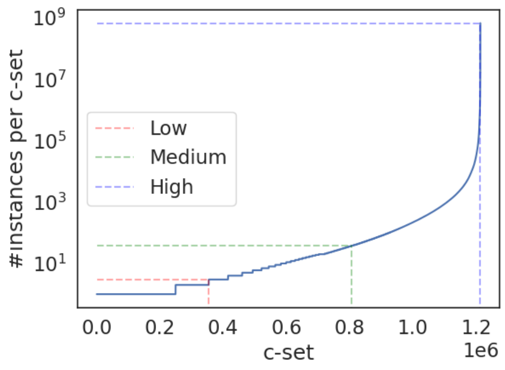
To select representatives C-sets, we study the C-sets using two features: (1) **The number of instances per C-set** measures how frequently a given combination of properties occurs. (2) **The number of properties per C-set** indicates how many properties are combined in a given combination.

We observe a very weak monotonic relationship between the number of instances and the number of properties in a C-set (Spearman $\rho = -0.18$, $p\text{-}value = 0.0$). This aligns with the observations made in the Schema.org observatory [10], where longer combinations of properties tend to have fewer instances, but this is not the case for all types. In other words, longer C-sets do not necessarily share the same instances as C-sets with higher instance counts and vice versa; hence, both features need to be sampled independently.

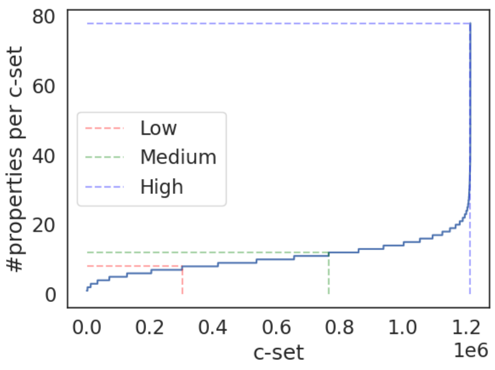
Next, we divided the distribution of the number of instances and the number of properties into 3 quantiles: Low, Medium, and High (Figure 10). This ensures a representative sample of C-sets with different lengths and cardinalities. For each feature, we grouped the C-sets by quantiles and sampled 30 pages from each, resulting in 180 pages in total.

For each webpage, we extracted the textual content from fully-rendered webpages using HTML2Text [26], and the human JSON-LD markup using `extruct` [27]. It is important to note that: (i) A webpage can have multiple JSON-LD markup and (ii) the type in the C-set is not necessarily the same as the type in the JSON-LD markup. For

[25]https://doi.org/10.5281/zenodo.12080148
[26]https://github.com/Alir3z4/html2text/.
[27]https://github.com/scrapinghub/extruct.



(a) #instances per C-set



(b) #properties per C-set

Fig. 10. Distribution of C-sets in WebDataCommons 2023.

Table 6

Statistics of the 180 web pages. PV = #property-value pairs (min, avg, max), DP = #distinct Properties, DT = #Distinct Types, DS = Document Size (min/avg/max in KB).

		PV	**DP**	**DT**	**DS**
number of instances	**Low**	4/27/193	155	26	2/23/78
	Medium	4/41/598	131	16	0.2/13/45
	High	3/85/924	180	19	2/22/84
number of properties	**Low**	1/75/1600	109	25	2.5/20/214
	Medium	8/28/168	97	20	1.5/19/73
	High	15/68/1000	147	20	2/17/53
	Overall	1/53/1600	364	78	0.2/19/214

Table 7

Results throughout the evaluation pipeline, where Input is the number of triples in the input. Valid, Fact, and Comp are the number of triples resulting from the step. The Rejection Rate is the percentage of triples rejected by the pipeline. MIMR: h for Human, l for LLMs.

	Input	Valid.	Fact.	Comp.	Rejection Rate	MIMR
Human	5690	4875	3315	2719	52.2%	
GPT-3.5	4190	3369	2489	2055	50.9%	($h =$ **0.687**, $l = 0.585$)
GPT-4	5260	4613	3573	3113	40.8%	($h = 0.568$, $l =$ **0.707**)

example, in the markup in Figure 5b, the C-set is { `isa:HowToStep`, `text` } but the markup is of type `Recipe`. When extracting the JSON-LD markup, we only retained those containing the C-set and considered the root type, i.e., the type on the first level in the markup, as input for the evaluating pipeline. This step provides more context to the LLMs while retaining representativeness with respect to the C-sets. In another word, instead of evaluating the markup { `"@type": "HowToStep", "text": "Preheat oven to 375F (190C)."` }, we evaluate the entire markup of type `Recipe` in Figure 5b.

Table 6 presents the statistics about our 180 webpages. We can see some diversity in the length of web pages and number of triples per page.

Table 8

Results throughout the evaluation pipeline, per feature (number of instances and number of properties), and per quantile (Low, Medium, High). The Input, Valid, Factuality (Fact.), and Compliance(Comp.) rows show the number of triples after each step. The MIMR for Humans (h) and LLMs (l), respectively. The Rejection Rate (RR) is the percentage of triples rejected by the pipeline.

(a) Number of instances per C-set

	Human			**GPT-3.5**			**GPT-4**		
	Low	Med	High	Low	Med	High	Low	Med	High
Input	910	861	1039	1293	572	401	1286	619	671
Valid.	748	731	812	781	452	373	1064	592	619
Fact.	463	488	602	589	245	284	819	469	513
Comp.	402	346	515	549	222	256	683	441	448
RR	55%	59%	50%	57%	61%	36%	46%	28%	33%
MIMR				($h = 0.603$, $l = \mathbf{0.669}$)	($h = \mathbf{0.689}$, $l = 0.577$)	($h = \mathbf{0.802}$, $l = 0.506$)	($h = 0.491$, $l = \mathbf{0.795}$)	($h = 0.576$, $l = \mathbf{0.673}$)	($h = \mathbf{0.653}$, $l = 0.638$)

(b) Number of properties per C-set

	Human			**GPT-3.5**			**GPT-4**		
	Low	Med	High	Low	Med	High	Low	Med	High
Input	535	740	1605	812	567	545	849	897	1605
Valid.	369	687	1528	793	474	496	778	713	1528
Fact.	273	457	1032	645	339	387	629	506	1032
Comp.	216	373	867	381	281	366	525	444	867
RR	59%	49%	45%	53%	50%	32%	38%	50%	45%
MIMR				($h = 0.585$, $l = \mathbf{0.643}$)	($h = \mathbf{0.687}$, $l = 0.618$)	($h = \mathbf{0.772}$, $l = 0.480$)	($h = 0.434$, $l = \mathbf{0.797}$)	($h = 0.569$, $l = \mathbf{0.758}$)	($h = \mathbf{0.704}$, $l = 0.567$)

4.4. Humans Vs LLMs

We compare human-generated markup to GPT-generated markup (GPT-3.5-Turbo-16k-0613 and GPT-4-0125-preview)[28] for 180 web pages, using the pipeline in Figure 4. According to the MIMR metric, human-generated markup contributes more to the ideal merge than GPT-3.5-generated markup, but less than GPT-4-generated markup. This is due to GPT-4 being better at following instructions [2], generating more triples with a lower rejection rate than GPT-3.5.

Table 7 presents the total number of RDF triples present in the output of each agent for the whole 180 web pages. For example, for humans, on the 5690 input triples, the validation agent only retained 4875 triples. When examining the proportion of rejected triples, 52.2% of human-generated triples are rejected. While most of these triples are valid, they are not factual or compliant. This mainly indicated the text of web pages should be improved to ground information available in the markup. This may concern information that is only available as images with no alternative text, for example. As for LLMs, 50.9% of GPT-3.5 generated markup, and 40.8% of GPT-4 generated markup were incorrect. This finding indicates that LLMs should not be used out-of-the-box to generate Schema.org markup and require curation to ensure the quality of the generated markup.

Table 8 presents the results throughout the evaluation pipeline per feature (number of instances and number of properties) and per quantile (Low, Medium, High). We observe the same pattern in both features: the MIMR metric is higher for Humans in the High quantiles, while it is higher for GPT-4 in the Low quantiles. Although LLMs

[28]We also tried with Mixtral but had a problem getting JSON-LD outputs.

Table 9

Contingency tables showing the number of votes for Human and MIMR for different generator models (GPT-3.5 and GPT-4).

	GPT-3.5		**GPT-4.5**	
	Human	MIMR	Human	MIMR
Markup A (human-generated)	13	12	9	7
Markup B (GPT-generated)	8	7	14	12
Cohen's Kappa	0.750		0.895	

cannot outperform Humans on web pages with the highest number of instances/properties, they can help improve web pages in the Low and Medium quantiles. This finding suggests that LLMs can help generate the first draft of Schema.org markup, which humans can further curate to improve its quality.

The in-depth analysis of the errors made by Humans and LLMs is presented in Appendix C.

4.5. Evaluating the accuracy of the MIMR metric

Judging the quality of the generated markup is a challenging task, as there might be multiple valid ways to represent the same information. The MIMR metric measures the quantity of information that contributes to the ideal merge of human and LLM-generated markups, but it does not capture the quality of the information. To evaluate the accuracy of our MIMR metric, we conducted a human evaluation by measuring the perceived quality by humans. This is done in three steps: (1) we randomly selected a subset of web pages from the dataset, (2) we asked human evaluators to compare the human-generated markup with the LLM-generated markup for each web page, and (3) we measured the MIMR-Human agreement.

We selected 10% random web pages to validate the scoring metric. Then, 7 participants were presented with 2 curated markups of the same web page and were asked to choose between A: human-generated, B: GPT-generated, or Tie [29]. Note that the participants were not aware of the markup's origin. Table 16 shows the participants' responses.

From the responses, we counted the votes for "Markup A" and "Markup B" and then added 1 vote to both "Markup A" or "Markup B" whenever a participant votes "Tie." Table 9 shows the total vote count for Humans and MIMR for different generator models (GPT-3.5 and GPT-4). To determine whether the MIMR metric is consistent with human preferences, we measure the inter-rater reliability using Cohen's Kappa coefficient following [13]. The high Kappa statistic indicates a substantial agreement between the human judgments and the MIMR metric.

5. Related Work

In this paper, we propose LLM4Schema.org, a comprehensive pipeline leveraging large language models (LLMs) and prompting techniques to generate Schema.org markup for web pages. The pipeline employs two LLM-based agents to ensure the factual accuracy and compliance of the generated markups, as well as to evaluate their reliability. The LLM-generated markups are curated and compared to human-generated markups.

The most related works to LLM4Schema.org pertain to LLM-augmented Knowledge Graph Construction (LLM-KGC) [19, 26], where knowledge graphs (KGs) are built using LLMs and prompting from raw texts. These include PiVE [15], AutoKG [34] and Text2KGBench [24]. These approaches rely on relatively simple ontologies and work at the sentence level of granularity. For instance, Text2KGBench targets small-sized ontologies by design (up to 20 types and 68 relations), using short text to accommodate the token size limitations of LLMs and validating LLM-generated triples against manually constructed ground truths. Additionally, KGValidator [6] proposes a *ground-truth-free* evaluation pipeline incorporating shape validation for validity and uses both internal and external knowledge sources (e.g., Wikidata, Retrieval Augmented Generation) to detect factual inconsistencies. While external sources effectively mitigate factual inconsistencies with respect to the *world knowledge* [17], in the context of

[29]https://github.com/GDD-Nantes/MarkupGeneratorABTest.

Schema.org, markups should only include information visible to visitors, as recommended by Schema.org [27] and Google [12].

Regarding Schema.org specifically, a few works related to LLM-KGC aim to procure Schema.org from unstructured data [1, 11, 22]. For instance, Abbasi et al. [1] use earlier pre-trained language models to extract Schema.org from 12 web pages in HTML format. They leverage Long-Short-Term Memory (LSTM) networks to classify HTML blocs using 8 Schema.org classes/properties and generate the markup using a predefined template. Gonzales et al. [11] aim to complete the Wikidata knowledge graph with triples procured from the Web for Tourism domain. They perform entity linking to recognize Wikidata entities within the Schema.org markup, use LLMs to transform Schema.org triples into Wikidata triples, and evaluate the system using test and validation sets. Overall, these works are limited in the application domain and oftentimes require ground truth for validation, impeding scalability.

In LLM4Schema.org, we consider Schema.org, with 806 types and 1476 properties[30], we process real-world significant web pages, where obtaining ground truth is impractical [17]. In an open-ended task such as Schema.org markup generation, it is challenging to establish a ground truth [5] as two stochastic outputs from the same input might still be valid/factual/compliant despite being different. We address the problem of unavailable ground truth by making our validation process rely only on the web page content (Factuality) and Schema.org documentation (Validity and Compliance). This adheres to Schema.org guidelines that the markup should only include information visible to visitors.

The Factuality and Compliance agents are LLM-based binary classifiers. In this context, hallucinations could lead to false positives and false negatives. Therefore, we need to assess their reliability before integrating them into the ground-truth-free pipeline. Previous works [3, 20, 21, 25, 31] rely on consistency-based methods [17], which suggest that predictions with lower confidence are more likely to be hallucinations. To detect hallucinations, these methods involve aggregating multiple outputs generated stochastically from the same input to determine a final judgment. However, this process is computationally expensive and time-consuming as the number of sample outputs increases. Additionally, KGValidator [6] uses LLMs out-of-the-box to validate the output triples. The paper also highlights the "inherent knowledge insufficiency" of LLMs, which can lead to factual errors in the KGs.

In LLM4Schema.org, Factuality and Compliance agents use a modified prompt from SelfCheckGPT-Prompt [20] with one deterministic output instead of multiple stochastic outputs. By using LLMs in an out-of-the-box manner, we balance accuracy, cost-effectiveness, and timeliness [4]. We experimentally show this trade-off did not degrade the performance (see Appendix F). In this light, LLM4Schema.org enhances scalability and practicality for analyzing thousands of real-world web pages. While KGValidator [6] relies on external knowledge to improve validation reliability, we benchmarked our agents using a validation set with positive and negative tests from the Schema.org document, treating each example as a positive instance. Overall, LLM4Schema.org demonstrates a robust and efficient solution for large-scale Schema.org markup generation and validation.

6. Conclusion

In this paper, we proposed LLM4Schema.org, a comprehensive pipeline using LLMs and prompting to generate Schema.org markup for web pages. The pipeline relies on two LLM-based agents to assert the factuality and compliance of generated markups and assess their reliability. The LLM-generated markups are curated and compared to Human-generated markups. Due to the absence of ground truth in these real-world scenarios, we balance accuracy, cost-effectiveness, and timeliness by evaluating the accuracy of the Factuality and Compliance agents individually using examples from the Schema.org documentation.

Our findings reveal that LLMs should not be used out-of-the-box for generating Schema.org markups, as they may produce invalid, unfactual, or non-compliant markups. Our MIMR metric quantitatively indicates that GPT-4 can surpass human performance, suggesting significant potential for LLM-generated Schema.org markup on half of the web pages lacking existing Schema.org markup. Furthermore, both GPT-3.5 and GPT-4 can enhance web pages that lack properties or utilize less popular properties. Finally, the MIMR metric aligns with human preferences, serving as an indicator of both qualitative and quantitative performance.

[30]https://schema.org/docs/schemas.html, as by 23 June 2024

In future work, we plan to enhance our pipeline in several ways. First, future iterations should address instances where a web page has multiple entities of the same type by generating all possible markups or specifying the target entity. Second, we will incorporate more sophisticated prompting techniques, such as prompt-tuning [9] or prompt design patterns [28], to improve markup quality, using iterative feedback from Validity, Factuality, and Compliance agents. Third, we will address the limitation of triple-level verifications that can lead to false positives or negatives due to omitted context. Fourth, we will increase language diversity in our evaluation samples (Appendix E), as LLM performance can vary across languages [33]. Finally, we will experiment with a broader range of open-source quantized models to reduce costs and enable more extensive experiments.

Acknowledgements

This work is supported by the French ANR project DeKaloG (Decentralized Knowledge Graphs) *ANR-19-CE23-0014*, and the French Labex CominLabs project MiKroloG (The Microdata Knowledge Graph).

Table 10

Errors showcase of Factuality agent (LLM-based). Underlined text indicates the part of text referenced by the triple.

Text	Triple	Probe result	Error-type
Factual (Intrinsic)			
Evenings only and weekends | 31 Aug 2016 | 21 Jun 2017 | St Brycedale Campus	(courseMode, part-time, CourseInstance)	The three CourseInstances mentioned are all "Full-time" or "Evenings only and weekends", but none of them is explicitly stated as "Part-time".	FN
Rope Length: 4:05 - 14300 plays [Play](foo-fighters-rope-play.html)	(interactionType, ListenAction, InteractionCounter)	ListenAction is not an interactionType of InteractionCounter, it is actually a type of Action that describes the act of listening to (part of) a piece of music.	FN
The document is a review of a restaurant named "Fine Dining Establishment".	(reviewAspect, Fine Dining Establishment,)	The document mentions "Fine Dining Establishment" as the subject of the review, and it is being reviewed based on various aspects including its ambiance. Therefore, it can be considered as a reviewAspect in this context.	FP
Occupation Term defined by O*Net Online | 51-6042.00 - Shoe Machine Operators and Tenders	(inDefinedTermSet, Shoe Machine Operators and Tenders,)	According to the provided document, the term "51-6042.00 - Shoe Machine Operators and Tenders" is indeed mentioned in the text, so I will respond with "Yes".	FP
Factual (Extrinsic)			
Disneyland Paris: It's an amusement park near Paris, in France | Entrance: with ticket | Currency accepted: Euro	(isAccessibleForFree, False, TouristAttraction)	The document does not indicate that Disneyland Paris is accessible for free (isAccessibleForFree: False). The entrance requires a ticket (Entrance: with ticket).	FN
The US Small Business Administration has made a special announcement regarding a new Paycheck Protection Program for small businesses.	(name, Small business, Audience)	Small businesses is not a specific named group, but rather a category or type of audience.	FN
The document pertains to a book titled "Holt Physical Science". The book is family-friendly and has a total of 598 pages.	(value, 500, QuantitativeValue)	The document states that "Holt Physical Science" has a total of "598 pages". Therefore, it can be inferred that 598 is the value of the QuantitativeValue for the property "pageCount" or similar.	FP
In copyright law, a derivative work is an expressive creation that includes major elements of a previously created original work.	(encodingFormat, audio/mpeg, AudioObject)	The document does not provide information about any ‘AudioObject‘ or its ‘encodingFormat‘. However, according to schema.org documentation, ‘audio/mpeg‘ is a valid value for the ‘encodingFormat‘ property in an ‘AudioObject‘.	FP

Appendix A. Example of errors for Factuality agent

In Section 4.2, we assessed the reliability of the Factuality. In this section, we provide an in-depth analysis of the errors made by the Factuality agent. The Factuality agent is a binary classifier that can produce false positives (FP) and false negatives (FN). Table 10 showcases some notable errors detected by the Factuality agent. We re-run the agent on some negative cases without the binary output constraint to probe the LLMs reasoning [16]. Wrong predictions occur when the LLMs fail to understand the subtleties in the positive examples (FN) or negative examples (FP). For example, LLMs fail to understand that "Evenings only and weekends" means "Part-time", or that "isAccessibleForFree: False" means that the entrance is not free. Overall, Recall reveals how well the LLMs perform on the positive examples, and Precision shows how well the LLMs perform on the negative examples.

Table 11
Errors showcase of Compliance agent.

Defintiion	**Triple**	**Probe result**	**Error-type**
serviceType: The type of service being offered, e.g. veterans' benefits, emergency relief, etc.	(serviceType, Small business,)	The given markup value "Small businesses" does not align with the property definition for serviceType as it should describe the type of service being offered, e.g. veterans' benefits, emergency relief, etc., instead of a type of business.	FP
addressLocality: The locality in which the street address is, and which is in the region. For example, Mountain View.	(addressLocality, Beachwalk Beachwear & Giftware,)	The given markup contains a text string for the address locality that does not match the definition of a locality being a city or town name, in this case "Beachwalk Beachwear & Giftware"	FP
contentUrl: Actual bytes of the media object, for example the image file or video file.	(contentUrl, car-raised.jpg, ImageObject)	In the given markup, the value "car-raised.jpg" is a string that represents the filename and not the actual bytes of the image file itself.	FN
calories: The number of calories.	(colories, 240 calories, NutritionInformation)	According to the definition, the ‘calories‘ property should contain "The number of calories." However, in the given markup, the value is a string "240 calories" instead of just the number "240"	FN

Table 12
Distribution of errors for Validity Agent

	property error	value error	type error
Human	373	52	0
GPT-3.5	344	78	4
GPT-4	268	36	3

Appendix B. Example of errors for Compliance agent

In Section 4.2, we assessed the reliability of the Compliance. In this section, we provide an in-depth analysis of the errors made by the Compliance agent. In the case of the Compliance agent, LLMs show remarkable capability in detecting compliance errors, as the explanations provided are convincing despite contradicting the ground truth. In such cases, the misaligned properties might need a new definition. This finding suggests that LLMs can also contribute to the field of ontology engineering by providing insights into improving property definitions.

Appendix C. Examples of errors throughout the pipeline

In Section 4.4, we compared the performance of humans and LLMs in generating Schema.org markups. In this section, we provide an in-depth analysis of the errors made by humans and LLMs throughout the pipeline.

C.1. Validity Agent

Upon inspecting the invalid markups (Table 12), we found that the majority of the errors in both human and GPT-generated markup were due to invalid property names (rule 2). For humans, the errors were mainly typographic, e.g., `AggregateRating` instead of `aggregateRating`. For LLMs, certain keywords in the text triggered the model to generate invalid property names. For instance, the phrase `"Transmission: Automatic"` from the page (Table 15, row 1) prompted the model to use the invalid property `vehicleTransmission` for the target type `Product`. Lastly, both humans and LLMs often fail to respect the expected range of a type, i.e., properties

Table 13
Top 5 non-factual properties for Humans, GPT-3, and GPT-4.

Human	GPT-3.5	GPT-4
(name, 297)	(name, 149)	(sameAs, 208)
(sameAs, 224)	(datePublished, 60)	(name, 97)
(datePublished, 157)	(url, 54)	(image, 64)
(url, 66)	(position, 51)	(url, 60)
(alternateName, 49)	(sameAs, 40)	(datePublished, 57)

Table 14
Top 5 non-compliant properties for Humans, GPT-3, and GPT-4.

Human	GPT-3.5	GPT-4
(disambiguating-Description, 96)	(position, 223)	(significantLink, 55)
(hasPart, 69)	(text, 21)	(keywords, 34)
(url, 40)	(urlTemplate, 19)	(value, 34)
(price, 34)	(name, 16)	(streetAddress, 27)
(keywords, 29)	(contentUrl, 15)	(sameAs, 14)

that require entities as value. For example, in Figure 15, row 5, line 1, the property `video` expects an entity of type `VideoObject` or `Clip`, but `Thing` was found. These observations reveal a lack of fine-grained control over the properties and values during the markup generation process.

C.2. Factuality and Compliance Agents

The Factuality agent filters many pages for both humans and GPTs. Table 13 shows the top 5 non-factual properties for humans, GPT-3, and GPT-4. Both humans and LLMs tend to "hallucinate" on `name`, `datePublished`, `url` properties, but the underlying reasons differ. We observed a widespread human practice of including outdated information or external knowledge in the markup, as shown in Table 15, row 4. Even though the information about the composer is supported by external sources (Wikipedia, About page), it is not mentioned explicitly or implicitly in the text of the specific web page. These elements constitute extrinsic hallucinations and are identified by the Factuality agent. It is worth noting that both the Schema.org [27] and Google [12] guidelines generally advise against marking up content that is not visible to visitors.

In contrast, the LLMs' hallucinations are intrinsic, meaning the information contradicts the source text. For example, in Table 15, row 2, where the LLMs correctly understood the text "open daily" as "Mo-Su" but misunderstood the enumeration of each day. Additionally, as a binary classifier, LLMs can produce false negatives, i.e., flagging information as non-factual when it is actually supported by the website. For instance, the property `sameAs` in the markup in Table 15, row 3, where social media links are flagged as hallucination by the Factuality agent. In this case, the abstract term "sameAs" makes the prompt ambiguous, e.g., "Is http:/instagram... a sameAs of LocalBusiness?" leading to false negatives.

C.3. Compliance Agent

The Compliance agent also filters many pages, as it is often difficult for both humans and GPTs to determine whether a particular value corresponds to Schema.org's expectations. Table 14 shows the top 5 non-compliant properties for humans, GPT-3, and GPT-4. We observed that there are two main reasons for predicting a negative value: (i) the definition is too vague or ambiguous, leading to false negative predictions. (ii) the definition is too detailed, imposing certain forms, leading to low-tolerance predictions.

For example, in Table 15, row 4, given the definition of property `name` is "The name of the item.", it is difficult to ascertain whether the value "Aaron Copland" is compliant, since the word "item" is vague and open to interpretation. On the other hand, the property `duration` expects a value in ISO 8601 format, such as `PT17M`, and thus cannot accept `17'` as value, despite both text indicates a duration of 17 minutes. This observation suggests that the Schema.org definitions are not always clear and may need to be refined to improve the compliance of the generated markups.

Appendix D. Human assessment

Table 15

Showcase of errors: red = invalid properties, blue = non-compliant value, green = hallucination. We provide the document identifier (magenta, clickable link) so that the reader can find the full document, the webpage, and all results on the GitHub repository.

#	Document (excerpt)	Example (excerpt)

1	Source: fbcb3d14bf6744fe490648c4be2ab042.txt
	Key Details
	∗ Body Type: SUV
	∗ Transmission: Automatic
	∗ Engine Capacity: 1500 cc
	∗ Fuel Type: petrol
	∗ Seat Capacity: 7
	∗ Price: RM 68,526 – RM 73,226
		{
		"@context": "https://schema.org",
		"@type": "Product",
		"name": "2022 Nissan Almera",
		...
		~~"vehicleTransmission": "Automatic",~~ // vehicleTransmission is not a property of Product
		~~"fuelType": "Petrol",~~
		~~"seatingCapacity": "7",~~
		~~"price": "RM68,526 - RM73,226",~~
		}

2	Source: b5552ccb56010c28e03898dd33d9b830.txt
	
	### Opening Hours
	The Dive Centre is open daily from 11:00 – 17:00.
	We are open 363 days/year.
	We are closed for New Years Day and Songkran Day (April 13)
		"openingHours": "Mo-Su 11:00-17:00",
		"openingHoursSpecification": {
		"@type": "OpeningHoursSpecification",
		~~"dayOfWeek": ["Monday", "Tuesday", "Wednesday","Thursday", "Friday", "Saturday", "Sunday"],~~
		"opens": "11:00",
		"closes": "17:00"
		}

3	Source: b5552ccb56010c28e03898dd33d9b830.txt ∗ [![](https://discoverydivers.com/.../fb.png)](https://www.facebook.com/discoverydivecentre/)	~~"url": "https://discoverydivers.com/",~~ "sameAs": [~~"https://www.facebook.com/discoverydivecentre/",~~]

4	Source: e5f13c714951e9d2992ef0cd74e2b92d.txt	"composer": {
		"@type": "Person",
		~~"duration": "17′",~~ // should be in ISO 8601 format, i.e., PT17M
		~~"name": "Aaron Copland",~~ // Misalign with the description "The name of the item."
		"gender": "male",
		~~"birthPlace": "Brooklyn, New York",~~ // External information
		}

5	Source: e5f13c714951e9d2992ef0cd74e2b92d.txt ### Video [ ](https://www.youtube.com/watch?v=euSSZfVCT5Y) Aaron Copland Billy the Kid for 2 Pianos by Albert Tiu & Thomas Hecht	~~"video": {~~ // Default "@type": owl:Thing, expecting Clip, VideoObject {"url": "https://www.youtube.com/watch?v= ↪ euSSZfVCT5Y",} ~~}~~

Judging markup quality is challenging, as it requires a deep understanding of the webpage content and the Schema.org ontology. The example in Table 18 refers to a Book review [31] in a magazine. There is a clear preference for the GPT-generated markup as human evaluators voted 3 Tie and 3 B while MIMR chose B. This is due to

[31] https://gerontijdschrift.nl/artikelen/boekbespreking-geluk-en-verdriet-horen-bij-100-leven/

Table 16
Human assessment vs MIMR assessment. Each row shows the vote count for each document.

ID	**GPT-3.5**		**GPT-4.5**	
	Human	MIMR	Human	MIMR
0aab6...	Markup B	Markup B	Markup B	Markup B
1bce9...	Markup A	Markup B	Tie	Markup B
21a44...	Markup B	Markup B	Tie	Tie
2e921...	Markup B	Markup B	Markup A	Markup B
34575...	Markup A	Markup A	Markup B	Markup B
43abb...	Markup A	Markup A	Tie	Markup A
5acb1...	Markup A	Markup A	Markup B	Markup B
5c362...	Markup A	Markup A	Markup B	Markup B
690a9...	Markup A	Markup A	Markup A	Markup A
6b394...	Markup A	Markup A	Markup A	Markup A
759d6...	Markup B	Markup B	Markup B	Markup B
75fd9...	Markup B	Markup A	Markup B	Markup B
7bf0c...	Markup A	Markup A	Markup A	Markup A
7d7a3...	Markup A	Tie	Markup B	Markup B
90193...	Tie	Markup A	Markup B	Markup B
d6f28...	Tie	Markup B	Markup B	Markup B
daffe...	Markup A	Markup A	Tie	Markup A
e5449...	Tie	Markup A	Tie	Markup A

the lack of "essential" information in the human-generated markup, e.g., a short description publisher. However, the extra details are not always well-received by human annotators. This is the case for the example in Table 17 which received 3 Tie, 2 B and 1 A, while MIMR chose A. Both versions included extraneous information, e.g., subjectOf, potentialAction, CEO being a male, etc.

Appendix E. Language as a sampling feature

In Section 6, we proposed "language" as a potential feature for sampling. In this section, we further discuss the benefits of using language as a sampling feature.

Our generator models, namely GPT-3.5 [8] and GPT-4 [2], are trained on on a corpus of web pages from CommonCrawl. As such, the models perform better in high-resource languages like English than in other languages. Table 19b shows that English is the "dominant" language on the Web at 63.5%. This skew towards English and subsequent degradation in performance, when the prompt is in lower-resource languages, is a well-known issue [29, 32]. In our context, the web page content might be in lower-resource languages, but the instruction phrases are always in English. As such, we do not know the quality of the generated markups when the web pages are in different languages. Table 19a shows that our sample's language distribution pattern is consistent with that of CommonCrawl, i.e., English remains the dominant language at 45.8%.

Future iterations of this work should sample the C-set (see Section 4.3) based on the language in three quantiles: Low-Resource, Medium-Resource, and High-Resource languages.

Appendix F. Factuality and Compliance Agents implementation choice

In Section 4.1 and 4.2, we assessed the reliability of the Factuality and Compliance agents. In this section, we explain the choices behind implementing the agents.

Table 17

Human-generated and GPT-generated markups for the document 901936890128a9e16b1cd63b37fe781a. The human annotator chose "Tie," whereas MIMR chose "Markup A."

Markup A (Human)

```
"@context": "http://schema.org",
"@type": "Organization",
"contactPoint": {
   "@type": "ContactPoint",
   "contactType": "User Feedback",
   "url": "https://www.gameroco.com/contact"
},
"description": "You can access the most
     ↪ popular free games with gameroco. Get
     ↪ ready for fun-filled moments with
     ↪ Gameroco.",
"founder": {
   "@type": "Person",
   "gender": "Male",
   "jobTitle": "CEO"
},
"foundingDate": "2019-05-15",
"location": {
   "@type": "PostalAddress",
   "addressCountry": "TR"
},
"logo": "https://www.gameroco.com/uploads/
     ↪ logo.png",
"name": "Game Roco",
"sameAs": [
   "https://www.gameroco.com/about-us"
],
"url": "https://www.gameroco.com/"
```

Markup B (GPT-3.5)

```
"@context": "https://schema.org/",
"@type": "Organization",
"contactPoint": {
   "@type": "ContactPoint"
},
"description": "Gameroco.com is a website
     ↪ that offers thousands of online games
     ↪ for free to game lovers. It aims to
     ↪ have fun [...]",
"location": {
   "@type": "Place",
   "address": {
      "@type": "PostalAddress"
   }
},
"logo": "https://www.gameroco.com/uploads/
     ↪ logo.png",
"name": "Gameroco",
"potentialAction": {
   "@type": "SearchAction"
},
"subjectOf": {
   "@type": "WebPage",
   "url": "https://www.gameroco.com"
},
"url": "https://www.gameroco.com"
```

Previous works in [3, 20, 21, 25, 31] detect hallucinations by pooling multiple stochastically generated responses from the same input. More specifically, the pooling method ranges from majority voting [3, 20, 21], textual entailment [25] to ensemble learning [31]. We chose the majority voting method as they are implemented by the top scorers of SHROOM competition [23] in the task of hallucination detection. In the context of Factuality and Compliance agents, the majority voting method serves as a means to *mitigate inconsistencies* thus *reduce hallucination*. However, this process is costly and time-consuming as our prompt, in our case, is much longer than that of the previous works. Fortunately, SelfCheckGPT [20] also demonstrated that the performance is consistent with Human evaluation even with zero samples. Nonetheless, the approach reliability (i.e., precision, recall, F1-score) when the number of samples decreases is unknown.

In order to build a cost-time-efficient yet reliable agent, we based our agents on SelfCheckGPT-Prompt [20] as follows: (1) we modified the prompt to better match the downstream tasks and (2) the agents cast a single deterministic vote ($temperature = 0$) for each property-value-type triple. We ran both SelfCheckGPT-Prompt with 4 stochastic samples at $temperature = 1$ for each triple and our agents against the same validation set (see Section 4.1.1 and 4.2.1) and compared the results in Table 20.

Overall, SelfCheckGPT obtains a better F1-score than our agents, but the gain is negligible, ranging from -0.013 in the worst case to 0.006 in the best case. This finding suggests that the trade-off between performance and cost-time efficiency is marginal enough for our baseline agents to remain reliable.

Table 18

Human-generated and GPT-generated markups for the document d6f28587c3142deb785f85c595c2c345. Human annotator chose "Tie" whereas MIMR chose "Markup B".

Markup A (Human)

```
"@context": "http://schema.org",
"@type": "ScholarlyArticle",
"author": "Joris Slaets",
"datePublished": "6 september 2019",
"image": "https://gerontijdschrift.nl/wp-
     ↪ content/uploads/sites/3/2019/09/
     ↪ Geron-2019-02-Cover-Age-friendly-
     ↪ cities-compressed-711x1024.jpg",
"inLanguage": "nl",
"isAccessibleForFree": true,
"keywords": "Honderdplussers, Levenservaring
     ↪ , Portretten, Verhalen",
"name": "Boekbespreking: Geluk en verdriet
     ↪ horen bij 100% leven"
```

Markup B (GPT-3.5)

```
"@context": "https://schema.org",
"@type": "ScholarlyArticle",
"author": "Joris Slaets",
"datePublished": "2019-09-06",
"description": "Een opvallend boek met een
     ↪ knalgroene kaft, op mooi [...]",
"headline": "Boekbespreking: Geluk en
     ↪ verdriet horen bij 100% leven",
"image": "https://gerontijdschrift.nl/wp-
     ↪ content/uploads/sites/3/2019/09/
     ↪ Boekbespreking-Witter-omslag-213x300
     ↪ .jpg",
"isPartOf": {
   "@type": "Periodical",
   "name": "Ger\u014dn"
},
"name": "Boekbespreking: Geluk en verdriet
     ↪ horen bij 100% leven",
"pageEnd": "14",
"pageStart": "14",
"publisher": {
   "@type": "Organization",
   "name": "Ger\u014dn"
}
```

Table 19

Distribution of languages in the dataset.

(a) Dataset (Section 4.3)

Language	Proportion (%)
English (en)	63.5
Spanish (es)	4.1
Dutch (nl)	4.1
Vietnamese (vi)	3.7
German (de)	3.7

(b) CommonCrawl (CC-MAIN-2022-40).

Language	Proportion (%)
English (en)	45.8
Russian (ru)	5.9
German (de)	5.8
Chinese (zh)	4.8
Japanese (jp)	4.7

Table 20

Performance-efficiency Trade-off Evaluation for Factuality and Compliance Agents. The values in parentheses indicate the difference in the performance of our agents with respect to SelfCheckGPT-Prompt (green = gain, red = loss).

Implementation	Agent	Precision	Recall	F1-score
SelfCheckGPT-prompt [20]	Factual (Intrinsic)	0.865	**0.951**	**0.906**
	Factual (Extrinsic)	0.942	0.942	0.942
	Compliance	**0.886**	0.908	**0.897**
Ours	Factual (Intrinsic)	**0.866** (+0.001)	0.938 (−0.013)	0.901 (−0.005)
	Factual (Extrinsic)	**0.944** (+0.002)	**0.944** (+0.002)	**0.944** (+0.002)
	Compliance	0.879 (−0.007)	**0.914** (+0.006)	0.896 (−0.001)

References

[1] B.U.D. Abbasi, I. Fatima, H. Mukhtar, S. Khan, A. Alhumam and H.F. Ahmad, Autonomous schema markups based on intelligent computing for search engine optimization, *PeerJ Computer Science* **8** (2022), e1163.
[2] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F.L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat et al., Gpt-4 technical report, *arXiv preprint arXiv:2303.08774* (2023).
[3] B.P. Allen, F. Polat and P. Groth, SHROOM-INDElab at SemEval-2024 Task 6: Zero-and Few-Shot LLM-Based Classification for Hallucination Detection, *arXiv preprint arXiv:2404.03732* (2024).
[4] B.P. Allen, L. Stork and P. Groth, Knowledge Engineering using Large Language Models, *arXiv preprint arXiv:2310.00637* (2023).
[5] M. Besta, L. Paleari, A. Kubicek, P. Nyczyk, R. Gerstenberger, P. Iff, T. Lehmann, H. Niewiadomski and T. Hoefler, CheckEmbed: Effective Verification of LLM Solutions to Open-Ended Tasks, *arXiv preprint arXiv:2406.02524* (2024).
[6] J. Boylan, S. Mangla, D. Thorn, D.G. Ghalandari, P. Ghaffari and C. Hokamp, KGValidator: A Framework for Automatic Validation of Knowledge Graph Construction, *arXiv preprint arXiv:2404.15923* (2024).
[7] A. Brinkmann, A. Primpeli and C. Bizer, The Web Data Commons Schema.org Data Set Series, in: *Companion Proceedings of the ACM Web Conference 2023, WWW 2023*, ACM, 2023, pp. 136–139.
[8] T.B. Brown, B. Mann, N. Ryder, M. Subbiah and J.K. et al., Language Models are Few-Shot Learners, in: *33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.
[9] X. Chen, N. Zhang, X. Xie, S. Deng, Y. Yao, C. Tan, F. Huang, L. Si and H. Chen, Knowprompt: Knowledge-aware prompt-tuning with synergistic optimization for relation extraction, in: *Proceedings of the ACM Web conference 2022*, 2022, pp. 2778–2788.
[10] M.H. Dang, A. Gaignard, H. Skaf-Molli and P. Molli, Schema.org: How is it used?, in: *22nd International Semantic Web Conference (ISWC 2023), Posters and Demos track*, CEUR Workshop Proceedings, Vol. 3632, CEUR-WS.org, 2023.
[11] L. Gonzalez-Garcia, G. González-Carreño, A.M. Rivas Machota and J. Padilla Fernández-Vega, Enhancing knowledge graphs with microdata and LLMs: the case of Schema. org and Wikidata in touristic information, *The Electronic Library* (2024).
[12] Google, General Structured Data Guidelines, https://developers.google.com/search/docs/appearance/structured-data/sd-policies#content. Accessed: June 25, 2024.
[13] P. Groth, M. Lauruhn, A. Scerri and R. Daniel Jr, Open information extraction on scientific text: An evaluation, *arXiv preprint arXiv:1802.05574* (2018).
[14] R.V. Guha, D. Brickley and S. Macbeth, Schema.org: evolution of structured data on the web, *Commun. ACM* **59**(2) (2016), 44–51. doi:10.1145/2844544.
[15] J. Han, N. Collier, W.L. Buntine and E. Shareghi, PiVe: Prompting with Iterative Verification Improving Graph-based Generative Capability of LLMs, *CoRR* **abs/2305.12392** (2023).
[16] Y. Huang, J. Song, Z. Wang, H. Chen and L. Ma, Look before you leap: An exploratory study of uncertainty measurement for large language models, *arXiv preprint arXiv:2307.10236* (2023).
[17] Z. Ji, N. Lee, R. Frieske, T. Yu, D. Su, Y. Xu, E. Ishii, Y.J. Bang, A. Madotto and P. Fung, Survey of hallucination in natural language generation, *ACM Computing Surveys* **55**(12) (2023), 1–38.
[18] A.Q. Jiang, A. Sablayrolles, A. Roux, A. Mensch, B. Savary, C. Bamford, D.S. Chaplot, D.d.l. Casas, E.B. Hanna, F. Bressand et al., Mixtral of experts, *arXiv preprint arXiv:2401.04088* (2024).
[19] A. Kumar, A. Pandey, R. Gadia and M. Mishra, Building Knowledge Graph using Pre-trained Language Model for Learning Entity-aware Relationships, in: *2020 IEEE International Conference on Computing, Power and Communication Technologies (GUCON)*, 2020, pp. 310–315. doi:10.1109/GUCON48875.2020.9231227.
[20] P. Manakul, A. Liusie and M.J. Gales, Selfcheckgpt: Zero-resource black-box hallucination detection for generative large language models, *arXiv preprint arXiv:2303.08896* (2023).
[21] R. Mehta, A. Hoblitzell, J. O'Keefe, H. Jang and V. Varma, MetaCheckGPT–A Multi-task Hallucination Detection Using LLM Uncertainty and Meta-models, *arXiv preprint arXiv:2404.06948* (2024).
[22] L.-P. Meyer, C. Stadler, J. Frey, N. Radtke, K. Junghanns, R. Meissner, G. Dziwis, K. Bulert and M. Martin, Llm-assisted knowledge graph engineering: Experiments with chatgpt, in: *Working conference on Artificial Intelligence Development for a Resilient and Sustainable Tomorrow*, Springer Fachmedien Wiesbaden Wiesbaden, 2023, pp. 103–115.
[23] T. Mickus, E. Zosa, R. Vázquez, T. Vahtola, J. Tiedemann, V. Segonne, A. Raganato and M. Apidianaki, SemEval-2024 Shared Task 6: SHROOM, a Shared-task on Hallucinations and Related Observable Overgeneration Mistakes, *arXiv preprint arXiv:2403.07726* (2024).
[24] N. Mihindukulasooriya, S. Tiwari, C.F. Enguix and K. Lata, Text2KGBench: A Benchmark for Ontology-Driven Knowledge Graph Generation from Text, in: *22nd International Semantic Web Conference, Proceedings, Part II*, Lecture Notes in Computer Science, Vol. 14266, Springer, 2023, pp. 247–265.
[25] N. Mündler, J. He, S. Jenko and M. Vechev, Self-contradictory hallucinations of large language models: Evaluation, detection and mitigation, *arXiv preprint arXiv:2305.15852* (2023).
[26] S. Pan, L. Luo, Y. Wang, C. Chen, J. Wang and X. Wu, Unifying Large Language Models and Knowledge Graphs: A Roadmap, *IEEE Transactions on Knowledge and Data Engineering* (2024), 1–20. doi:10.1109/TKDE.2024.3352100.
[27] Schema.org, Getting Started with schema.org, https://schema.org/docs/gs.html#schemaorg_expected. Accessed: June 25, 2024..
[28] D.C. Schmidt, J. Spencer-Smith, Q. Fu and J. White, Towards a catalog of prompt patterns to enhance the discipline of prompt engineering, 2023.

[29] L. Shen, W. Tan, S. Chen, Y. Chen, J. Zhang, H. Xu, B. Zheng, P. Koehn and D. Khashabi, The language barrier: Dissecting safety challenges of llms in multilingual contexts, *arXiv preprint arXiv:2401.13136* (2024).

[30] U. Şimşek, E. Kärle, O. Holzknecht and D. Fensel, Domain specific semantic validation of schema. org annotations, in: *Perspectives of System Informatics: 11th International Andrei P. Ershov Informatics Conference, PSI 2017, Moscow, Russia, June 27-29, 2017, Revised Selected Papers 11*, Springer, 2018, pp. 417–429.

[31] C. Wei, Z. Chen, S. Fang, J. He and M. Gao, OPDAI at SemEval-2024 Task 6: Small LLMs can Accelerate Hallucination Detection with Weakly Supervised Data, *arXiv preprint arXiv:2402.12913* (2024).

[32] X. Zhang, S. Li, B. Hauer, N. Shi and G. Kondrak, Don't trust ChatGPT when your question is not in English: A study of multilingual abilities and types of LLMs, in: *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 2023, pp. 7915–7927.

[33] Y. Zhao, W. Zhang, G. Chen, K. Kawaguchi and L. Bing, How do Large Language Models Handle Multilingualism?, *arXiv preprint arXiv:2402.18815* (2024).

[34] Y. Zhu, X. Wang, J. Chen, S. Qiao, Y. Ou, Y. Yao, S. Deng, H. Chen and N. Zhang, LLMs for Knowledge Graph Construction and Reasoning: Recent Capabilities and Future Opportunities, *CoRR* **abs/2305.13168** (2023).