# Filling in the Gaps: LLM-Based Structured Data Generation from Semi-Structured Scientific Data

**Anonymous Authors**[1]

## Abstract

The restructuring of scientific data is crucial for scientific applications such as literature review automation, hypothesis generation, and experimental data integration. As pre-trained Large Language Models (LLMs) have been improved and become more accessible, they have the potential to enhance these processes by efficiently interpreting and combining large amounts of data. In this work, we present a comprehensive overview of this restructuring framework, exploring approaches designed to gather, augment, and restructure unstructured scientific data. We explore how LLMs can better integrate diverse unstructured sources of data into a comprehensive document or improve existing knowledge graphs. As a way of exploring these approaches, we study the specific case of synthesizing documents for chemical reactions, where organized data is lacking. We observed that improvements in web page parsers significantly reduce the cost of using LLMs. By developing a parser for chemical reactions, we construct a framework for document augmentation and knowledge graph completion, and significantly reduce the usage costs (in terms of input/output tokens) of LLMs.

## 1. Introduction

In scientific research, the task of gathering information from various sources and reconstructing it into structured data is crucial for many processes, including literature review automation (Qureshi et al., 2023; Wang et al., 2024; Lu et al., 2023), hypothesis generation (Zhou et al., 2024), and experimental data integration (Bran et al., 2023). The recent advent of pre-trained Large Language Models (LLMs) has enabled the use of their advanced text processing capabilities to reorganize unstructured data into desired formats for these tasks. Most scientific information resides on the internet, often scattered across different webpages without proper organization. This fragmented nature of data poses a significant challenge for researchers who need to compile comprehensive datasets from these unstructured sources.

LLM-based development frameworks, such as LangChain (Chase, 2022), provide methods to collect this scattered information by searching for and parsing site data, and then utilizing LLMs to reconstruct it into structured formats like text documents or knowledge graphs. These frameworks leverage the ability of LLMs to understand context and semantics, making them ideal for synthesizing coherent representations of data from disparate sources. However, several choices and limitations exist within this process. For instance, there are various parsers available for site data extraction, each with its own strengths and weaknesses. The data reconstruction process typically relies on text-based data, often neglecting tabular or image data. Additionally, the use of LLMs incurs additional costs associated with inference requiring GPU or API usage, which can be prohibitive for large-scale data processing tasks.

In this study, we explore how to effectively gather diverse information through web search and reconstruct it into structured data in scientific contexts. Our focus is specifically on the domain of chemical reactions, a field where well-organized and comprehensive data is often not readily available. Based on these observations, we found that an HTML parser for parsing web pages significantly reduces the cost required when using LLMs. Therefore, we developed a specialized parser for chemical reaction websites called ReactionParser. Using this parser, we evaluate the cost of using LLMs in document augmentation and knowledge graph completion through a data reconstruction pipeline.

## 2. Background

### 2.1. Large Language Models (LLMs)

Large language models (LLMs) have been predominantly trained on massive datasets collected from the internet and are mostly utilized for natural language processing tasks,

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

such as question answering or summarization. Recently, decoder-only models (Radford et al., 2018; 2019; Brown et al., 2020) have demonstrated superior performance in generation tasks compared to traditional encoder-decoder models (Vaswani et al., 2017). LLMs operate using a fundamental unit known as a 'token', which typically comprises subword-level combinations of characters, depending on the specific tokenizer used. For instance, tokenizing the sentence "Hello, world!" would result in the tokens `["Hello", ",", "world", "!"]`. Due to the attention mechanism present in language models, which essentially encodes all tokens by their relationship with other tokens in the input sequence, this architecture scales quadratically in computation with respect to the number of input tokens. Therefore, the number of tokens is crucial when working with LLMs, as it significantly dictates the computational inference cost. Many proprietary LLMs services, such as Claude-Opus or GPT, charge per input/output token. For example, Claude-Opus charges $75 per 1 million output tokens (Anthropic), while GPT-4 charges $60 per 1 million tokens. Decoder-only LLMs learn and predict the probability of the next token with a causal mask, thus emulating plausible text completions. In other words, it can be considered a model $p(t|P)$ that predicts the probability of the next token $t$ appearing when prompt $P$ is given. However, to perform arbitrary tasks defined by the user, such as in a chatbot system, an instruction-tuning phase is needed. When trained with large amounts of instruction-output formatted text, these LLMs become probabilistic models that predict the likelihood of the next token given an instruction. Such LLMs are referred to as instruction-tuned LLMs.

### 2.2. Knowledge Graphs

Knowledge graphs (KGs) represent sets of entities and their relationships through nodes and edges. Due to the complex interactions and semantic relations contained in subgraphs, KGs can be used for higher-level reasoning tasks, integrating dense information at both the node and edge levels. Knowledge graphs play a crucial role in various AI applications, including search engines, recommendation systems, and question answering. In the context of chatbot systems, they can help identify connections within data and better contextualize user queries, providing more accurate and meaningful responses. Recently, there has been much interest in integrating knowledge graphs with LLMs, leading to various real-world use cases. For example, Google uses language models, like BERT (Devlin et al., 2018), to incorporate knowledge graphs into its search algorithms, thereby better understanding the intent of user queries and providing more relevant search results. Additionally, in the medical field, knowledge graphs and LLMs are combined to extract meaningful patterns from vast amounts of data, including medical records, research documents, and clinical trial results. This integration helps decision support systems in disease diagnosis and treatment development. Thus, the value of KGs is greatly enhanced when used in conjunction with LLMs. In this work, we explore the task of improving existing KGs from web data, referred to as knowledge graph completion.

## 3. LLM-based Data Restructuring

In this section, we share our observations and considerations for each step of data reconstruction using LLMs.

### 3.1. Webpage Domain Restriction

The first consideration is selecting the websites from which to fetch information. It is crucial to differentiate between sites that intentionally block access and those that do not. A webpage's address consists of a domain and a path, concatenated together, with the `{domain}/robots.txt` file implicitly specifying which paths are allowed or disallowed for access. Search services like Google and Bing use this information to display results. Another important factor is the reliability of the information on these sites. The internet hosts a variety of sites with different purposes. It is essential to distinguish between sites containing professional information and those with a community-oriented nature.

### 3.2. HTML Parser

After obtaining website addresses through search, the next step is to fetch information from these websites. Websites are composed of HTML code, and an HTML parser is necessary to convert this into text. There are various open-source parsers available, such as `Pandoc`, `html2markdown`, `html2text`, `Markdownify`, and `Beautiful Soup 4`, which can be chosen based on the purpose. However, a webpage contains not only the main information but also text from menus, tables of contents, buttons, etc. These parsers generally cannot distinguish between these different parts. To address this, using an LLMs to organize the content from HTML code is possible; however, HTML tokenization consumes a significant number of tokens. For instance, tokenizing the HTML code of Wikipedia's Oxygen page with tiktoken requires nearly 200,000 tokens, which is beyond the capacity of most LLMs to handle at once.

### 3.3. Image and Tabular Data

Webpages contain not only text but also images and tables, which also need to be processed. Images might or might not include descriptive text, and their URLs can be very lengthy, necessitating additional handling. Tables come in various formats, and even tables with the same format can have different meanings for the data in the same cells. This should also be considered. The parser used here must effectively

extract the main content of the web page to maintain the essential information while reducing the number of tokens used in subsequent LLMs operations, thereby minimizing the cost of using the LLMs.

### 3.4. Summarization

Although we collected text stripped of HTML tags using an HTML parser, the length of the content varied across different websites. Aggregating the content from multiple sites at once resulted in duplication and inclusion of irrelevant information. Therefore, it is challenging to use this directly for data reprocessing with LLMs. There is a need for an additional step to extract only the relevant information. LLMs can also be utilized in this process, and the type of LLMs used does not significantly affect the content variation.

### 3.5. Utilization of Prior Data

It is possible to reconstitute the data into the desired format using only information obtained through searches. However, if existing related data is already available, leveraging it is also an option. We found that when creating documents, the format (such as section titles, markdown format, etc.) of the generated document varied significantly when using only information obtained from searches. On the other hand, when integrating existing documents with information found through searches, the LLM tended to adhere to the format of the existing documents while incorporating additional information.

### 3.6. Data Integration

Finally, it is necessary to integrate all the data into the desired format. LLMs are used in this process, and it is crucial to ensure that the LLM generates data in the required format. For instance, when recomposing into a document, the content needs to be well-organized by sections. If the data is stored in the form of a knowledge graph, the relationships between entities must be correctly expressed, or the query commands for storing in a graph database must be accurately generated. Our experience indicates that models with well-tuned instruction sets adhere better to these formats. Additionally, when index numbers are assigned to information obtained through searches and instructions are given to mark these index numbers in the integrated content, high-performance LLMs generally adhered to these instructions well.

## 4. Framework Design

Based on the consideration described in Section 3, we design a new data restructuring framework. Section 4.1 describes the methodology for identifying unique chemical reaction types, which helps in gathering essential keywords where the information is generally scarce. In Section 4.2 we examine web domains containing information specific to chemical reactions or organic chemistry/synthesis. Subsequently, Section 4.3 details the process of developing specialized parsers for each domain page. Lastly, Sections 4.4 and 4.5 explain the use of parser outputs for document supplementation and knowledge graph completion, respectively.

### 4.1. Named Chemical Reactions

We obtained 2,146 unique reaction types from chemical reaction records extracted from the Pistachio database (NextMove), where we randomly sampled for our experiments. Due to the specificity of these reaction classes, most of them do not have well-structured entries in sites like Wikipedia, and performing downstream tasks based on LLMs, such as QA or mechanism elucidation, can become a challenge without access to factual/structured sources.

For each of the named reactions, we would like to generate structured documents from trustworthy sources, with information such as reaction mechanism, conditions, catalysts, solvents, and examples. As mentioned before, for well-studied reactions, our focus is on augmenting existing documents, and for less-common named reactions, we would like to synthesize new documents with information freely available.

### 4.2. Gathering Accessible and Frequent Domains

To gather dispersed information from web pages, we utilize the Google Search API since most search engines only index web pages that are allowed in the {domain}/robots.txt of each domain. From the web pages that appeared in these search results, we identified those that returned errors when accessed via a GET request. We specifically gathered the domains of web pages that resulted in errors to exclude them from future searches. The most common error we encountered was the 403 forbidden error (authorization error), typically indicating that access was denied by the server, possibly due to reasons such as copyright restrictions. Despite the existence of methods to bypass these errors (e.g. using Selenium (Stewart et al., 2024)), we chose not to employ such methods in order to adhere to our ethical standards and not violate any policies. After filtering out the domains that produced errors, we analyze how other domains were distributed. The results of this distribution analysis are plotted in Figure 3 in Appendix. The domain names are represented on the x-axis and y-axis shows the number of occurrences for each domain. We observed that responses to queries regarding chemical reactions are predominantly concentrated in specific domains. Given this result, we decided to develop specialized parsers for these particular domains. This strategy enabled us to tailor our searches to these domains more effectively. We adjust the settings

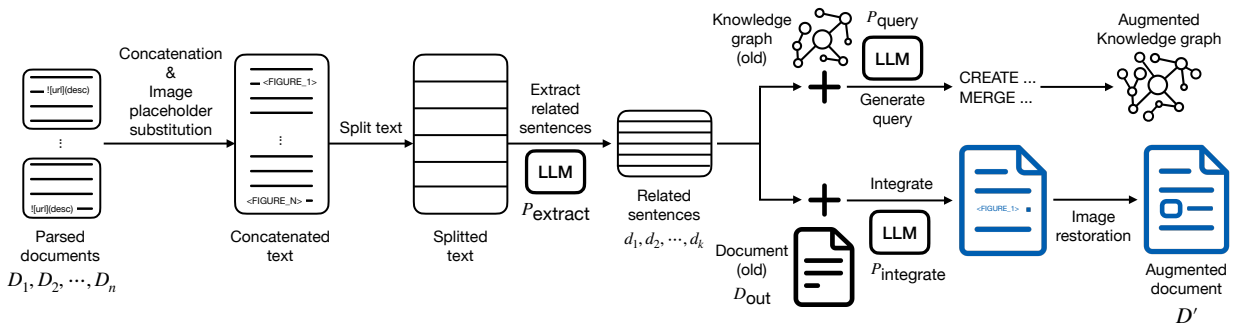


*Figure 1.* Detailed process of document augmentation and knowledge graph completion. Parsed documents are chunked and related sentences are extracted for both tasks. Cypher queries are generated to make new entities and relationships for KG. And outdated(old) document is integrated with the related sentences.

of the Google Search API to preferentially retrieve results from these targeted domains, thereby optimizing our data collection process. This approach not only enhanced the efficiency of our searches but also ensured the relevance and accuracy of the information gathered.

### 4.3. Parser Implementation

We have implemented separate parsers for each of the top 17 chemical reaction-related domains identified in Section 4.2, tailored to the specific needs of each domain. In this section, we introduce the implementation process, image processing techniques, and our method for parsing infoboxes that summarize the information on a Wikipedia page.

#### 4.3.1. Implementation process

The implementation process for a parser in each domain is as follows. First, access one of the search result pages using an internet browser(e.g., Google Chrome). Next, use the Google Chrome development tool to identify the lowest tag block that includes the main content (e.g., title, abstract, contents), excluding elements like sidebars or scripts. Then, store the attributes found in this tag. For instance, if the tag found is `<div id="main_id" class="contents">`, the tag is `div`, and the attributes are `id` and `class` along with their values. Next, obtain the web page's HTML code via a GET request and use the Beautiful Soup 4 Python library to exclude all tags except the identified one. Additionally, remove any unnecessary tags (e.g., button, link) within the identified tag. Finally, modify the text to conform to markdown syntax based on the characteristics of the remaining tags. For tags like `<h1>`, `<h2>`, ..., `<h6>`, append a corresponding number of asterisks '*' to signify markdown syntax. For `<a>` tags representing links, remove the hyperlink to prevent it from appearing in the text. For `<ul>` tags representing unordered lists, prepend each item with '-' and a newline character. For `<ol>` tags representing ordered lists, prepend each item with a number and a newline character. Tags that alter the text's appearance, such as `<b>`, `<em>`, `<strong>`, etc., are configured differently on the domain.

#### 4.3.2. Image parsing

Images complement the descriptions in the text and contain implicit information. They are also important for creating interleaved text-image datasets when used together with text, which can improve explainability. To obtain images, the URL of the image, which is found in the src attribute is needed. But often only the relative URL of the domain is provided. Therefore, it is crucial to identify the base URL of the domain hosting the image. For this, we manually verified and set unique base URLs for each domain. Another issue is the excessive presence of irrelevant images, such as site logos and button icons, in the content. To address this, we excluded unrelated image tags using the method described in Section 4.3.1.

#### 4.3.3. Wikipedia Infobox

Wikipedia features high-quality documents created collaboratively by multiple contributors. Among these documents, the infobox summarizes specific objects, encapsulating their characteristics in tabular data. Parsing tables is challenging due to the variety in size and shape, which makes it difficult to apply a single rule for parsing. However, based on our observation that infoboxes are typically composed of two columns, we have implemented a parser specifically for two-column tables. As we iterate through each row of the table, if there are only two columns, we designate the value of the first column as the key and the value of the second column as the value, and proceed with parsing. This approach allows us to adequately maintain the essential summary information provided in Wikipedia's infoboxes.

#### 4.3.4. Subscript for chemical formula

In chemical reactions, chemical formulas are commonly used to represent the reaction in text. The numbers indicating the number of molecules, displayed as subscripts, are typically represented by the `<sub>` tag in HTML. The `<sub>` tag is also commonly used in markdown syntax; consequently, we have implemented a feature in our markdown to ensure that sections wrapped in `<sub>` and `</sub>` tags are displayed exactly as they appear, maintaining the tag format.

### 4.4. Data Restructuring Task 1: Document Augmentation

Here, we describe the first data restructuring task using information obtained from the parser: document augmentation. The markdown documents obtained from web pages are denoted as $\mathcal{D} = D_1, D_2, \cdots, D_n$, and the insufficient or outdated documents are denoted as $D_{\text{out}}$[1].

#### 4.4.1. Concatenation

Although we used ReactionParser to convert raw HTML to markdown format to reduce the number of tokens handled by the LLM, including all parsed documents $\mathcal{D}$ and the original document $D_{\text{out}}$ simultaneously can still result in a high token count. This may prevent the LLM from processing them all at once, especially if it does not support long contexts. For example, the number of tokens of parsed document sometimes exceeds 32k, while open-source LLMs such as Llama-3 (AI@Meta, 2024) can only deal with 8k tokens. Therefore, we consolidate the information by merging all documents and then dividing the text into similarly sized chunks, summarizing each one, and recombining them. To achieve this, we first concatenate all the parsed documents.

#### 4.4.2. Image Placeholder Substitution

The parsed documents $\mathcal{D}$ might include image syntax like `![description](URL)`, but some URLs are extremely long due to Unicode characters. We observed that these lengthy URLs often cause LLMs to output odd characters when used for downstream tasks. To address this issue and the difficulties LLMs have in interpreting lengthy URLs, we simplified image syntax in all documents using regular expressions, replacing them with a simpler sequence, `<FIGURE_`$I$`>`, where $I$ is the image index. We refer to the replaced string `<FIGURE_`$I$`>` as an image placeholder and the process of replacing it as image placeholder substitution.

[1] $q$ is omitted for simplicity

#### 4.4.3. Extracting relevant information

To focus solely on information relevant to a query $q$ and reduce unrelated content, we divided the concatenated document into 10,000-character segments. Through the loop, we use an instruction-tuned Llama-3 with a specific prompt, $P_{\text{extract}}$, to extract the related segments, $\{d_1, d_2, \cdots, d_k\}$.

#### 4.4.4. Incomplete document integration

Subsequently, we use this extracted information along with another prompt, $P_{\text{integrate}}$, to generate a new document, $D'$, by integrating the information into a prior document, $D_{\text{out}}$.

The overall process is mathematically represented as follows:

$$d_1, d_2, \cdots, d_k = \text{LLM}(D_1, D_2, \cdots, D_n | P_{\text{extract}}) \quad (1)$$

$$D' = \text{LLM}(D_{\text{out}}, d_1, d_2, \cdots, d_k | P_{\text{integrate}}) \quad (2)$$

#### 4.4.5. Image placeholder restoration

The integration result may include the `<FIGURE_`$I$`>` string, representing the image placeholder we saved. This image placeholder is then replaced with the original syntax string `![description](URL)`, a process we call image placeholder restoration. The whole process is shown in Figure 1 and the prompts $P_{\text{extract}}$ and $P_{\text{integrate}}$ that we designed and used are available at the public URLs $P_{\text{extract}}$ and $P_{\text{integrate}}$[2].

### 4.5. Data Restructuring Task 2: Knowledge Graph Completion

In this subsection, we introduce a KG completion method as another data restructuring task, which involves enhancing an incomplete or prior knowledge graph by adding content found based on search.

#### 4.5.1. Base knowledge graph

To create our base KG, we utilized Wikidata (Vrandečić & Krötzsch, 2014). We downloaded the Wikidata dump file and collected all entities pointing to chemical reactions (QID 36534), thereby creating a KG that represents chemical reactions. Specifically, we utilize SPARQL (201, 2013) to get the chemical reaction-related Wikidata pages. This graph does not include unique chemical reactions discussed in subsection 4.1 and mainly comprises well-known chemical reactions. We stored this KG using Neo4j (Neo4j, 2012), which is advantageous for handling large-scale graph data. Neo4j allows for various operations such as adding data or finding relationships through the Cypher (Francis et al., 2018) query language.

[2] Due to the length of the prompt, we share all prompts of LLMs used in this paper via a public, anonymized URL

#### 4.5.2. Generating Cypher Query

To add data to the knowledge graph using Neo4j, Cypher queries must be generated. Based on the capability of LLMs to extract triplets from raw text, we used an LLM-based integration and Cypher query generation method. Similar to the documentation creation method described in section 4.4, we use the LLM(*gpt-4-turbo-2024-04-09*) with relative segments $\{d_1, d_2, \cdots, d_k\}$, which contain information about query $q$, and the existing knowledge graph's schema (state of knowledge graph) $S$. The schema contains entity types, properties of entities, relationship types, properties of relationships. However, there is no way to get information about existing entity names, so we additionally include existing chemical reaction entity names with the schema. This approach enables the integration of information found through search with the existing knowledge graph. The detailed prompt for Cypher query generation is represented at the URL $P_{\text{cypher}}$.

## 5. Experiment

In Section 3, we introduced various options to consider, and in Section 4, we presented our final framework using ReactionParser as the HTML parser. However, in our experiments, the main focus is on comparing different HTML parsers, as this has the most significant impact on data reconstruction. We conduct comparisons against existing parsers, such as `Pandoc`, `html2markdown`, `html2text`, `Markdownify`, and `BeautifulSoup4 (BS)`. All parsers, except for `BS`, are used to convert HTML into markdown format, while `BS` is used to convert HTML into plain text. Although ReactionParser utilizes `BS` to handle HTML tags, in this context, `BS` is employed as a basic text parser, and we use its function to convert directly to text without preprocessing for comparison purposes.

In this section, first, we examine results generated using ReactionParser and compare them against existing alternatives. Then we analyze the quality of the restructured documents[3] and knowledge graphs after supplementing them with information from a parser. The key points of focus are as follows: How effectively can the specialized parser supplement incomplete documents? Section 5.1. Another point of interest is how much information is added when the specialized parser supplements a knowledge graph, (Section 5.2). Additionally, the effectiveness of ReactionParser in parsing relevant content compared to other methods is evaluated, with detailed comparisons available in Appendix A.1.

[3]All parsed text and generated documents can be found at https://github.com/ReactionParser/ReactionParser/tree/main/results

### 5.1. Parsers' Impact on Final Document Augmentation

In this experiment, we conduct a document augmentation task to investigate the effects of web data parsers on LLM-based document synthesis from factual sources. Specifically, we aim to analyze how well the LLM retrieves and reprocesses useful information from different parsed web data results. Due to the high variance in structure and content of documents produced solely from parsed results, we started with an initial chemical reaction document and augmented it. This serves as regularization, ensuring most documents follow a similar pattern. We then compare the augmented document results with data reprocessed by the LLM. To create the *base document*, we used various closed and open-source LLM alternatives with different sizes and performance. The closed LLMs used for document generation include *gpt-4-turbo-2024-04-09*, *gpt-3.5-turbo-0125*, *claude-3-haiku-20240307*, *claude-3-sonnet-20240229* and *claude-3-opus-20240229*. The open-source LLMs used include *Gemma-2B*, *Gemma-7B*, *Phi-3-3.8B*, and *Mistral-7B*, to explore models with different scales and general performance.

#### 5.1.1. Metrics

To assess the usefulness of the information identified by ReactionParser in the document augmentation, we use an LLM-based evaluation method. We conduct a relative evaluation using the LLM. We designed the evaluation prompt, $P_{\text{compare}}$, to measure informativeness about chemical reaction queries. The prompt is shown in URL $P_{\text{compare}}$. When providing documents A to G in the prompt, the evaluation LLM outputs the likelihood of selecting the next token. The score is then calculated by summing the likelihoods of the tokens representing each document's identifier. However, a single letter can be represented by various tokens. In this experiment, using *gpt-4-turbo-2024-04-09* for evaluation, tokens such as `"A"` and `"<A"` are used to represent the letter A. For an arbitrary letter $X$ representing the index of a document, we use $T_X$ to denote the set of different tokens representing the same letter. Given the augmented documents $D_A, D_B, \cdots, D_G$, and the base document $D_O$, the score of a document $S(D_X)$ is calculated as $S(D_X) = \sum_{x \in T_X} p(t = x | D_A, D_B, \cdots, D_G, D_O, P_{\text{compare}})$

In addition to the score, we introduce *gain* and *gain per token* metrics to demonstrate how much information from the parsed data is presented concisely related to the query. *Gain* represents the relative preference of each augmented document compared to the base document, which is set to zero. It is calculated as $S(D_X) - S(D_O)$ for any document $X$. *Gain per token* is calculated by dividing the *gain* by the total number of tokens $(N_T)$ used to generate the final document: $(S(D_X) - S(D_O))/N_T$. The total number of tokens includes 1) tokens used when extracting relevant

*Table 1.* Comparison with various open-source parsers when incorporating information found through search into document augmentation.

| | Gain↑ | Gain per token (M)↑ |
|---|---|---|
| LLM only | 0.000 | - |
| LLM+Pandoc | 0.078 | 2.345 |
| LLM+html2markdown | **0.099** | 1.344 |
| LLM+html2text | 0.062 | 2.581 |
| LLM+Markdownify | 0.087 | 3.885 |
| LLM+BS | 0.059 | 3.942 |
| LLM+ReactionParser | 0.075 | **9.235** |

sentences and 2) tokens used when generating the final augmented document. We call 1) the extraction token count (ETC). The ETC for all augmented documents is shown in Table 5 in Appendix.

#### 5.1.2. Results

We compare a total of seven final documents: five created using open-source parsers, one with ReactionParser, and one without any external source. The results are shown in Table 1. For each document, we used a set of 20 queries (named chemical reactions) and averaged the results across them. In Table 1, the rows indicate which parser is used. Compared to the base document, all parsers' outputs showed a relative improvement in terms of useful content, resulting in a higher preference. However, in terms of *Gain*, there were no significant differences in results among the different parser methods. When comparing the number of tokens to represent the same information, we can see that a specialized parser offers advantages. The *Gain per token* column in Table 1 shows the gain in score per token. For instance, using `Pandoc` to extract content from the raw HTMLs yields a similar amount of information as ReactionParser, but with approximately four times more tokens used. This indicates that using specialized parsers preserves the same amount of information while also being more cost-effective when restructured by the LLM. The left graph in Figure 2 shows a comparison of *Gain per token* for each parser, and full results are available in Table 4 in Appendix.

### 5.2. Parsers' Impact on Knowledge Graph Completion

This experiment, similar to the one in Section 5.1, aims to examine the impact of the parsed content quality on the task of knowledge graph completion. To create the initial knowledge graph, we use Wikidata (Vrandečić & Krötzsch, 2014) to produce a knowledge graph representing chemical reactions, as described in Section 4.5.1. The process for generating entities and relationships is carried out using the *Llama-3 70B*, as explained in Section 4.5.2.

*Table 2.* Comparison of the number of newly obtained entities (Entity gain) and triplets (Triplet gain) during knowledge graph completion using each parser. In the *Gain* column, the number to the left of the '/' represents the Entity gain, while the number to the right represents the Triplet gain.

| | Gain↑ | Gain per token (K)↑ |
|---|---|---|
| Wikidata+Pandoc | 254/158 | 1.382 |
| Wikidata+html2markdown | **337**/143 | 0.690 |
| Wikidata+Markdownify | 308/122 | 2.050 |
| Wikidata+html2text | 269/116 | 1.981 |
| Wikidata+BS | 246/146 | 3.276 |
| Wikidata+ReactionParser | 293/**166** | **6.810** |

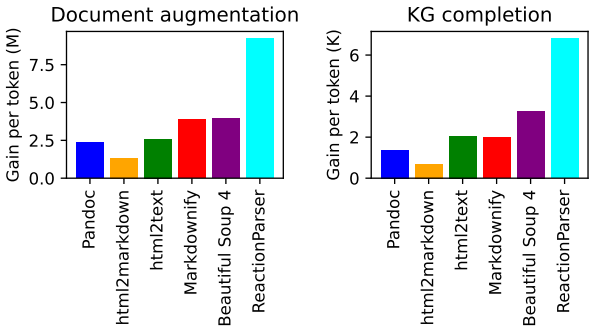


*Figure 2.* Comparison of Gain per token in different data restructuring tasks.

#### 5.2.1. Metrics

Unlike Section 5.1, where we explore how much information was successfully preserved by each parser in raw text form, here we focus on measuring the amount of information added to the knowledge graph (KG) through new entities and triplets. To ensure only chemical reaction-related information is added during KG query generation, we designed the prompt to specifically include information about chemical reactions, although this is not guaranteed. Additionally, similar to the experiment in Section 5.1, we compare the increase in information by measuring the growth in entities and triplets (*gain*) and dividing it by the number of tokens used for query generation. This ratio, *gain per token*, is used as a metric for the amount of information per token. The total number of tokens includes the extraction token count (ETC), similar to 5.1, and the query generation token count (QTC). We calculate the final *gain per token* using the formula: $(\text{Entity gain} + \text{Triplet gain})/(\text{ETC} + \text{QTC})$.

#### 5.2.2. RESULTS

Similar to Section 5.1, we compare the augmented KGs, which are enhanced by parsed information, against the base KG created using only Wikidata. We use the same 20 queries as in Section 5.1. The results are shown in Table 2. The chemical reaction KG created using only Wikidata has 8,243 entities and 22,043 triplets. When the parsed document information is incorporated, approximately 200-300 entities and 110-160 triplets are added. The number of entities added is about twice the number of triplets, which is expected because adding two different entities results in the addition of only one triplet. We find that ReactionParser handles a larger amount of information with fewer tokens, similar to the results in Section 5.1. Compared to Wikidata + `BS`, which has the highest gain per token, ReactionParser contains approximately 108% more information given the same token usage. This indicates that, as in the results of Section 5.1, using a specialized parser is more cost-effective when the LLM reprocesses the parsed information for downstream tasks. The right graph in Figure 2 shows a comparison of *Gain per token* for each parser, and full results are available in Table 6 in Appendix.

## 6. Limitations

A primary consideration when developing our pipeline including ReactionParser was the issue of data copyright and licenses. Recently, copyright infringement related to the use of news articles for training language models has become a pressing issue, and companies are addressing this by collaborating with data holders. In this work, we utilized web pages' content solely for research and experimental purposes. We avoided instances where general access is restricted by the server (e.g., a 403 error) or where licensing issues were identified. For commercial use or training of language models, it is imperative to verify the licensing agreements of each site.

Additionally, the parser implementation explored here requires extra modifications if it is to be extended to other scientific topics. Even though we have structured our code implementation in a way that can serve as a recipe when building specialized parsers for new scientific topics, special tuning is still required due to the diversity of internet sources.

As for document synthesis, another issue concerns the uncertainty associated with the use of language models. Even though open-source and proprietary pre-trained language models have improved at an unprecedented rate over the last few years, natural issues such as hallucinations, instruction following, steerability, and context length are still prevalent. For instance, although we instruct LLMs to focus on certain aspects when generating supplementary documents, there is no guarantee that the LLMs will adhere to these instructions. Also, when extracting related content, there is a risk of missing relevant details or not capturing the exact content accurately. However, we believe the ability of these models to perform these tasks will continue to improve and parsing/restructuring data will still be a critical step for scientific document synthesis.

## 7. Conclusions and Future Work

In this study, we share the considerations and our observations when reconstructing scientific data using LLMs. Specifically, we developed an HTML parser called ReactionParser, which has the most significant impact on the cost of using LLMs, and based on this, we created an overall data reconstruction framework. This framework demonstrated that it can significantly reduce the cost of using LLMs in two data restructuring tasks: document augmentation and knowledge graph completion. We believe that the contents of this paper will be helpful for other data reconstruction tasks as well. Although we conducted our exploration and experiments with the theme of chemical reactions in this paper, we will expand to a broader range of scientific themes in the future. Additionally, we will extend the framework to handle not only text and graph data but also other forms of structured data, which will provide a variety of data for training AI models for science.

## Impact Statement

We do not anticipate a significant social impact from the findings of this study. However, we acknowledge that the use of LLMs for data reconstruction may result in substantial electricity consumption, thereby posing an environmental burden.

## References


SPARQL 1.1 Query Language. Technical report, W3C, 2013. URL http://www.w3.org/TR/sparql11-query.

AI@Meta. Llama 3 model card. 2024. URL https://github.com/meta-llama/llama3/blob/main/MODEL_CARD.md.

Anthropic. Introducing the next generation of claude. URL https://www.anthropic.com/news/claude-3-family. Accessed on 2024.03.04.

Bran, A. M., Cox, S., Schilter, O., Baldassari, C., White, A. D., and Schwaller, P. Chemcrow: Augmenting large-language models with chemistry tools. *arXiv preprint arXiv:2304.05376*, 2023.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D.,

Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901, 2020.

Chase, H. LangChain, October 2022. URL https://github.com/langchain-ai/langchain.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Francis, N., Green, A., Guagliardo, P., Libkin, L., Lindaaker, T., Marsault, V., Plantikow, S., Rydberg, M., Selmer, P., and Taylor, A. Cypher: An evolving query language for property graphs. In *Proceedings of the 2018 international conference on management of data*, pp. 1433–1445, 2018.

Liu, Y., Iter, D., Xu, Y., Wang, S., Xu, R., and Zhu, C. Gpteval: Nlg evaluation using gpt-4 with better human alignment. *arXiv preprint arXiv:2303.16634*, 2023.

Lu, J., Yu, L., Li, X., Yang, L., and Zuo, C. Llama-reviewer: Advancing code review automation with large language models through parameter-efficient fine-tuning. In *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*, pp. 647–658. IEEE, 2023.

Neo4j. Neo4j - the world's leading graph database, 2012. URL http://neo4j.org/.

NextMove. Nextmove software pistachio. URL https://www.nextmovesoftware.com/pistachio.html. Accessed on 2023.12.11.

Qureshi, R., Shaughnessy, D., Gill, K. A., Robinson, K. A., Li, T., and Agai, E. Are chatgpt and large language models "the answer" to bringing us closer to systematic review automation? *Systematic Reviews*, 12(1):72, 2023.

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

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Shantanu, Bartolome, A., Lunnemark, A., Obermeyer, F., Torget, H., Kilpatrick, L., Pap, L., Mariatta, Welsh, M., Stathas, N., Nigam, P., Sanders, T., Lu, X. H., jonathanagustin, and youkaichao. *openai/tiktoken*. 4 2024. URL https://github.com/openai/tiktoken.

Stewart, S., Barantsev, A., jimevans, Fortner, T., jleyba, Bakken, J., Burns, D., Molina, D., Fabulich, D., Wagner-Hall, D., Rodionov, A., Inman-Semerau, L., Jagani, P., Messeri, E., Williams, M., andreastt, García, B., Rosenvold, K., Huggins, J., K, S., Hunt, D., Chai, H.-B., Harsha, S., Tierney, L., joerg1985, Reynaud, F., Badle, S., Menard, K., sevaseva, and Xiong, J. *SeleniumHQ/selenium*. 5 2024. URL https://github.com/SeleniumHQ/selenium.

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

Vrandečić, D. and Krötzsch, M. Wikidata: a free collaborative knowledgebase. *Communications of the ACM*, 57 (10):78–85, 2014.

Wang, L., Ma, C., Feng, X., Zhang, Z., Yang, H., Zhang, J., Chen, Z., Tang, J., Chen, X., Lin, Y., et al. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):1–26, 2024.

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

Zhou, Y., Liu, H., Srivastava, T., Mei, H., and Tan, C. Hypothesis generation with large language models. *arXiv preprint arXiv:2404.04326*, 2024.

# A. Appendix

## A.1. Parsers Comparison

We first evaluate how well ReactionParser processes the targeted web pages. To do this, we compare ReactionParser with open-source parsers such as `Pandoc`, `html2markdown`, `html2text`, `Markdownify`, and `BS`. In our experiments, we randomly select 100 web pages from the search results of named chemical reaction-related queries and benchmark across all parsers.

### A.1.1. Metrics

One major challenge in evaluating the quality of parser outputs is the lack of ground truth data, i.e., assessing the balance between relevant and irrelevant information in terms of semantic content. Specifically, ReactionParser addresses the underexplored topic of chemical reactions, which naturally leads to scarce reference answers. This issue is similarly encountered in recent natural language generation tasks. G-Eval (Liu et al., 2023) offers a method to evaluate generated text in scenarios lacking reference texts via LLMs. In G-Eval, different criteria are predefined depending on the task—for example, fluency, coherence, and consistency are used in summarization tasks. Since we aim to measure the quality of the parsed content, we use criteria such as **structure**—how well structured in the output text; **relevance**—relative to the full parsed content how much of it is related to the original query; and **coherence**—how unified the content is. Specifically, G-Eval completes the evaluation prompt using the Chain-of-Thought (CoT) (Wei et al., 2022) prompting technique. Here, we explain how our evaluation prompts were defined. An example evaluation prompt for the output's structure quality is shown in URL $P_{\text{structure}}$. As indicated in the input section of the file $P_{\text{structure}}$, we specify the evaluation criteria as a prompt and generate an `evaluation rubric` (a series of evaluation points or steps) using a pretrained LLM. The generated evaluation steps are then concatenated to the system message of an instruction-tuned LLM (gpt-4-turbo-2024-04-09) to obtain our final evaluation prompt. Based on this prompt, an instruction-tuned LLM generates a score $s \in \{1, 2, 3, 4, 5\}$ for the parsed document. However, this approach can result in scores being concentrated within a narrow range. To address this, G-Eval proposes to multiply the generated score by the probability of the assigned score, leading to a more uncertainty-informed metric. The final score can be defined as follows:

$$\sum_{s=1}^{5} s \cdot p(t = "s" | D, P_{\text{structure}}) \tag{3}$$

This formula allows us to obtain more representative and evenly distributed scores. Similarly, evaluation prompts for relevance and coherence were generated and are available at: $P_{\text{relevance}}$ and $P_{\text{coherence}}$. The final score reported for each category is the mean score across all test data. The overall average score is calculated as the mean of the scores for *Structure*, *Relevance*, and *Coherence*. Additionally, we calculate the number of tokens in the parsed results. The fewer tokens, the better signal-to-noise ratio in the parsed contents which directly translates to lower computational costs for the downstream LLM responsible for synthesizing our final documents. We measure the number of tokens using tiktoken (Shantanu et al., 2024), which is used in GPT-3.5 and GPT-4.

### A.1.2. Results

We used a random sample of 100 web pages from search results and averaged their scores. The results of comparing different parsers using the metrics described above are shown in Table 3. Overall, ReactionParser demonstrated the best performance in *Structure*, *Relevance*, and *Coherence*, leading to superior average performance. The `html2markdown` parser produced nearly unparsed HTML, showing particularly poor performance in terms of *Coherence* and resulting in a high token count due to the presence of needless HTML tags. For `Pandoc`, while the structure of chemical reaction content was generally well-maintained, a significant amount of unparsed CSS class code within the HTML still remained, leading to a lower *Relevance* score. Both `html2text` and `Markdownify` showed generally good performance with well-parsed markdown formats, but they included a lot of irrelevant content, leading to slightly lower performance compared to ReactionParser and a relatively higher token count. In the case of `BS`, it parsed only the text, excluding images, resulting in the second-lowest token count after ReactionParser. However, its performance was hindered by irrelevant content, excessive whitespace, and lack of proper formatting between section names and the main contents.

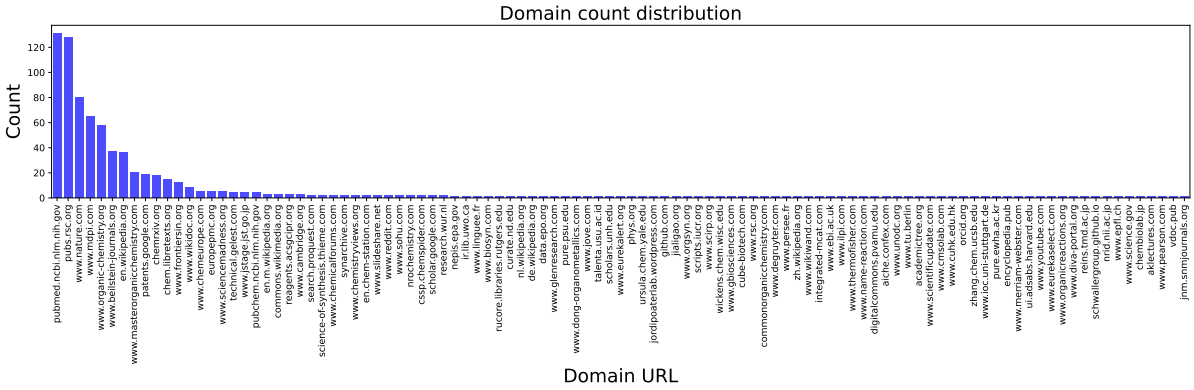


*Figure 3.* Domain histogram of search results. $x$-axis shows the domain and $y$-axis represents its count.

*Table 3.* Comparison of HTML parsing performance between open source parsers and ReactionParser. The measurements include how well the content is structurally parsed (Structure), the relevance of the parsed content to the searched chemical reactions (Relevance), and the coherence of the parsed text (Coherence). The last column also shows the number of tokens representing the parsed content (# of tokens).

| | Structure↑ | Relevance↑ | Coherence↑ | Avg↑ | # of tokens↓ |
|---|---|---|---|---|---|
| Pandoc | 3.11 | 2.45 | 3.31 | 2.95 | 36,719 |
| html2markdown | 2.33 | 2.76 | 1.74 | 2.27 | 70,889 |
| html2text | 3.48 | 2.58 | 3.73 | 3.26 | 26,837 |
| Markdownify | 2.95 | 3.60 | 3.15 | 3.23 | 26,173 |
| BS | 2.57 | 3.25 | 2.80 | 2.87 | 16,232 |
| ReactionParser | **3.61** | **3.86** | **4.00** | **3.82** | **11,280** |

*Table 4.* Comparison with various open-source parsers when incorporating information found through search into document augmentation. The LLMs listed in the columns represent the names of the LLMs that created the base documents without the added search information (LLM only). For evaluation, another LLM (GPT-4) is used to assess how much sufficient information is included. *Gain* is a metric normalized to zero for the base document, and *Gain per token* is the metric divided by the number of tokens used to create the document.

| Informativeness | | | | |
|---|---|---|---|---|
| | GPT-4↑ | GPT-3.5↑ | Claude-Opus↑ | Claude-Sonnet↑ |
| LLM only | 0.025 | 0.000 | 0.000 | 0.000 |
| LLM+Pandoc | 0.074 | 0.071 | 0.104 | 0.050 |
| LLM+html2markdown | 0.100 | **0.136** | 0.023 | **0.125** |
| LLM+html2text | 0.075 | 0.125 | 0.118 | 0.025 |
| LLM+Markdownify | 0.050 | 0.062 | **0.125** | 0.075 |
| LLM+BS | **0.107** | 0.051 | 0.080 | 0.075 |
| LLM+ReactionParser | 0.069 | 0.055 | 0.050 | 0.149 |
| | Claude-Haiku↑ | Gemma-2B↑ | Gemma-7B↑ | Phi-3-3.8B↑ |
| LLM only | 0.000 | 0.000 | 0.000 | 0.000 |
| LLM+Pandoc | 0.050 | **0.123** | 0.050 | **0.180** |
| LLM+html2markdown | 0.108 | **0.154** | 0.099 | **0.099** |
| LLM+html2text | 0.075 | 0.101 | 0.051 | 0.000 |
| LLM+Markdownify | **0.093** | 0.042 | **0.150** | 0.100 |
| LLM+BS | 0.025 | 0.025 | 0.075 | 0.068 |
| LLM+ReactionParser | **0.149** | 0.055 | 0.075 | 0.052 |
| | Mistral-7B↑ | Average↑ | Gain↑ | Gain per token (M)↑ |
| LLM only | 0.025 | 0.006 | 0.000 | - |
| LLM+Pandoc | 0.051 | 0.084 | 0.078 | 2.345 |
| LLM+html2markdown | 0.100 | **0.105** | **0.099** | 1.344 |
| LLM+html2text | 0.039 | 0.068 | 0.062 | 2.581 |
| LLM+Markdownify | **0.138** | 0.093 | 0.087 | 3.885 |
| LLM+BS | 0.075 | 0.064 | 0.059 | 3.942 |
| LLM+ReactionParser | 0.072 | 0.081 | 0.075 | **9.235** |

*Table 5.* Comparison of token counts resulting from using different parsers in the document augmentation task. Extraction Token Count (ETC) represents the number of tokens used by the LLM to extract content related to the searched chemical reactions query from each parsed document.

| Token spent | GPT-4↓ | GPT-3.5↓ | Claude-Opus↓ | Claude-Sonnet↓ |
|---|---|---|---|---|
| LLM+Pandoc | 4,331 | 3,570 | 4,087 | 3,784 |
| LLM+html2markdown | 5,232 | 4,543 | 5,020 | 4,749 |
| LLM+html2text | 3,871 | 3,105 | 3,623 | 3,346 |
| LLM+Markdownify | 3,730 | 3,012 | 3,501 | 3,182 |
| LLM+BS | 3,682 | 2,980 | 3,541 | 3,173 |
| LLM+ReactionParser | **2,828** | **2,068** | **2,570** | **2,246** |
| | Claude-Haiku↓ | Gemma-2B↓ | Gemma-7B↓ | Phi-3-3.8B↓ |
| LLM+Pandoc | 3,840 | 4,015 | 4,122 | 3,708 |
| LLM+html2markdown | 4,807 | 4,942 | 4,987 | 4,634 |
| LLM+html2text | 3,347 | 3,530 | 3,630 | 3,242 |
| LLM+Markdownify | 3,301 | 3,478 | 3,507 | 3,162 |
| LLM+BS | 3,242 | 3,390 | 3,470 | 3,082 |
| LLM+ReactionParser | **2,323** | **2,465** | **2,571** | **2,169** |
| | Mistral-7B↓ | ETC↓ | Average↓ | |
| LLM+Pandoc | 4,387 | 297,267 | 33,311 | |
| LLM+html2markdown | 5,326 | 694,824 | 73,906 | |
| LLM+html2text | 3,959 | 208,903 | 24,056 | |
| LLM+Markdownify | 3,875 | 193,432 | 22,418 | |
| LLM+BS | 3,821 | 118,859 | 14,924 | |
| LLM+ReactionParser | **2,850** | **59,365** | **8,146** | |

*Table 6.* Comparison of the number of newly obtained entities (Entity gain) and triplets (Triplet gain) during knowledge graph completion using each parser. Extraction Token Count (ETC) represents the number of tokens used by the LLM to extract content related to the searched chemical reactions from each parsed document. Query Generation Token Count (QTC) represents the number of tokens used by the LLM to generate Cypher queries for adding data to the knowledge graph database. The final *Gain per token* is calculated as (Entity gain+Triplet gain)/(ETC+QTC), considering both the entity and triplet counts along with ETC and QTC.

| | ETC↓ | QTC↓ | Entity gain↑ | Triplet gain↑ | Gain per token (K)↑ | Entities | Triplets |
|---|---|---|---|---|---|---|---|
| Wikidata | - | - | - | - | - | 8,243 | 22,043 |
| Wikidata+Pandoc | 297,267 | 950 | 254 | **158** | 1.382 | 8,497 | 22,201 |
| Wikidata+html2markdown | 694,824 | 932 | **337** | 143 | 0.690 | 8,580 | 22,186 |
| Wikidata+Markdownify | 208,903 | 836 | 308 | 122 | 2.050 | 8,551 | 22,165 |
| Wikidata+html2text | 193,432 | 870 | 269 | 116 | 1.981 | 8,512 | 22,159 |
| Wikidata+BS | 118,859 | 787 | 246 | 146 | 3.276 | 8,489 | 22,189 |
| Wikidata+ReactionParser | **59,365** | **692** | 293 | 116 | **6.810** | 8,536 | 22,159 |