A Cost-Efficient Modular Sieve for Extracting Product Information from Company Websites

Anna Hätty[1*] **Dragan Milchevski**[1*] **Kersten Döring**[2*] **Marko Putnikovic**[4*] **Mohsen Mesgar**[1*]
Filip Novovic[4] Maximilian Braun[2] Karina Borimann[3] and Igor Stranjanac [4]
[1]Bosch Center for Artificial Intelligence, Renningen, Germany
[2]Bosch Global Services, Feuerbach, Germany
[3]Bosch Digital, Feuerbach, Germany
[4]Bosch Digital, Belgrade, Serbia
firstname.lastname@bosch.com

Abstract

Extracting product information is crucial for informed business decisions and strategic planning across multiple industries. However, recent methods relying only on large language models (LLMs) are resource-intensive and computationally prohibitive due to differences in website structures and numerous non-product pages. To address these challenges, we propose a novel modular method that leverages low-cost classification models to filter out company web pages, significantly reducing computational costs. Our approach consists of three modules: web page crawling, product page classification using efficient machine learning models, and product information extraction using LLMs on classified product pages. We evaluate our method on a new dataset comprising approximately 7,000 product and non-product web pages, achieving a 6-point improvement in F1-score, a 95% reduction in computational time, and an 87.5% reduction in cost compared to end-to-end LLMs. Our research demonstrates the effectiveness of our proposed low-cost classification module to identify web pages containing product information, making product information extraction more effective and cost-efficient.

1 Introduction

Information (e.g., names and descriptions) about products that a company offers is essential for numerous applications such as product search (Wei et al., 2013; Brinkmann et al., 2023b), product recommendation (Malik et al., 2022), and product knowledge graph construction (Zalmout et al., 2021; Deng et al., 2023). Developing a method for obtaining product information is challenging due to (1) the exponential growth of companies and their web pages, which may or may not contain product information; and (2) an unknown structure of company product pages across different company websites. Therefore, it is imperative to develop an automated and cost-efficient method to deal with the ever-increasing number of web pages and also handle non-unified structure of product pages.

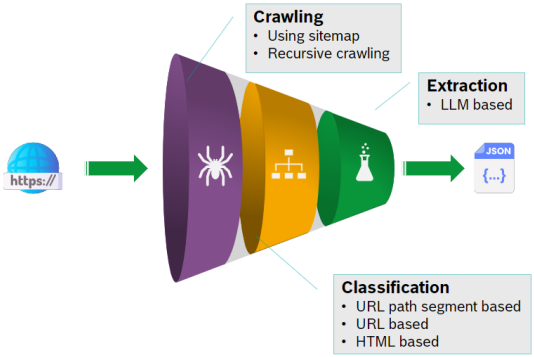


Figure 1: A general depiction of our method, including its three modules: web page crawling, product page classification, and product information extraction.

Previous work has primarily focused on processing product information from e-commerce web shop data (Zou et al., 2024; Gong and Eldardiry, 2024; Ding et al., 2022; Roy et al., 2021; Yan et al., 2021). However, in combination with different page structures across companies, such methods would fall short if dealing with all web pages on a company website, where these pages may or may not be product pages. Moreover, including non-product pages into the input of a product information extraction system increases **computational costs** due to the huge number of web pages to process. Another challenge is that these pages might be misleading because they could have a structure similar to product pages and diminish **the quality of product information**. To the best of our knowledge, there is no full method that collects company web pages from diverse company websites and extracts product information from such pages while balancing computational costs and the quality of the obtained product information.

[*] indicates equal contribution of the first five authors. All authors are listed in alphabetical order by first name.

Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Industry Track, pages 1444–1456
November 12-16, 2024

In this paper, we propose a novel full pipeline for extracting product information from all web pages on a company website. In contrast to previous work, our method handles pages that may or may not contain information about products. In particular, we extract the product name and product description presented on a company website. Our method consists of three major modules (see Figure 1): (1) crawling, (2) classification, and (3) extraction. In the crawling module, we scrape the company website to collect web pages. For the classification module, we introduce three models to distinguish between product and non-product web pages. Finally, for the extraction module, we instruct a pre-trained LLM to find product names and descriptions on product web pages. LLMs have demonstrated effectiveness in extracting information from unknown structured inputs (Hui et al., 2024; Wang et al., 2023).

To evaluate the quality and computational cost of our method, we curated a new dataset comprising product and non-product web pages from diverse companies. The product pages are annotated by a product scouting expert. To ensure the robustness of our method, we split the dataset at the company level to include only unseen companies in the test set. Our experimental results show that our method outperforms off-the-shelf LLMs in terms of computational cost efficiency while achieving better quality then its peers. To measure the quality, we use precision, recall, and F1 score to assess whether a method identifies correct product pages. We also use ROUGE (a recall oriented lexical metric) and BERTScore (an advanced semantic similarity metric) to evaluate the correctness of the extracted product names and descriptions.

Our main contributions are: **(i) Task:** While extracting product information from product pages is known, finding products from heterogeneous web pages across company websites has not been studied. **(ii) Sieve method:** We introduce a modular method to identify product pages from a company website and then use them to extract product information. Worth noting that, our novelty in method is the entire pipeline that reduces input space for extraction. **(iii) Empirical evaluation:** We collect and annotate a new dataset that contains a representative cross-section of company websites' product and non-product pages. Our experiments demonstrate that our method is effective and cost-efficient compared to an off-the-shelf LLM.

2 Related work

Product information extraction involves extracting attribute/value pairs (specifications) from product information such as name and description. Extraction can be performed using a closed-world assumption with a predefined set of attributes, or an open-world assumption where attributes are unknown (Zheng et al., 2018). The open-world assumption is more suitable for extracting product information from unstructured data obtained through crawling.

Product information extraction approaches can be categorized into four major groups: (1) rule-based methods, (2) sequence tagging and named-entity recognition (NER), (3) extractive question answering, and (4) generative approaches. Rule-based methods often use token-matching techniques, such as regular expressions, to extract attribute/value pairs (Gopalakrishnan et al., 2012). These methods lack scalability, as a new rule is required for each new attribute (Wang et al., 2020). The sequence labeling approach often involves constructing a model for each attribute (Yan et al., 2021, Zheng et al., 2018), which also does not scale and generalize well. To address this, question answering approaches consider each attribute as a question - the task is to identify the attribute value as the answer (Ding et al., 2022). For instance, Wang et al., 2020 use a single BERT model to encode both the context (product information) and question, which makes the approach scalable and generizable. A drawback is that this approach is not suitable for extracting implicit product information, i.e. one that is not explicitly mentioned in the product text (Blume et al., 2023). This problem is resolved by recent API-based large language models (LLMs), such as GPT-3.5, used to generate attribute/value pairs based on the product information provided on the product web page (Gong and Eldardiry, 2024; Blume et al., 2023; Zou et al., 2024; Brinkmann et al., 2023a). We employ a generative approach as the latest state-of-the-art in product information extraction (Gong and Eldardiry, 2024) and focus on generating product name and description from the data available on a product page.

3 Method

We develop a novel method to extract products and their descriptions from company web pages. Figure 2 depicts details of our method.

3.1 Crawling

The crawling module consists of two main components: URL collection and HTML scraping. To ensure compliance, we respect the *robots.txt* for each company domain. For the URL collection step, we design two approaches.

Sitemap-based crawler. Our first approach is based on sitemaps available on company websites. A sitemap is a hierarchical structure of web pages on a company website used to navigate the website accurately. By traversing a sitemap, we collect URLs in a computationally efficient manner. However, this approach may not be effective if companies do not provide a sitemap, or if sitemaps are outdated (not including all web pages from the latest website versions).

Recursive crawler. We start with all hyperlinks mentioned on the main page of a company website. We retain those links that belong to the company's domain and discard others. We apply this recursion to each of these links for 5 times. To crawl HTML pages from a set of URLs collected from a company website, we first exclude URLs that contain any word from a clearance list. This list is defined by a product scouting expert, and relies on path segments (e.g., blog, downloads, and archive). Using this technique, we ensure that we do not crawl HTML pages that are clearly non-product pages. This module outputs a set of URLs and their corresponding HTML codes.

3.2 Classification

One of the main goals of the proposed method is to be computationally cost and time efficient. Since extracting product information from all crawled web pages is resource and cost inefficient, we introduce a classification module to first identify product web pages. As shown in Figure 2, we introduce a sequential classification module based on three types of information: (1) URL path segments, (2) URLs, and (3) HTMLs. The main motivation is that classifying a web page using each information type is less computationally expensive than the subsequent one.

URL path segment classifier. Given a URL, we extract its path segments by splitting it using "/" character. Then, if any of the URL's segments appear in our predefined whitelist, this page is labeled as a product page and given to the extraction module. This whitelist is curated by product scouting experts and contains tokens that may indicate a product page. If no URL segments match the whitelist, the web page is classified as a non-product page and passed to the URL classifier.

URL classifier. To prevent the classifier from becoming biased to company domains, we eliminate the domain segments from the URL. We filter out signs, numbers, and stemmed lower-cased words if their length is shorter than three characters. We then apply TF-IDF to these pre-processed URLs before passing them to the classification model. As with the URL path segment classification, we give the corresponding HTML to the extraction module if the model determines that a URL refers to a product page. Otherwise, the web page is given to the HTML classifier.

HTML classifier. In contrast to previous classification steps, this classifier deals with the HTML code of a web page. For pre-processing, we remove the content within tags such as script, style, and link because such content addresses the presentation style of an HTML page and is not relevant for classification. As the final step of the classification module, if a web page is identified as a product page, it is passed to the extraction module. Otherwise, it is a non-product page and discarded.

Note that the order of the classifiers is chosen for runtime efficiency: when the first product classifier predicts a product page, the subsequent classifiers are skipped, and thus, the more runtime-efficient models are applied first.

3.3 Extraction

Product name and description are the most essential information required to represent a product. Therefore, this module extracts these two pieces of information from a product page.

Although computationally expensive, LLMs have proven effective for information extraction (Brinkmann et al., 2023b; Xu et al., 2024)[1]. Since we already narrowed down web pages to only product pages, we can reduce the cost by applying these more expensive models to fewer web pages. To

[1]Although rule-based information extraction was predominantly used in industrial settings (Chiticariu et al., 2013), the emergence of LLMs has led to significant improvements in many aspects of ML-based components. One notable improvement is the increased flexibility in adapting a model. Given the challenge posed by the highly diverse input texts in our task, LLMs are the most suitable choice for the extraction module.

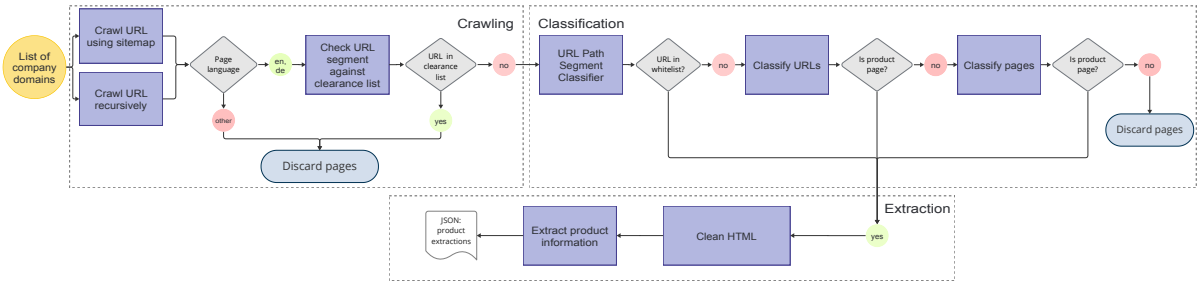


Figure 2: Component-level view: A breakdown of our method's modular design.

do so, we use the HTML code of a product page as context and then prompt pre-trained LLMs to extract the product name and description from the context.

We define a context limit for LLM prompts. This limit is calculated as the prompt length in terms of number of tokens minus a fixed number of tokens for the output. To fit HTML to this context limit, we shrink HTML codes by applying the following HTML cleaning techniques. The cleaning dynamically adjusts the HTML size to accommodate the context limit. In particular, we remove JavaScript codes, CSS styles, comments, hyperlinks, unknown tags, et sim., to only retain crucial content of a product page. If the HTML code still remains longer than the context length, we transform it to Markdown, which is known to be a lighter markup language. The process stops as soon as the context limit is met. If the HTML code and Markdown are still too long, we apply a hard cutoff.

4 Datasets

We collect and annotate two sets of web pages associated with different company websites. Table 1 provides major statistics about the datasets. We use one set for training and the other one for test purposes. In this way, we ensure that company websites used during evaluation have not been used for training our models.

4.1 Training Set

We crawl websites of 83 companies using our sitemap-based crawler (see Section 3), resulting in 301,785 web pages. We randomly select 30,077 web pages and instruct one product scouting expert to label each page with either 0 or 1, where 1 indicates a product page. The expert annotates a page as a product page if it presents any type of information about a product. We obtain 4,513 product pages and 25,564 non-product pages. Since annotating all product pages with the product name and description requires a lot of time from the expert, we let the expert annotate only 558 product pages. To be even more time-efficient, we use GPT-3.5 to suggest product names and descriptions to the expert. Then, the expert corrects the mistakes that GPT-3.5 made with their annotations.

Property	Train	Test	Controlled Test
# Companies	83	75	75
# Crawled pages	301,785	45,622	45,622
Product pages			
# Pages	4,513	456	309
(Min, Max) / company	(1, 1031)	(1, 62)	(1, 52)
Avg. / company	79.2	17.5	12.9
Avg. HTML size	57,790	45,895	51,858
Non-Product pages			
# Pages	25,564	6,619	6,766
(Min, Max) / company	(1, 5621)	(1, 2387)	(1, 2387)
Avg / company	323.6	108.5	110.9
Avg. HTML size	78,266	38,666	38,551

Table 1: Dataset statistics.

4.2 Test Set

We collect and annotate a set of web pages crawled from a new set of companies. In particular, we use both sitemap-based and recursive crawlers to scrape additional 75 company websites, resulting in 45,622 web pages. This set contains companies distinct from those present in the training set. We randomly select 7,075 web pages to annotate. From these web pages, we define two variant test examples.

Test. This set includes all selected web pages annotated by the same expert who annotated the training samples. We conduct the same annotation procedure used for the training set. As a result, each web page in this test set is accompanied by product page label, product name, and product description

annotations.

Controlled Test. While the test set has high coverage of web pages, it may be biased in favor of LLMs since the expert annotates the output of GPT-3.5. To study the effect of GPT-3.5, we select all product pages identified by the expert and re-annotate them with product page labels, product names, and descriptions from scratch. As shown in Table 1, the number of product pages in the controlled test is less than that of the test (309 vs 456). The reason behind this difference is that the expert annotates a web page as a product page if it contains any type of information about company products (e.g., web pages related to product catalogue and about us). On the other hand, we label a web page as a product page only if it includes detailed information about one product, which is more aligned with our research in this work.

5 Experiments

We evaluate the performance and cost efficiency of our method (see Section 3) by empirically addressing the following questions: **(Q1)** For the entire task, our method deals with both product and non-product pages together. How do the effects of classifying web pages and processing only product pages impact the quality and computational cost of product information extraction? **(Q2)** How effective is the sequential classification module in classifying web pages? **(Q3)** To what extent can a zero-shot LLM extract product information from a given product page?

5.1 Experimental Settings

In the pre-processing step, we utilize the Python modules *boilerpy* [2] and *lxml* [3] to remove uninformative HTML content, such as the page formatting information.[4] The *URL classifier* is a logistic regression model, trained using *scikit-learn*. For the *HTML classifier*, we fine-tune MarkupLM (Li et al., 2022) to identify product page HTML strings. Additional information about the experimental setup of the classifiers can be found in Appendix C.

For the extraction experiments, we use *Llama-3-8B-Instruct* with zero temperature deployed on *NVIDIA A100-SXM4-80GB MIGs*. We utilize vLLM (Kwon et al., 2023), which leverages the **PagedAttention** mechanism, resulting in up to 24x higher throughput compared to HuggingFace Transformers without requiring any model architecture modifications. For comparison, we also use *GPT-3.5-Turbo-1160* on Azure with 240k tokens per minute (TPM) limits and zero temperature. It is worth noting that we use our crawling method as a baseline to collect data for evaluating the other pipeline components. Developing more advanced crawling methods and their evaluation are not within the scope of this paper and left for future work. We leave more implementation details, such as the prompts we use to interact with these models, in Appendix A.

5.2 Results

We report the experimental results supporting our answers to the above questions.

Identifying product pages and extracting product information from them can significantly improve quality and reduce computational costs of the overall task. To evaluate our method on the entire task, it is essential to handle both product and non-product pages together. Table 2 reports the performance of our method (i.e., Cls. + Ext.) compared to using only Ext. for all web pages. Given a web page, we request the Ext. method (i.e., LLaMA) in one prompt to return three outputs: (1) a binary product page label, (2) product name, (3) and product description. If this method finds no product name or description, it returns "Not Found" accordingly. We observe that using our classification module to identify product pages and then feeding only these web pages to the Ext. model improves the performance remarkably. This is because the Ext. model may incorrectly identify non-product pages as product pages and then extract incorrect information from these pages as a product name and description. Although extensive prompt engineering may improve the performance of the Ext. model, it is a computationally expensive process. One significant advantage of our method is that the Cls. module substantially reduces the computational cost of the extraction component while its cost compared to the whole method is next-to-zero. Table 3 compares our method with two LLM baselines in terms of total execution time and expenses. For this experiment, we use 11,660 web pages as input, among which 583 samples are product pages. We measured the total processing time and total expense with 10, 50, 100, and

[2] https://github.com/jmriebold/BoilerPy3/

[3] https://lxml.de/

[4] Note that page formatting can indirectly convey some information about the content, e.g. prominence.

Model	Test BertScore	Test ROUGE	Controlled Test BertScore	Controlled Test ROUGE
Product Name				
Ext.	88.97	28.33	89.10	27.98
Cls. + Ext.	**95.33**	**93.90**	**95.39**	**94.54**
Product Description				
Ext.	85.97	26.26	86.13	25.94
Cls. + Ext.	**92.72**	**92.45**	**93.27**	**93.26**

Table 2: Task evaluation. Cls. is our classification module and Ext. is our extraction module using LLaMA. BertScore is weighted average F1-score. ROUGE is ROUGE-1.

	≈Time (Min)	≈ Expense (EUR)
GPT-3.5	133	35
Cls. + GPT-3.5	7	3
LLaMA	101	16
Cls. + LLaMA	**5**	**2**

Table 3: Total execution time and expenses of the examined methods.

500 parallel threads. We discuss the results of 100 threads here and report the rest in Appendix B. After classifying web pages using our classification module and feeding only product pages to the Ext. module (powered by GPT-3.5 or LLaMA), the total time and expense significantly diminish. We further observe that 15% of requests to GPT-3.5 were rejected with an error message *429*, indicating that we reach the TPM limit, suggesting to retry after 2 seconds. This highlights a significant limitation: using GPT-3.5 on Azure would not be scalable for processing large volumes of HTML pages with the given default subscription.

The classification module achieves a higher recall and F1 score compared to each individual classifier. We compare the performance of our classification module to each of its components to gain insight into its overall effectiveness. Table 4 shows the results. We use the classifier with URL path segment features as the baseline as it is a straightforward method to filter out non-product pages. The expert definition of these terms resulted in a high precision value and consequently low recall. The "URL" in Table 4 is a Logistic Regression model trained on TF-IDF feature representations of URLs. It shows the best recall and F1 scores among the three classifiers. However, the pages that this classifier identifies as non-product should be rechecked with HTML classifier to ensure we do not miss any product pages. "All" represents

Cls	Test P	Test R	Test F1	Controlled Test P	Controlled Test R	Controlled Test F1
Path Seg.	91.16	36.18	51.81	71.27	41.75	52.65
URL	62.42	62.28	62.35	48.35	71.20	57.59
HTML	61.61	57.02	59.23	49.53	67.64	57.18
All	55.85	**82.68**	**66.67**	42.96	**93.85**	**58.94**

Table 4: Classification module evaluation in terms of precision (P), recall (R) and F1-meaure (F1). All is the classification module used in our method.

the performance of our classification module where three components are sequentially connected (Figure 2). Our module outperforms the classification components, demonstrating its effectiveness for use in our entire method.

Prompting off-the-shelf LLMs is sufficient to extract product information. To study the impact of LLaMA as an off-the-shelf model for extracting product information in our method, we compare its performance with a fine-tuned BERT model as a baseline. As Table 5 shows, LLaMA, without any fine-tuning and in a zero-shot setting, outperforms the fine-tuned BERT extraction model, evaluated only on product pages. The extraction results show a slight improvement for the controlled set compared to the test set. This effect is also evident in the task evaluation (Table 2). The finding suggests that expert annotations are not biased towards the GPT-3.5 suggestions, since the models perform equally well on a manually annotated dataset.

For the BERT Ext. we fine-tune DistilBERT (Sanh et al., 2019). To harness the full content of web page, we extract text snippets from HTML of the page along with their corresponding nearest tags. With this, we can simplify the extraction task by formulating it as classification, where a model should predict labels "product name", "product description", and "other" for each text snippet. We label text snippets from all product pages in the training set, excluding 558 pages that are already annotated with product name and descriptions. In particular, we identify explicit information about the text content within HTML tags (e.g. *class= "product_name"*). Since this approach is more effective for product names than for descriptions due to the text length, we also identify descriptions as text that contains the identified product name and is longer than 100 characters. Due to the higher frequency of "others" labels compared to the other two labels, we randomly sample a maximum of five

Model	Test BertScore	Test ROUGE	Controlled Test BertScore	Controlled Test ROUGE
Product Name				
BERT Ext.	89.61	52.55	87.25	47.99
LLaMA Ext.	**91.98**	**62.07**	**92.77**	**71.58**
Product Description				
BERT Ext.	83.15	25.71	82.93	24.53
LLaMA Ext.	**87.62**	**37.63**	**88.05**	**39.79**

Table 5: Extraction module performance using only the product pages. BertScore is average F1-score. ROUGE is ROUGE-1.

cases per product page. We fine-tune DistilBERT on 1,368 product names, 388 product descriptions, and 1,326 others examples. We select the best performing model on 558 annotated web pages from training set. We report the performance of this best model on our test sets. During the prediction step, we retrieve the text snippets with the highest scores for product name and description, and then remove the HTML tags.

Overall, the results in Table 4 and Table 5 show the validity of the methods used in our classification and extraction modules.

In another small-scale validation experiment, we want to explore the performance of more standard linguistic tools, i.e., named entity recognizers (NER, Keraghel et al., 2024). The task of NER is closely related to our task of extracting the product name. However, a crucial difference is that we do not operate on the plain text but on the HTML, and that we aim for the main product name. Thus, work like GPT-NER (Wang et al., 2023), that bridges the gap between LLMs and classic sequence labeling, is close to our work, although we have HTML tags as indirect string markers. For those reasons, the experiment can only be applied to product names, not to descriptions, and with some further modifications. Two named entity recognizers provide the entity type *PRODUCT* by default: Stanza NER (Stanford NLP) (Qi et al., 2020) which is trained on the OntoNotes corpus [5], and SpaCy NER [6]. In both cases, the entity type *PRODUCT* is only available for English. Thus, the test dataset is restricted to English pages in a preprocessing step. Further, HTML tags are removed and the plain text is taken as input to the models. A difference to the previous extraction modules is that the tools extract all product names, rather than the main one. In the evaluation, all predicted product names are taken into consideration. The results are given in Table 6. The results show decent performance for BertScore, but very low ROUGE scores.

[5] https://stanfordnlp.github.io/stanza/ner_models.html

[6] https://spacy.io/usage/linguistic-features/#named-entities

Model	Test BertScore	Test ROUGE
Product Name		
Spacy NER	72.00	04.44
Stanza NER	75.89	13.51

Table 6: Extraction performance for product names on English product pages using NER. BertScore is average F1-score. ROUGE is ROUGE-1.

6 Conclusions

We introduced a full pipeline to efficiently extract product information from web pages on a company website. This approach is in contrast to previous work where any type of web pages (product vs non-product pages) is fed into extraction models, which is inefficient and costly.

Our method consists of three modules: web page crawling, product page classification, and product information extraction. By introducing a classification module that effectively filters out non-product pages, we achieve cost-efficiency and reduce computational overhead. The classification module improves the qualitative performance of extraction as well. The reason behind this is that the classifier is more effective in filtering out non-product pages compared to the examined pre-trained LLM.

While being effective, our approach still has two limitations. The method is not optimized for extracting several products per page. Also, there are no processes to recognize and merge products that are mentioned several times on different pages. The former needs additional prompting whereas the latter can be addressed by duplicate detection methods.

In future, building on the promising initial results, we plan to extend our method to extract technical details from product pages, as LLMs have often been capable of generating such information as structured output. In addition, regular retraining of classifiers and performance monitoring is needed to keep the high quality of the overall method.

Ethics statement

We acknowledge the importance of responsible data collection and adhere to the high standards of ethics in our data acquisition process. Specifically, when obtaining product information from company websites, we ensure that our web crawling methods are transparent, respectful, and compliant with relevant laws and regulations. We adhere to the terms outlined in each website's *robots.txt* file and implement rate limiting to prevent any undue burden on website servers. Furthermore, during this process, we take measures to respect the intellectual property rights of content owners and do not collect or store any personal information, such as contact details. Our commitment to responsible data collection is guided by the principles of fairness, transparency, and respect for privacy.

Acknowledgements

We would like to thank Manuel Fischer, Erik Mackeprang, Marc Eichenauer, Alexander Brandt, Lars Siemon, Petar Radosavljevic, and Dennis Motzke for their invaluable contributions to our work. Their expert guidance, insightful feedback, platform support, and unwavering support have been instrumental in shaping the quality and direction of our work.

We also want to note that the first five authors have made equal and major contributions to this work, with the remaining authors providing significant contributions as well.

References

Ansel Blume, Nasser Zalmout, Heng Ji, and Xian Li. 2023. Generative models for product attribute extraction. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 575–585, Singapore. Association for Computational Linguistics.

Alexander Brinkmann, Roee Shraga, and Christian Bizer. 2023a. Product attribute value extraction using large language models. *ArXiv*, abs/2310.12537.

Alexander Brinkmann, Roee Shraga, Reng Chiz Der, and Christian Bizer. 2023b. Product information extraction using chatgpt. *ArXiv*, abs/2306.14921.

Laura Chiticariu, Yunyao Li, and Frederick R. Reiss. 2013. Rule-based information extraction is dead! long live rule-based information extraction systems! In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 827–832, Seattle, Washington, USA. Association for Computational Linguistics.

Shumin Deng, Chengming Wang, Zhoubo Li, Ningyu Zhang, Zelin Dai, Hehong Chen, Feiyu Xiong, Ming Yan, Qiang Chen, Mosha Chen, Jiaoyan Chen, Jeff Z. Pan, Bryan Hooi, and Huajun Chen. 2023. Construction and applications of billion-scale pre-trained multimodal business knowledge graph. *Preprint*, arXiv:2209.15214.

Yifan Ding, Yan Liang, Nasser Zalmout, Xian Li, Christan Grant, and Tim Weninger. 2022. Ask-and-verify: Span candidate generation and verification for attribute value extraction. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 110–110, Abu Dhabi, UAE. Association for Computational Linguistics.

Jiaying Gong and Hoda Eldardiry. 2024. Multi-label zero-shot product attribute-value extraction. *Preprint*, arXiv:2402.08802.

Vishrawas Gopalakrishnan, Suresh Parthasarathy Iyengar, Amit Madaan, Rajeev Rastogi, and Srinivasan Sengamedu. 2012. Matching product titles using web-based enrichment. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, CIKM '12, page 605–614, New York, NY, USA. Association for Computing Machinery.

Yulong Hui, Yao Lu, and Huanchen Zhang. 2024. Uda: A benchmark suite for retrieval augmented generation in real-world document analysis. *Preprint*, arXiv:2406.15187.

Imed Keraghel, Stanislas Morbieu, and Mohamed Nadif. 2024. A survey on recent advances in named entity recognition. *arXiv preprint arXiv:2401.10825*.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*.

Junlong Li, Yiheng Xu, Lei Cui, and Furu Wei. 2022. MarkupLM: Pre-training of text and markup language for visually rich document understanding. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6078–6087, Dublin, Ireland. Association for Computational Linguistics.

Yu-Chen Lin, Si-An Chen, Jie-Jyun Liu, and Chih-Jen Lin. 2023. Linear classifier: An often-forgotten baseline for text classification. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 1876–1888, Toronto, Canada. Association for Computational Linguistics.

Varun Malik, Ruchi Mittal, and Shubhranshu Vikram Singh. 2022. Epr-ml: E-commerce product recommendation using nlp and machine learning algorithm. *2022 5th International Conference on Contemporary Computing and Informatics (IC3I)*, pages 1778–1783.

Peng Qi, Yuhao Zhang, Yuhui Zhang, Jason Bolton, and Christopher D. Manning. 2020. Stanza: A Python

natural language processing toolkit for many human languages. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*.

Kalyani Roy, Pawan Goyal, and Manish Pandey. 2021. Attribute value generation from product title using language models. In *Proceedings of the 4th Workshop on e-Commerce and NLP*, pages 13–17, Online. Association for Computational Linguistics.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *ArXiv*, abs/1910.01108.

Qifan Wang, Li Yang, Bhargav Kanagal, Sumit Sanghai, D. Sivakumar, Bin Shu, Zac Yu, and Jon Elsas. 2020. Learning to extract attribute value from product via question answering: A multi-task approach. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, page 47–55, New York, NY, USA. Association for Computing Machinery.

Shuhe Wang, Xiaofei Sun, Xiaoya Li, Rongbin Ouyang, Fei Wu, Tianwei Zhang, Jiwei Li, and Guoyin Wang. 2023. Gpt-ner: Named entity recognition via large language models. *Preprint*, arXiv:2304.10428.

Bifan Wei, Jun Liu, Qinghua Zheng, Wei Zhang, Xiaoyu Fu, and Boqin Feng. 2013. A survey of faceted search. *J. Web Eng.*, 12:41–64.

Derong Xu, Wei Chen, Wenjun Peng, Chao Zhang, Tong Xu, Xiangyu Zhao, Xian Wu, Yefeng Zheng, Yang Wang, and Enhong Chen. 2024. Large language models for generative information extraction: A survey. *Preprint*, arXiv:2312.17617.

Jun Yan, Nasser Zalmout, Yan Liang, Christan Grant, Xiang Ren, and Xin Luna Dong. 2021. AdaTag: Multi-attribute value extraction from product profiles with adaptive decoding. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4694–4705, Online. Association for Computational Linguistics.

Nasser Zalmout, Chenwei Zhang, Xian Li, Yan Liang, and Xin Dong. 2021. All you need to know to build a product knowledge graph. *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*.

Guineng Zheng, Subhabrata Mukherjee, Xin Luna Dong, and Feifei Li. 2018. Opentag: Open attribute value extraction from product profiles. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18, page 1049–1058, New York, NY, USA. Association for Computing Machinery.

Henry Zou, Gavin Yu, Ziwei Fan, Dan Bu, Han Liu, Peng Dai, Dongmei Jia, and Cornelia Caragea. 2024. EIVEN: Efficient implicit attribute value extraction using multimodal LLM. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 6: Industry Track)*, pages 453–463, Mexico City, Mexico. Association for Computational Linguistics.

Appendices

A Additional details of the LLM extraction setup

For the LLM-based product information extraction, we use the following prompt template for both *GPT-3.5-Turbo-1160* and *Llama-3-8B-Instruct*:

```
You are an AI Assistant for market research called Product Extractor. Your primary
responsibility is to parse unstructured text such as HTML or Markdown and extract
structured information from it. Ensure that the output of your responses is consistently
formatted in JSON and free of invalid escape characters.

Provide a confidence score on a scale of 0.0 to 1.0, where 0.0 indicates uncertainty, 0.5
suggests moderate certainty, and 1.0 denotes full certainty.

If any information is missing, use the phrase "not found" and provide a certainty score on a
scale of 0.0 to 1.0.

{format_instructions}

## Input
{input}

Answer only in the requested format.
```

And these are the format_instructions:

```
The output should be formatted as a JSON instance that conforms to the JSON schema
below.
As an example, for the schema
{
    "properties": {
        "foo": {
            "title": "Foo",
            "description": "a list of strings", "type": "array",
            "items": {
                "type": "string"
            }
        }
    },
    "required": ["foo"]
}
the object {"foo": ["bar", "baz"]} is a well−formatted instance of the schema.
The object {"properties": {"foo": ["bar", "baz"]}} is not well−formatted.

Here is the output schema:
```
{
    "properties": {
        "product_name": {
            "title": "Product Name",
            "description": "name of the product",
            "type": "string"
        },
        "product_description": {
            "title": "Product Description",
            "description": "full product description",
            "type": "string"
        },
        "product_name_confidence": {
            "title": "Product Name Confidence",
            "description": "product name confidence", "example": "0.8",
            "type": "number"
        },
        "product_description_confidence":{
            "title": "Product Description Confidence",
            "description": "product description confidence", "example": "0.9",
            "type": "number"
        }
    },
    "required": ["product_name", "product_description", "product_name_confidence", "
            product_description_confidence"]
}
```
```

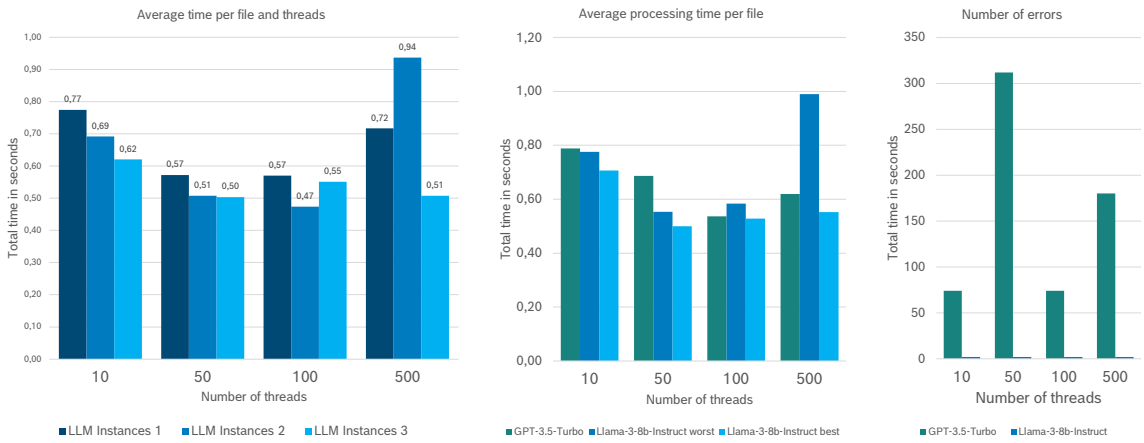
B LLM scaling experiments

As part of our scaling experiments, we aimed to assess the extraction processing time of deployed Llama-3-8B-Instruct models on our infrastructure. We conducted experiments with *1*, *2*, and *3* instances of *Llama-3-8B-Instruct* deployed on *1*, *2*, or *3* NVIDIA A100-SXM4-80GB MIGs. For model serving, we utilized vLLM (Kwon et al., 2023), which leverages the **PagedAttention** algorithm resulting in up to 24x higher throughput compared to HuggingFace Transformers, without requiring any model architecture modifications. For this experiment, we use 583 web pages as input containing product information. We measured the total processing time, processing time per file, and average processing time per file with 10, 50, 100, and 500 parallel threads, as illustrated in Figure 3 a).

We bench-marked the execution times of *GPT-3.5-Turbo-1160* on Azure and compared them to those of *Llama-3-8B-Instruct*. The results are presented in 3 b) and 3 c), with the latter showing the number of failed requests for GPT-3.5-Turbo versus Llama-3-8B-Instruct. Notably, when calling GPT-3.5-Turbo, we encountered an error code 429, indicating that we had exceeded the token rate limit of our current OpenAI S0 pricing tier which is 240K TPM. The error message suggested retrying after 2 seconds which highlights a significant limitation: using GPT-3.5-Turbo on Azure would not be scalable for processing large volumes of HTML pages, as we would repeatedly hit the TPM limit.

C Additional details of the classification setup

Having a look at the results of the *URL classifier* in Table 7, the final dataset for training contains 10,935 samples and it takes ≈0.2 seconds to train the model with scikit-learn on a standard local machine. In the context of the *URL classifier*, the train dataset is even extended with an additional set of 308 product pages from a previous scouting project of other experts, increasing the number of product pages to a total of 4,821 samples. The F1 score reached a value of 97.9%, with a precision of 100% and a recall of 95.8%. The high values in comparison to the test set results can be attributed to our experimental approach. After conducting fine-tuning experiments, we adopted an iterative selection process, by adding false positives and false negatives from the hold-out split, which were identified after training on a majority of product pages and a subset of non-product pages. No additional fine-tuning experiments were performed, but the default parameter set was chosen. The procedure of selecting false positives and false negatives was repeated several times, always training from scratch and including all data from the training dataset. In



(a) (b) (c)

Figure 3: (a) Average extraction time per HTML file and number of running threads using LLama-3-8b-Instruct. (b) Comparison of average extraction times between *GPT-3.5-Turbo-1160* and *LLama-3-8B-Instruct*.
(c) Number of failed requests of *GPT-3.5-Turbo-1160* and *LLama-3-8B-Instruct*.

terms of model interpretability, the first 30 term coefficients of the final logistic regression model can be seen in Figure 4.

Regarding the *HTML classifier*, it is not trivial to evaluate the similarity of HTML files in order to come up with an empirically well representative and diverse training dataset. A naïve approach is to train and evaluate the model in a cross-validation setup and investigate the outcome in terms of outliers. Therefore, the MarkupLM was initially evaluated applying 5-fold cross-validation with a relatively balanced subset of the training set. Furthermore, additional experiments for hyperparameter tuning resulted in 1,000 training iterations with a batch size of 18 and a dropout rate of 0.5. The optimizers Adam, SGD, and AdamW showed a similar performance, so AdamW was chosen with a learning rate of 1×10^{-5} and a weight decay of 1×10^{-4}. Other parameter selections have not resulted in convergence. Later on, this setup was extended with more non-product pages, constantly added to each split. A total number of 134 experiments has been logged with MLflow although not all of these experiments resulted in successful runs.

The final setup includes ≈83% of the product pages for training and the rest for the hold-out split. Considering the non-product pages and the imbalanced classes in general, just ≈17% of them were added to the training part and ≈10% to the hold-out split. The remaining test set, solely consisting of non-product pages, was used for monitoring the negative F1 score while optimizing towards the dev set, thus not influencing the learning process. It is worth noting that the constantly added prediction time of the comparably large test set significantly increases the runtime of the training process, which is ≈6.5h on a g4dn.4xlarge (GPU) instance from AWS.

Due to its higher recall in comparison to the other folds, the model of the 3rd split was chosen to be deployed for using it as part of the pipeline classification module. Its F1 score reached a value of 91.1% with 86.2% precision and a recall of 96.6% over the complete training cycle, as shown in in Table 7. The development split is imbalanced, comprising 754 positive and 2,693 negative samples. Consequently, we would expect a higher precision than 86.2% on a balanced hold-out set. This expectation is supported by the model's performance on the test dataset, with larger sample size of 18,451. Here, the negative recall is 96.8%, closely mirroring the 95.7% negative recall observed in the dev split. The implementation of early stopping after 50 iterations, i.e. continued training of the current model solely if the F1 score on the hold-out split increased or switching back to a previous model, has led to the final model selection at iteration 700 of 1,000.

Given the curated tokens of product scouting experts, the *URL path segment classifier* can be taken as the baseline for the overall experimental setup of the classification components in Table 7. The *URL classifier* outperforms this whitelist-based approach. While the *HTML classifier* outperforms

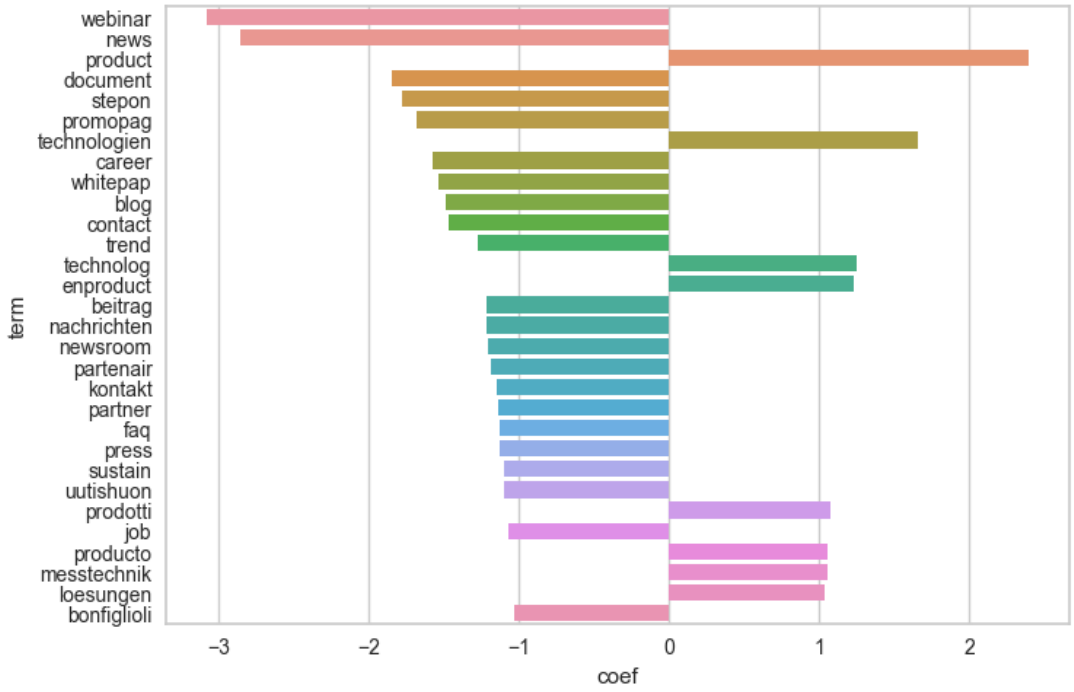


Figure 4: First 30 term coefficients of logistic regression model.

	PS all	URL test	PS test	HTML dev	PS dev	Base dev	HTML test	PS test	Base test
# pages	-	10,935	-	8,047	-	8,047	8,047	-	8,047
Acc.	0.8299	**0.9999**	0.8744	0.9585	0.8123	**0.9799**	0.9682	0.8618	**0.9818**
F1 pos.	0.5309	**0.9787**	0.0049	0.9105	0.6023	**0.9579**	-	-	-
P pos.	0.4528	**1.0**	0.0025	0,8615	0,5606	**0.9339**	0.0	0.0	0.0
R pos.	0.6414	**0.9583**	0.2500	0.9655	0.6507	**0.9832**	-	-	-
F1 neg.	0,8961	**0,9999**	0,9329	0,9729	0,8771	**0.9868**	0,9838	0,9258	**0.9908**
P neg.	0,9316	**0,9999**	0,9989	0,9900	0,8977	**0.9948**	1.0	1.0	1.0
R neg.	0,8631	**1.0**	0,8751	0,9565	0,8574	**0.9789**	0,9682	0,8618	**0.9818**

Table 7: Evaluation metrics for the different classifiers and the given amount of pages used for training (HTML test set solely consists of non-product pages). Abbreviations: PS - URL path segment classifier, URL - URL classifier, HTML - HTML classifier, Base - LightGBM HTML classifier, Acc. - accuracy, F1 - F1 score, P - precision, R - recall, pos. - positives, neg. - negatives.

Model	Accuracy	F1 Score	Precision	Recall	AUC
Light Gradient Boosting Machine	**0.9778**	**0.9758**	**0.9738**	**0.9779**	**0.9970**
Extreme Gradient Boosting	0.9771	0.9750	0.9723	**0.9779**	0.9965
Extra Trees Classifier	0.9739	0.9715	0.9691	0.9740	0.9955
Random Forest Classifier	0.9721	0.9696	0.9668	0.9724	0.9961
Gradient Boosting Classifier	0.9693	0.9663	0.9700	0.9627	0.9943
Decision Tree Classifier	0.9616	0.9583	0.9533	0.9635	0.9618
Ada Boost Classifier	0.9570	0.9529	0.9558	0.9503	0.9901
K Neighbors Classifier	0.9542	0.9498	0.9517	0.9480	0.9840
Ridge Classifier	0.9375	0.9321	0.9248	0.9398	0.0
SVM - Linear Classifier	0.9370	0.9317	0.9230	0.9410	0.0

Table 8: 10-fold cross-validation results of the 10 best performing baseline models for HTML classification.

(a) Company 1 product page (b) Company 2 product page (c) Company 3 product page

Figure 5: Product pages from different companies

the *URL path segment classifier* as well, the more appropriate baseline here is to compare the deep learning approach with a collection of models dealing with TF-IDF input (Lin et al., 2023). This approach is similar to the one chosen for URL classification, but due to the size and diversity of the given HTML pages, it needs to be limited to the top 10K features of all training documents. Furthermore, it makes use of the same cleaning functionality as applied within the MarkupLM setup and it excludes English stop words. With an accuracy of ≈98% on the dev and the test hold-out split, the LightGBM classifier shows better results than the deep learning model in Table 7. In addition, Table 8 lists the training evaluation metrics of the LightGBM classifier and comparably good models. However, when applying the LightGBM model to page content of unknown companies, the performance drops drastically in comparison to the MarkupLM results in Table 4. In case of the test set, the F1 score reaches a value of ≈24%, with a precision of 40.0% and a recall of 17.1%. In comparison, the precision decreases to 32.8% on the controlled test set, while the recall increases to 20.7%, resulting in an F1 score of 25.4%. This drop of performance is reasonable for the given amount of data and the limited feature space, because the complexity of an arbitrary company web page of type product or non-product cannot easily be generalized by a TF-IDF-based approach.

D Product pages

Our method is designed to handle product pages from various companies, each with their unique HTML structure and format. In contrast, product catalog pages (e.g., Amazon or Shopify) are not the focus of this work, as they typically have a standardized structure and can be easily parsed using tools like Beautiful Soup and regular expressions. Figure 5 illustrates the diversity of HTML pages from three different companies, with the original text replaced by dummy Lorem ipsum code for demonstration purposes. To view the actual product pages, click on the link below each image. This highlights the complexity of the pages we encounter, many of which resemble ordinary blog posts or about us pages. Furthermore, it underscores the importance of having a product page classifier component prior to the extraction component to ensure accurate processing.