AutoScraper: A Progressive Understanding Web Agent for Web Scraper Generation

Anonymous ACL submission

Abstract

Web scraping is a powerful technique that extracts data from websites, enabling automated data collection, enhancing data analysis capabilities, and minimizing manual data entry efforts. Existing methods, wrappers-based methods suffer from limited adaptability and scalability when faced with a new website, while language agents, empowered by large language models (LLMs), exhibit poor reusability in diverse web environments. In this work, we introduce the paradigm of generating web scrapers with LLMs and propose AutoScraper, a two-stage framework that can handle diverse and changing web environments more efficiently. AutoScraper leverages the hierarchical structure of HTML and similarity across different web pages for generating web scrapers. Besides, we propose a new executability metric for better measuring the performance of web scraper generation tasks. We conduct comprehensive experiments with multiple LLMs and demonstrate the effectiveness of our framework. Our work is now open-source.

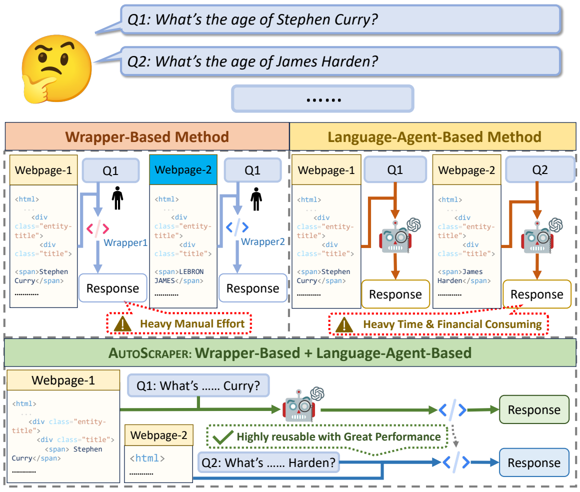

Figure 1: An illustration of comparing wrapper-based methods, language-agent-based methods and AutoScraper .

1 Introduction

Web scraping is a process where software automates the extraction of data from websites, typically using bots or web scrapers to gather specific information (Thapelo et al., 2021). It is important because it allows for efficient data collection and aggregation, which can be crucial for market research, competitive analysis, and real-time data monitoring.

Due to the diversity of sources and information on the internet, the construction of a web scraper requires substantial human effort. Consequently, two types of methods for automatic web information acquisition have been proposed, categorized as wrapper-based and language-agent-based (Sarkhel et al., 2023). The wrapper-based method entails complex sequences of operations within customized rule-based functions, which are designed to efficiently access and retrieve desired data from websites, which is especially beneficial for structured websites with stable layouts (Kushmerick, 1997; Dalvi et al., 2011; Bronzi et al., 2013). Conversely, the language-agent-based method leverages powerful natural language processing capabilities of large language models (LLMs) to interpret free-text queries and directly extract data within websites to meet the demand, effectively handling both structured and dynamic web content (Whitehouse et al., 2023; Marco Perini, 2024).

Although both types of methods facilitate web scraping to varying degrees, as shown in Figure 1, they exhibit significant shortcomings in terms of scalability. Wrapper-based method, while reusable, struggles with entirely new website structures, which necessitates extensive human effort to develop additional customized functions (Gulhane et al., 2011; Lockard et al., 2019). Conversely, although language-agent-based methods demonstrate superior performance in adapting to new content, their reliance on a limited number of superpowerful API-based LLMs for web scraping incurs considerable time and financial costs. Together, these challenges impede the broader adoption and scalability of current web scraping technologies, limiting their practicality in dynamic and diverse web environments.

To address the shortcomings of the aforementioned two paradigms, the paradigm of generating web scrapers with LLMs would be the optimal solution. On one hand, compared to wrapper-based methods, it fully leverages the reasoning and reflection capacities of LLMs, reducing manual design on new tasks and enhancing scalability. On the other hand, compared to language-agent-based methods, it introduces repeatable extraction procedures, reducing the dependency on LLMs when dealing with similar tasks, and thereby improving efficiency when handling a large number of web tasks. However, there are several challenges associated with using LLMs to generate web scrapers:

1. **Long HTML document.** Although LLMs excel in comprehending long textual content, HTML, as semi-structured data, comprises both structured (tags and attributes) and unstructured (textual content) elements. Consequently, it is challenging for LLMs to generate executable web scrapers that strictly adhere to the hierarchical structure of web pages in complex markup contexts.

2. **Reusability.** A good scraper needs to be reusable across multiple web pages. However, the differences in content and structure between various web pages can lead to the creation of a scraper that references a webpage, which can only be applied to some web pages.

3. **Appropriate evaluation metrics.** For a scraper to be considered useful, it must be able to automatically extract the desired results from all web pages. However, existing evaluation metrics for web information extraction, which focus on the extraction results from individual web pages, do not adequately reflect the usability of the scraper. This can potentially mislead experimental conclusions.

We introduce AUTOSCRAPER, a two-stage framework to address the web scraper generation task. Illustrated in Figure 2, AUTOSCRAPER comprises two main components: progressive generation and synthesis. The progressive generation stage leverages the hierarchical structure of HTML for progressive understanding to address the long HTML document. Subsequently, the synthesis module integrates multiple scrapers generated on different web pages to produce a cohesive, website-specific scraper that functions universally within that site. Besides, we propose a new evaluation metric for web scraper generation tasks, called the executability metric. Unlike traditional information extraction metrics that measure single web pages, this metric measures multiple web pages within a website, accurately reflecting the reliability and reusability of the scraper.

We evaluate AUTOSCRAPER on three available datasets with 7 LLMs. On all three datasets, AUTOSCRAPER consistently outperforms all baselines and achieves new state-of-the-art results in zero-shot settings. Also, AUTOSCRAPER can surpass supervised learning methods. Moreover, AUTOSCRAPER demonstrates superior efficiency on large-scale web information extraction task. Compared to traditional wrappers, AUTOSCRAPER adjusted more quickly according to different websites and task requirements. This flexibility enables scrappers to handle diverse and changing web environments more efficiently. Compared to the language agent paradigm, it introduces intermediate functions to enhance reusability and reduce the dependency on LLMs when dealing with similar tasks, thereby improving efficiency when handling a large number of web tasks.

2 Related Work

Wrapper-based methods for web scraping utilize the hierarchical structure of the webpage. Method of this category includes rule-based (Zheng et al., 2008), learning wrappers (i.e a DOM-specific parser that can extract content) (Gulhane et al., 2011; Kushmerick, 1997; Dalvi et al., 2011), heuristic algorithm (Lockard et al., 2018, 2019) and deep learning neural network (Lin et al., 2020; Zhou et al., 2021; Li et al., 2022; Wang et al., 2022). These methods demand substantial human involvement, including creating wrapper annotations, applying heuristic scoring rules (such as visual proximity), crafting features for neural network input, and using prior knowledge for verification. Therefore, it is difficult for wrapper-based methods to automatically scale up when facing web scraping tasks across a large number of different websites.

With the emergence of powerful LLMs (OpenAI, 2023; Touvron et al., 2023), language agent (Sumers et al., 2023) act in interactive environments with the help of LLM-based reasoning, grounding, learning, and decision making. Current language agents target the web mainly aim to streamline the web environment (Sridhar et al., 2023; Gur et al., 2023; Zheng et al., 2024) and to devise strategies for planning and interacting with the web (Sodhi et al., 2023; Ma et al., 2023). However, these frameworks mainly focus on the concept of the open-world web simulation environments (Shi et al., 2017; Yao et al., 2023; Deng et al., 2023; Zhou et al., 2023), encompassing a broad spectrum of tasks found in real-life scenarios, such as online shopping, flight booking, and software development. These task scenarios are oriented towards individuals, and there is a huge difference in the requirements for accuracy and efficiency compared to web scraping. Therefore, current language-agent-based methods, cannot effectively utilize the HTML structural similarities between multiple web pages, reducing the dependency on LLMs when performing repetitive operations, resulting in inefficiencies.

3 Preliminaries

In this section, we first define the scraper generation task and then present the dataset collection process and its corresponding evaluation metrics.

3.1 Task Formulation

First, we formulate our scraper generation task. Given a set of webpages on the same website $w \in \mathcal{W}$ describing a subject entity s (also called topic entity in the previous literature), and its corresponding predefined target attribute $r \in \mathcal{R}$, the task objective is to generate an executable rule/action sequence $\mathcal{A}$ to extract target information o from all webpages.

3.2 Datasets

We adopt the semi-structure information extraction task as a testbed for the scraper generation task.

SWDE (Hao et al., 2011) is a Structured Web Data Extraction dataset that contains webpages from 80 websites in 8 domains, with 124,291 webpages. Each of the websites from the same domains focuses on 3-5 attributes in the web pages.

EXTENDED SWDE (Lockard et al., 2019) involves fine-grained manual annotation of 21 sites in 3 domains from SWDE. While SWDE contains an average of 4,480 triples for 3 predicates per website, the EXTENDED SWDE dataset averages 41K triples for 36 predicates per site.

Dataset	$\mathbf{Num}_{\mathbf{Case}}$	$\mathbf{Num}_{\mathbf{Task}}$	$\mathbf{Num}_{\mathbf{Web}}$
SWDE	320	32	32,000
EXTENDED SWDE	294	221	29,400
DS1	83	11	186

Table 1: The statistic of web scraping task benchmarks. We report the number of the case ($\mathbf{Num}_{\mathbf{Case}}$), the number of the different extraction task ($\mathbf{Num}_{\mathbf{Task}}$) and the total number of webpages ($\mathbf{Num}_{\mathbf{Web}}$).

DS1 (Omari et al., 2017) contains 166 annotated webpages from 30 real-life large-scale websites categorized into books, shopping, hotels, and movies.

We transform the dataset with the following settings. First, we design instructions for each of the domains, and for each of the attributes as the input information for LLMs[1]. Second, for each website in each domain, we sample 100 web pages as the whole test set. We consider the set of webpages on the same websites and the corresponding extraction instruction as a case. For example, for the ESPN websites[2] in NBA player domains, the sampled 100 detail webpage of players and the instruction *Please extract the team of the player he plays now* is a complete case of our scraper generation task. Third, we pre-process the web pages by removing irrelevant elements in a webpage. We use open-source BeautifulSoup library[3] and filter out all DOM element nodes with `<script>` and `<style>`, as well as delete all attributes in the element node except `@class`. We replace the original escape characters in the annotations to ensure consistency with the corresponding information on the web. The statistic of the dataset we transformed is shown in Table 1.

3.3 Evaluation Metrics

Existing evaluation schemes for web page information extraction tasks still follow the traditional metrics of text information extraction tasks, namely precision, recall, and F1 score. They limit the assessment of methods for the scraper generation task to two aspects. First, it focuses on extraction with a single webpage, rather than considering the generalizability from the perspective of a collection

[1] Further details about the prompt is in Appendix D
[2] https://global.espn.com/nba/
[3] https://beautifulsoup.readthedocs.io

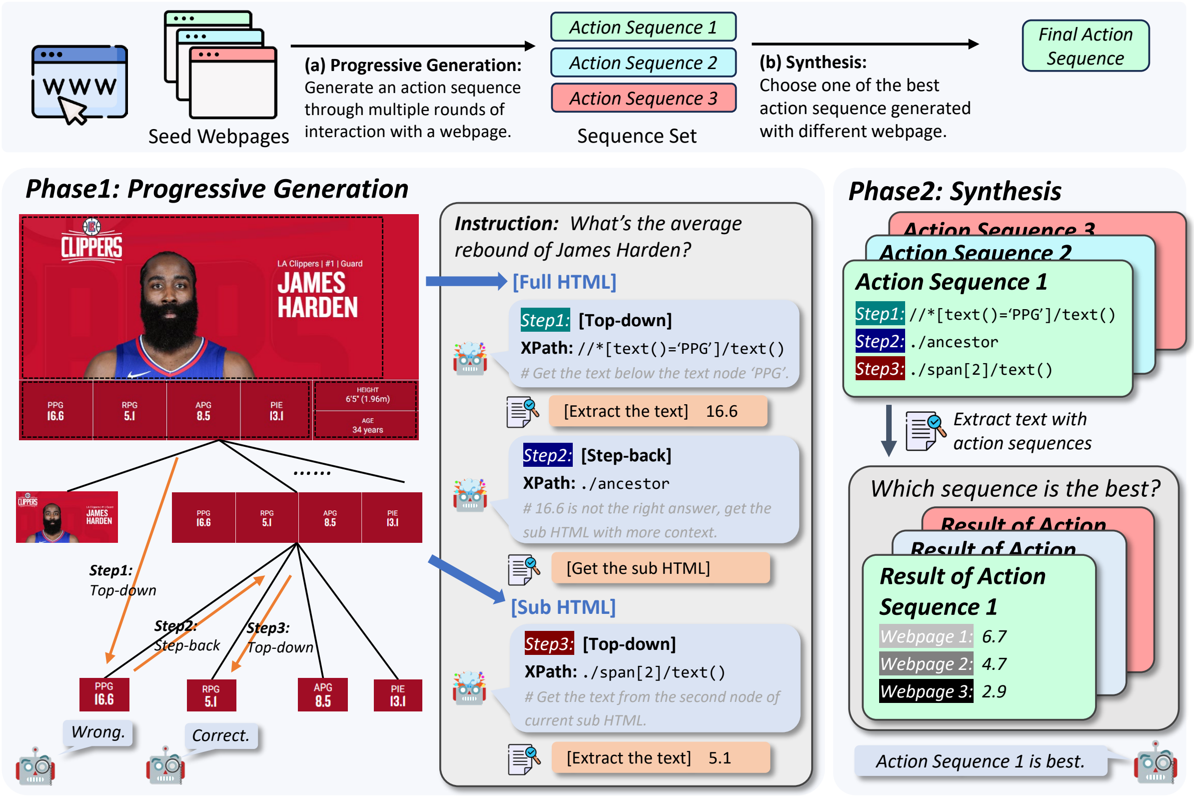

Figure 2: AUTOSCRAPER framework of two phases: **(a) progressive generation** and **(b) synthesis**.

of webpages. Second, it does not effectively measure the transferability when adopting the action sequence to other web pages.

To address this issue, we transform the traditional IE task evaluation into an executable evaluation. Based on the traditional IE evaluation on a collection of web pages, we categorize the executability of action sequences into the following six situations. Specifically, for each extraction task on a website, the result is classified based on the extraction result on precision, recall, and f1-score. (1) **Correct**: both precision, recall and f1-score equal 1, which indicates the action sequence is precisely; (2) **Precision(Prec.)**: only precision equals 1, which indicates perfect accuracy in the instances extracted following the action sequence, but misses relevant instances; (3) **Recall(Reca.)**: only recall equals 1, which means that it successfully identifies all relevant instances in the webpage but incorrectly identifies some irrelevant instances; (4) **Un-executable(Unex.)**: recall equals 0, which indicates that the action sequence fails to identify relevant instances; (5) **Over-estimate(Over.)**: precision equals 0, which indicates that the action sequence extracts the instances while ground truth is empty; (6) **Else**: the rest of the situation, including partially extracting the information, etc.

Since the above classifications are mutually exclusive, we use the ratio metric to calculate the proportion of each result in our task.

$$M_R = \frac{\#\text{ case of situation}}{\#\text{ total case}} \tag{1}$$

We are more concerned with success rate, so for the *Correct* metric, higher values indicate a better proportion of generated execution paths; whereas for the *Un-executable* metric, lower values are preferable.

4 AUTOSCRAPER

In this section, we describe our framework AUTOSCRAPER for generating a scraper to extract specific information from semi-structured HTML. Our approach is divided into two phases: first, we adopt a progressive generation module that utilizes the hierarchical structure of web pages; second, we employ a synthesis module based on results from multiple web pages. The overall framework is presented in Figure 2.

4.1 Modeling

Unlike the wrapper method that generates an XPath, we model the scraper generation task as an action sequence generation task. In specific, we generate an action sequence $\mathcal{A}_{seq}$ that consists of a sequence of XPath[4] expression from a set of seed webpages (i.e., a small portion of webpages in the test case for generating the sequence).

$$\mathcal{A}_{seq} = [\text{XPath}_1, \text{XPath}_2, ..., \text{XPath}_n] \quad (2)$$

where n denotes the length of the action sequence. We execute the XPath in the sequence using the parser in order. In the sequence, all XPath expressions except the last one are used for pruning the web page, and the last one is used for extracting the corresponding element value from the pruned web page.

4.2 Progressive Generation

Dealing with the lengthy content and hierarchical structure of webpages, generating a complete and executable scraper in one turn is difficult. However, the HTML content is organized in a DOM tree structure, which makes it possible to prune irrelevant page components and hence, limit the length and height of the DOM tree to improve the performance of LLM generation.

Specifically, we perform a traversal strategy consisting of **top-down** and **step-back** operations. **Top-down** refers to starting from the root node of the current DOM tree, progressively refining down to the specific node containing the target information. **Step-back** refers to reassessing and adjusting selection criteria by moving up the DOM tree to choose a more reliable and broadly applicable node as a foundation for more consistent and accurate XPath targeting. At each step, we first employ a top-down operation, guiding the LLMs to directly write out the XPath leading to the node containing the target information and to judge whether the value extracted with XPath is consistent with the value it recognizes. If execution fails, then adopt a step-back operation to retreat from the failed node, ensuring the web page includes the target information, which is driven by LLMs. The detail is shown in Algorithm 1.

4.3 Synthesis

Although we gain an executable action sequence within the progressive generation process, there are still differences in the specific location of the target information and the structure between different web pages. The action sequence may collect XPath with specific characteristics in a single HTML and lose generalizability. To enhance the reusability of the action sequence, we propose a synthesis phase.

Specifically, we randomly select n_s webpages from the case as seed webpages. Then, we generate an action sequence for each of them. Subsequently, we execute multiple different action sequences to extract information from the seed web pages, respectively. We collect all action sequences and their corresponding results and then choose one that can extract all the target information in the web pages as the final action sequence.

5 Experiment

Intending to put AUTOSCRAPER to practical use, we investigate the following research questions: 1) Can AUTOSCRAPER outperform the state-of-the-art scraper generation methods? 2) How does AUTOSCRAPER framework improve the performance of the scraper generation task? 3) Does AUTOSCRAPER meet the requirements for web scraping tasks, specifically being accurate and efficient?

5.1 Experimental Settings & Evaluation Metrics

We conduct our experiment on various LLMs including closed-source LLMs: **GPT-3.5-Turbo** (OpenAI, 2022), **Gemini Pro**(Team et al., 2023) and **GPT-4-Turbo** (OpenAI, 2023) as well as open-source LLMs: **Mistral-7B** (Jiang et al., 2023), **CodeLlama-34B** (Rozière et al., 2024), **Mixtral 8×7B** (Jiang et al., 2024) and **Deepseek-Coder-33B** (Guo et al., 2024). Furthermore, we apply different LLM-prompt-based web agents as our baselines, including **COT** (Wei et al., 2023) and **Reflexion** (Shinn et al., 2023) and **AUTOSCRAPER** to them. The comparison between them is discussed in Appendix B.1. Due to the limited-length context of LLMs, all experiments are conducted under zero-shot settings.

We test them on three datasets: SWDE (Hao et al., 2011), EXTEND SWDE (Lockard et al., 2019) and DS1 (Omari et al., 2017). The experimental result of the last two can be found in Appendix A.1 and A.2. We set the size of seed webpages $n_s = 3$ and max retry times $d_{max} = 5$.

In addition to the execution evaluation metrics described in Section 3.3, we also employ traditional evaluation metrics to more comprehensively assess the quality of different action sequences.

[4] https://en.wikipedia.org/wiki/XPath

Models	Method	Executable Evaluation						IE Evaluation		
		Correct(↑)	Prec	Reca	Unex.(↓)	Over.	Else	Prec	Reca	F1
Closed-source LLMs										
GPT-3.5-Turbo	COT	36.75	8.83	6.71	43.46	0.71	3.53	89.45	50.43	47.99
	Reflexion	46.29	11.66	2.83	37.10	0.71	1.41	**94.67**	55.85	55.10
	AutoScraper	**54.84**	11.83	8.96	**19.35**	1.08	3.94	85.85	**73.34**	**69.20**
Gemini Pro	COT	29.69	10.94	7.50	47.19	1.25	3.44	81.21	45.22	41.81
	Reflexion	33.12	6.56	4.06	52.50	0.63	3.12	**87.45**	42.75	40.88
	AutoScraper	**42.81**	11.87	4.69	**34.38**	1.25	5.00	85.70	**57.54**	**54.91**
GPT-4-Turbo	COT	61.88	12.50	7.19	14.37	0.94	3.12	87.75	79.90	76.95
	Reflexion	67.50	13.75	4.37	10.94	0.94	2.50	**93.28**	82.76	82.40
	AutoScraper	**71.56**	14.06	5.31	**4.06**	0.63	4.37	92.49	**89.13**	**88.69**
Open-source LLMs										
Mistral 7B	COT	**3.44**	0.31	0.63	**95.31**	0.00	0.63	94.23	**4.55**	**4.24**
	Reflexion	2.19	0.00	0.31	97.19	0.00	0.31	95.60	2.78	2.49
	AutoScraper	2.87	0.00	0.00	96.77	0.36	0.00	**98.57**	3.23	2.87
CodeLlama	COT	17.98	3.75	2.25	74.53	0.00	1.50	**79.75**	21.98	21.36
	Reflexion	18.08	4.80	2.95	73.06	0.00	1.11	78.96	23.26	22.44
	AutoScraper	**23.99**	8.12	1.48	**64.94**	0.00	1.48	78.59	**28.70**	**28.41**
Mixtral 8×7B	COT	28.75	8.13	4.37	57.81	0.31	0.63	**89.79**	38.23	37.26
	Reflexion	36.25	6.88	3.12	51.25	0.00	2.50	89.35	44.57	43.60
	AutoScraper	**46.88**	10.62	7.19	**30.31**	0.63	4.37	87.32	**62.71**	**59.75**
Deepseek-coder	COT	36.56	10.94	5.63	42.50	0.63	3.75	86.05	48.78	47.05
	Reflexion	37.19	11.25	4.06	44.69	1.25	1.56	**86.41**	48.28	47.08
	AutoScraper	**38.75**	11.25	5.31	**39.69**	0.63	4.37	84.91	**52.11**	**49.68**

Table 2: The executable evaluation and IE evaluation of LLMs with three frameworks in SWDE dataset. We examine 7 LLMs, including 3 closed-source LLMs and 4 open-source LLMs.

Specifically, we adopt precision (P.), recall (R.), and macro-f1 (F1), which are calculated as the mean of the corresponding metrics for each case.

5.2 Main Results on SWDE

Results in Table 2 show that: *1)* With AutoScraper generating action sequence, LLMs can achieve better performance. Compared to the COT and Reflexion baseline, our method performs a higher ratio of correct and a lower ratio of unexecutable. Also, it should be noted that Mixtral 8×7B + AutoScraper can outperform ChatGPT + Reflexion, indicating the superiority of AutoScraper in the generation of executable action sequences in the scraper generation task. *2)* Models with small parameter sizes have significant difficulties in understanding and writing executable paths, so they can be considered challenging to apply in this task. On the contrary, large-scale models demonstrate a more stable ability in instruction alignment, web structure comprehension, and reflection on execution results; *3)* Traditional IE evaluation metrics cannot well describe the success rate of our task. Especially for the precision metric, it fails to reveal the performance gap among different methods with different models. This is because the extraction metrics only evaluate the results that have been extracted, ignoring that unexecutable or empty extractions also greatly damage the executability.

5.3 Generate with Golden Label

To better illustrate the effectiveness of our framework in generating executable action sequences, we compare the performance of COT, Reflexion, and AutoScraper , while answering the instruction. By offering the same extraction targets, we can effectively detect the performance of different frameworks in generating action sequences.

Table 3 shows experimental results, from which we can have the following observations: *1)* Our proposed progressive understanding framework still effectively enhances the model's performance under this setting; *2)* LLMs still suffer in accurately understanding web page contents with semi-structured markup languages, which illustrate the performance gap between Table 2 and Table 3; *3)* Compared to closed-source LLMs, even provided with golden labels, Open-source LLMs are unable to achieve sustained performance improve-

Models	Method	Executable Evaluation					
		Correct(↑)	Prec	Reca	Unex.(↓)	Over.	Else
Closed-source LLMs							
GPT-3.5-Turbo	COT	41.70	12.92	7.38	35.42	0.74	1.85
	Reflexion	47.23	16.24	2.21	33.21	0.37	0.74
	AutoScraper	**56.89**	19.43	5.65	**13.43**	0.71	3.89
Gemini Pro	COT	33.44	9.38	9.06	44.69	0.94	2.50
	Reflexion	35.31	9.38	6.88	43.75	1.56	3.12
	AutoScraper	**45.31**	13.44	6.25	**30.31**	1.25	3.44
GPT-4-Turbo	COT	61.88	11.56	9.06	11.56	1.25	4.69
	Reflexion	71.25	7.19	4.69	14.37	0.94	1.56
	AutoScraper	**75.31**	10.94	4.37	**4.06**	0.63	4.69
Open-source LLMs							
Mistral 7B	COT	2.19	0.00	0.31	97.19	0.00	0.31
	Reflexion	2.19	0.00	0.00	**97.50**	0.31	0.00
	AutoScraper	2.19	0.00	0.00	97.19	0.31	0.31
CodeLlama	COT	21.40	6.27	2.21	66.79	0.74	2.58
	Reflexion	22.21	4.93	3.94	66.95	0.49	1.48
	AutoScraper	**26.20**	12.55	5.54	**53.51**	0.00	2.21
Mixtral 8×7B	COT	27.50	7.50	5.31	56.87	0.94	1.87
	Reflexion	34.69	8.13	5.31	49.06	0.63	2.19
	AutoScraper	**45.62**	11.56	5.94	**32.50**	1.25	3.12
Deepseek-coder	COT	35.00	18.75	5.31	36.25	0.63	4.06
	Reflexion	**38.75**	11.87	2.81	42.19	0.63	3.75
	AutoScraper	38.44	20.94	4.06	**31.56**	0.94	6.56

Table 3: The executable and IE evaluation with 7 LLMs on SWDE dataset with golden label.

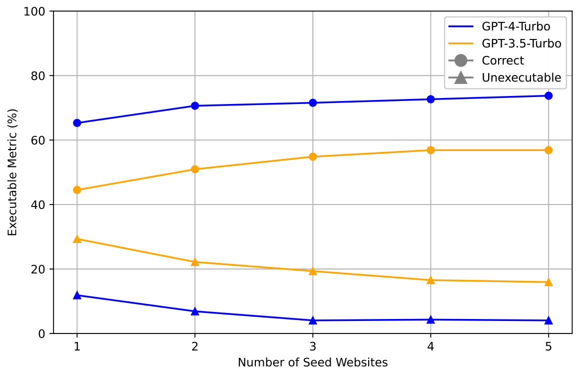

Figure 3: The performance of AutoScraper with different number of seed websites in SWDE dataset.

Models	Method	Exec Eval		IE Eval
		Correct(↑)	Unex.(↓)	F1
GPT-3.5-Turbo	COT	36.75	43.46	47.99
	- *synthesis*	27.56	57.24	34.44
	Reflexion	46.29	37.10	55.10
	- *synthesis*	28.62	59.01	35.01
	AutoScraper	54.84	19.35	69.20
	- *synthesis*	44.52	29.33	58.44
Gemini Pro	COT	29.69	47.19	41.81
	- *synthesis*	27.56	57.24	33.09
	Reflexion	33.12	52.50	40.88
	- *synthesis*	28.62	59.01	37.60
	AutoScraper	42.81	34.38	54.91
	- *synthesis*	39.46	31.56	56.48
GPT-4-Turbo	COT	61.88	14.37	76.95
	- *synthesis*	46.88	30.00	61.20
	Reflexion	67.50	10.94	82.40
	- *synthesis*	56.87	25.31	69.78
	AutoScraper	71.56	4.06	88.69
	- *synthesis*	65.31	11.87	80.41

Table 4: Ablation study on AutoScraper. We report **Correct**, **Unexecutable** from the executive evaluation, and **F1** score from the IE evaluation in SWDE dataset.

Model	F1
Render-Full (Hao et al., 2011)	84.30
FreeDOM (Lin et al., 2020)	82.32
SimpDOM (Zhou et al., 2021)	83.06
MarkupLM$_{\text{BASE}}$ (Li et al., 2022)	84.31
WebFormer (Wang et al., 2022)	92.46
Reflexion + GPT-4-Turbo	82.40
AutoScraper + GPT-4-Turbo	88.69

Table 5: Comparing the extraction performance (F1) of 5 baseline models to our method AutoScraper using GPT-4-Turbo on the SWDE dataset. Each value of the supervised model in the table is trained on 1 seed site.

ment. This phenomenon demonstrates that the bottleneck for these models lies not in understanding the webpage content but in understanding the webpage's hierarchical structure itself.

5.4 Ablation Study

To further justify the effectiveness of each component of AutoScraper, we perform an ablation study. The results are shown in Table 4. It shows that: 1) AutoScraper without a second module still beat the other two baseline methods among different LLMs. 2) The second module of AutoScraper, **synthesis** module, not only improves AutoScraper, but also improves the performance of other methods. Using more web pages for inference can make the generated scraper more stable and have better generalization.

5.5 Seed Websites

In all previous experiments, we fixed the number of seed websites $n_s = 3$, which demonstrates the effectiveness of the synthesis module. In this experiment, we offer different numbers of seed webpages and test the performance of AutoScraper. The result is shown in Figure 3.

As the number of seed webpages increases, the correct ratio increases, while the unexecutable ratio decreases. It suggests that the performance of AutoScraper can still be further improved by providing more seed webpages. In addition, the performance improvement reduces as the number increases, which shows that there is an upper limit to improve the performance of AutoScraper by increasing the number of seed webpages.

5.6 Comparison with supervised baselines

To further demonstrate that AutoScraper is adaptive to different web information extraction tasks, we conduct a comparison with 5 baseline models in web information extraction on supervised learning scenarios: Render-Full (Hao et al., 2011) proposes a complicated heuristic algorithm for computing visual distances between predicted value nodes and adjusting the predictions. FreeDOM (Lin et al., 2020) and SimpDOM (Zhou et al., 2021) encode textual features of DOM tree node with LSTM, while MarkupLM (Li et al., 2022) is pre-trained on HTML with text and markup information jointly. WebFormer (Wang et al., 2022) leverages the web layout for effective attention weight computation.

Table 5 shows the result. Although the comparison is unfair because our method is in zero-shot settings, AutoScraper beat most of them on F1 scores. It shows that by designing an appropriate framework, LLMs can surpass supervised learning methods in some web information extraction tasks.

5.7 Efficiency Analysis

Suppose the number of seed webpages is n_s, the number of webpages on the same website is $N_{\mathcal{W}}$, the time to generate a wrapper is T_g, the time of synthesis is T_s, and the time for extract information from a webpage with a wrapper is T_e. The total time for extracting all information from all websites with AutoScraper is

$$T_1 = T_G + T_E = (n_s T_g + T_s) + N_{\mathcal{W}} T_e \quad (3)$$

Besides, the time for LLMs directly extracting information from a webpage is T_d, and the total time for extracting all information from all websites directly is

$$T_2 = N_{\mathcal{W}} T_d \quad (4)$$

In a real-world scenario, there are many web pages from the same websites to be extracted. Although generating a wrapper takes more time than extracting directly from a single webpage, the extraction efficiency of subsequent web pages would be significantly improved. To explore how many webpages are needed to make AutoScraper more efficient in web IE, we calculate the threshold of $N_{\mathcal{W}}$. Suppose $T_1 \leq T_2$, we have

$$T_G + T_E = (n_s T_g + T_s) + N_{\mathcal{W}} T_e \leq N_{\mathcal{W}} T_d \quad (5)$$

$$N_{\mathcal{W}} \geq \frac{n_s T_g + T_s}{T_d - T_e} \quad (6)$$

It should be noted that T_g depends on d_{max} in Algorithm 1 and can be roughly considered as $T_g \approx d_{max} T_d$. In our experimental settings, we set $d_{max} = 5$ and $n_s = 3$. Also, under the approximation that $T_s \approx T_d$ and $T_d \gg T_e$, AutoScraper have better extraction efficiency when a website contains more than 16 webpages.

5.8 Error Analysis

We perform an analysis by looking at the recorded action sequence of AutoScraper with GPT-4-Turbo and identify the following common failure modes. We mainly focus on the case categorized as unexecutable, over-estimate, and else.

Non-generalizability of webpages The target information and corresponding webpage structures exhibit variations across different webpages, leading to a lack of generalizability in AutoScraper (i.e., the inability to apply the same rules across all webpages in the same website). For instance, for the task *"Please extract the name of the company offering the job"* in the website job-careerbuilder, most webpages contain the company name, but there is one webpage where the company name is *"Not Available"* on another node of DOM tree.

Miss in multi-valued Presented with the task of generating a scraper for extracting *address* in restaurant webpages or *contact phone number* from university websites, the target information is located in multiple locations in the webpage, such as the information bar, title, etc. Although AutoScraper is capable of generating action sequences to extract portions of information, crafting a comprehensive action sequence that captures all of the information remains a challenge.

6 Conclusion

In this paper, we introduce the scraper generation task and the paradigm that combines LLMs and scrapers to improve the reusability of the current language-agent-based framework. We then propose AutoScraper , a two-phase framework including progressive generation and synthesis module to generate a more stable and executable action sequence. Our comprehensive experiments demonstrate that AutoScraper can outperform the state-of-the-art baseline in the scraper generation task.

Limitation

We introduce a paradigm that combines LLMs with scrapers for web scraper generation tasks and propose AUTOSCRAPER to generate an executable action sequence with progressively understanding the HTML documents. Though experimental results show the effectiveness of our framework, there are still some limits to our work.

First, our framework is restricted to the paradigm in the information extraction task for vertical webpages. LLMs with scrapers provide high efficiency in open-world web IE tasks, but can hardly transfer to existing web environments such as Mind2Web (Deng et al., 2023), WebArena (Zhou et al., 2023).

Second, our framework rely on the performance of backbone LLMs. Enhancing LLMs' ability to understand HTML is a very valuable research question, including corpus collection and training strategy. We will conduct research on HTML understanding enchancement in future work.

Ethic statement

We hereby declare that all authors of this article are aware of and adhere to the provided ACL Code of Ethics and honor the code of conduct.

Use of Human Annotations Human annotations are only utilized in the early stages of methodological research to assess the feasibility of the proposed solution. All annotators have provided consent for the use of their data for research purposes. We guarantee the security of all annotators throughout the annotation process, and they are justly remunerated according to local standards. Human annotations are not employed during the evaluation of our method.

Risks The datasets used in the paper have been obtained from public sources and anonymized to protect against any offensive information. Though we have taken measures to do so, we cannot guarantee that the datasets do not contain any socially harmful or toxic language.

References

Mirko Bronzi, Valter Crescenzi, Paolo Merialdo, and Paolo Papotti. 2013. Extraction and integration of partially overlapping web sources. *Proc. VLDB Endow.*, 6(10):805–816.

Nilesh Dalvi, Ravi Kumar, and Mohamed Soliman. 2011. Automatic wrappers for large scale web extraction. *arXiv preprint arXiv:1103.2406*.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. 2023. Mind2web: Towards a generalist agent for the web.

Pankaj Gulhane, Amit Madaan, Rupesh Mehta, Jeyashankher Ramamirtham, Rajeev Rastogi, Sandeep Satpal, Srinivasan H Sengamedu, Ashwin Tengli, and Charu Tiwari. 2011. Web-scale information extraction with vertex. In *2011 IEEE 27th International Conference on Data Engineering*, pages 1209–1220.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Y. Wu, Y. K. Li, Fuli Luo, Yingfei Xiong, and Wenfeng Liang. 2024. Deepseek-coder: When the large language model meets programming – the rise of code intelligence.

Izzeddin Gur, Hiroki Furuta, Austin Huang, Mustafa Safdari, Yutaka Matsuo, Douglas Eck, and Aleksandra Faust. 2023. A real-world webagent with planning, long context understanding, and program synthesis.

Qiang Hao, Rui Cai, Yanwei Pang, and Lei Zhang. 2011. From one tree to a forest: a unified solution for structured web data extraction. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '11, page 775–784, New York, NY, USA. Association for Computing Machinery.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7b.

Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, Gianna Lengyel, Guillaume Bour, Guillaume Lample, Lélio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, Szymon Antoniak, Teven Le Scao, Théophile Gervet, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2024. Mixtral of experts.

Nicholas Kushmerick. 1997. *Wrapper induction for information extraction*. University of Washington.

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.

Bill Yuchen Lin, Ying Sheng, Nguyen Vo, and Sandeep Tata. 2020. Freedom: A transferable neural architecture for structured information extraction on web documents. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1092–1102.

Colin Lockard, Xin Luna Dong, Arash Einolghozati, and Prashant Shiralkar. 2018. Ceres: Distantly supervised relation extraction from the semi-structured web. *arXiv preprint arXiv:1804.04635*.

Colin Lockard, Prashant Shiralkar, and Xin Luna Dong. 2019. Openceres: When open information extraction meets the semi-structured web. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 3047–3056.

Kaixin Ma, Hongming Zhang, Hongwei Wang, Xiaoman Pan, and Dong Yu. 2023. Laser: Llm agent with state-space exploration for web navigation.

Marco Vinciguerra Marco Perini, Lorenzo Padoan. 2024. Scrapegraph-ai. A Python library for scraping leveraging large language models.

Adi Omari, Sharon Shoham, and Eran Yahav. 2017. Synthesis of forgiving data extractors. In *Proceedings of the tenth ACM international conference on web search and data mining*, pages 385–394.

OpenAI. 2022. Chatgpt.

OpenAI. 2023. Gpt-4 technical report.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. 2024. Code llama: Open foundation models for code.

Ritesh Sarkhel, Binxuan Huang, Colin Lockard, and Prashant Shiralkar. 2023. Self-training for label-efficient information extraction from semi-structured web-pages. *Proceedings of the VLDB Endowment*, 16(11):3098–3110.

Tianlin Shi, Andrej Karpathy, Linxi (Jim) Fan, Josefa Z. Hernández, and Percy Liang. 2017. World of bits: An open-domain platform for web-based agents. In *International Conference on Machine Learning*.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning.

Paloma Sodhi, S. R. K. Branavan, and Ryan McDonald. 2023. Heap: Hierarchical policies for web actions using llms.

Abishek Sridhar, Robert Lo, Frank F. Xu, Hao Zhu, and Shuyan Zhou. 2023. Hierarchical prompting assists large language model on web navigation.

Theodore R Sumers, Shunyu Yao, Karthik Narasimhan, and Thomas L Griffiths. 2023. Cognitive architectures for language agents. *arXiv preprint arXiv:2309.02427*.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M. Dai, and Anja Hauth. 2023. Gemini: A family of highly capable multimodal models.

Tsaone Swaabow Thapelo, Molaletsa Namoshe, Oduetse Matsebe, Tshiamo Motshegwa, and Mary-Jane Morongwa Bopape. 2021. Sasscal websapi: A web scraping application programming interface to support access to sasscal's weather data. *Data Science Journal*, 20:24–24.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Qifan Wang, Yi Fang, Anirudh Ravula, Fuli Feng, Xiaojun Quan, and Dongfang Liu. 2022. Webformer: The web-page transformer for structure information extraction. In *Proceedings of the ACM Web Conference 2022*, pages 3124–3133.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-thought prompting elicits reasoning in large language models.

Chenxi Whitehouse, Clara Vania, Alham Fikri Aji, Christos Christodoulopoulos, and Andrea Pierleoni. 2023. Webie: Faithful and robust information extraction on the web. *arXiv preprint arXiv:2305.14293*.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. 2023. Webshop: Towards scalable real-world web interaction with grounded language agents.

Longtao Zheng, Rundong Wang, Xinrun Wang, and Bo An. 2024. Synapse: Trajectory-as-exemplar prompting with memory for computer control.

Xiaolin Zheng, Tao Zhou, Zukun Yu, and Deren Chen. 2008. Url rule based focused crawler. In *2008 IEEE International Conference on e-Business Engineering*, pages 147–154. IEEE.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. 2023. Webarena: A realistic web environment for building autonomous agents.

Yichao Zhou, Ying Sheng, Nguyen Vo, Nick Edmonds, and Sandeep Tata. 2021. Simplified dom trees for transferable attribute extraction from the web. *arXiv preprint arXiv:2101.02415*.

A Experiments

A.1 Main results on EXTENDED SWDE

Because EXTENDED SWDE dataset focuses on *OpenIE* task (the relation is also expected to be extracted), we first map relations into a predefined list of attributes and remove unusual ones. Specifically, we conducted experiments with 294 attributes from 21 websites selected from the EXTENDED SWDE dataset.

Table 9 shows the result. By comparing Table 2, we find that: 1) Under complex extraction task settings (multiple target values and ambiguous problem description), the closed-source LLMs perform better in generating executable action sequences compared to the open-source LLMs. 2) There are some tasks with unclear descriptions, such as the *"Calendar System"* and *"Facilities and Programs Offered"* on university websites, which affect the wrapper generation performance of all methods.

A.2 Main results on DS1

Due to DS1 only contains 166 hand-crafted webpages, and for each website, there are only two webpages, so we take one webpage for inference and the other for evaluation. Meanwhile, due to the number of the seed websites being equal to one, we test three methods without applying the synthesis module described in Section 4.3.

Table 10 shows the result in the DS1 dataset. Among all LLMs with three methods, GPT-4-Turbo + AUTOSCRAPER achieves the best performance, and AUTOSCRAPER beats the other two methods in all LLMs, which is consistent with the conclusion we make above.

B Analysis on AUTOSCRAPER

B.1 Comparison with COT & Reflexion

Figure 4 more intuitively shows the specific differences between different baselines in the experiment. The most significant difference between AUTOSCRAPER and other methods lies in whether the hierarchical structure of web pages is utilized to help LLMs reduce the difficulty of complex web structures. COT only executes one turn while the other executes multiple turns and can learn from the failed execution of the wrapper. Compared to the Reflexion method, AUTOSCRAPER employs top-down and step-back operations to prune the DOM tree during each XPath generation process, thereby reducing the length of the web page. In

Algorithm 1: Algorithm for progressive understanding

Data: origin HTML code h_0, task instruction I, max retry times d_{max}
Result: Executable action sequence $\mathcal{A}_{seq}$ to extract the value in the HTML

Initial history $\mathcal{A}_{seq} \leftarrow [], k = 0$;
while *True* **do**
  if $k > d_{max}$ **then** break;
  // Top-down
  $value, xpath \leftarrow \textbf{LLM}_g(h_k, I)$;
  $result \leftarrow \textbf{Parser}_{text}(h_k, xpath)$;
  if *result* == *value* **then** break;
  // Step-back
  repeat
    $xpath \leftarrow xpath + "/.."$;
    $h_{k+1} \leftarrow \textbf{Parser}_{node}(h_k, xpath)$;
  until *h contains value*;
  Append$(\mathcal{A}_{seq}, xpath)$;
  $k \leftarrow k + 1$;
end
return $\mathcal{A}_{seq}$

Models	1	2	3	4	5	Avg.
GPT4	214	61	13	18	10	1.57
GPT-3.5-Turbo	115	65	22	30	43	2.35
Gemini Pro	94	52	33	27	105	2.99
Mixtral 8×7B	89	53	43	24	104	3.00
Mistral 7B	28	7	11	7	84	3.82
Deepseek-coder	137	70	55	29	23	2.14
CodeLlama	75	35	32	18	80	2.97

Table 6: Length of action sequence of AUTOSCRAPER based on different LLMs in SWDE dataset.

contrast, the Reflexion method can only reflect and regenerate after producing an unexecutable XPath, which does not effectively simplify the webpage.

B.2 Further Study with AUTOSCRAPER

The length of the action sequence is dependent on the LLMs capability. To comprehensively explore the performance of different LLMs in understanding web page structure, we explore the impact of models on the number distribution of the steps. In particular, we collect all the action sequences and calculate the average steps of AUTOSCRAPER with different LLMs. The experimental result is reported in Table 6.

We observe that AUTOSCRAPER with stronger LLMs generate fewer lengths of action sequence. AUTOSCRAPER with GPT-4-Turbo generates 1.57 steps on average, while the AUTOSCRAPER with Mistral 7B generates 3.82 steps on average. This phenomenon can be interpreted as more powerful models having a better understanding of the web page hierarchical structure, thus being able to accurately output the appropriate XPaths in longer/deeper web pages, thereby reducing the number of steps.

The "U" curve of compression ratio We define the length of HTML as the number of tokens in the HTML, and its height as the height of the DOM tree represented by the HTML. we define the compression ratio of length and height as the ratio of the length/height of the original web page to that of the web page after being pruned by AUTOSCRAPER .

$$Compression_L = \frac{\#\text{tokens of new HTML}}{\#\text{tokens of origin HTML}}$$
$$Compression_H = \frac{\#\text{height of new HTML}}{\#\text{height of origin HTML}} \quad (7)$$

We calculate their compression ratio of the **Correct** case and rank LLMs based on their performance. Figure 5 shows the result. It is interesting to note that there is a "U" curve on both the length and height compression ratios. This phenomenon can be explained from two aspects: on one hand, when LLM is powerful, it can generate the correct XPath without the process of step-back to re-accessing the sub-DOM tree; on the other hand, when the model is weak, it is unable to effectively understand the hierarchical structure of web page, and thus cannot generate reliable, effective XPaths for the web page.

XPath fragility within AUTOSCRAPER The fragility of XPath often refers to the characteristic of XPath expressions becoming ineffective or inaccurately matching the target element when faced with new webpages. This is mainly due to XPath specifying specific information through *predicates*, such as text, `@class`, etc.

We mainly focus on the fragility of text because these webpages are from the same websites (i.e. `@class` is a good characteristic for generating stable action sequences). Table 7 shows XPath expressions that rely on text. We aim to explore the reusability of generating XPath based on text features. We manually calculated the proportion of bad cases with two types of predicates, *contains* and *equal* [5]. The results in Table 8 show that the

[5]https://www.w3schools.com/xml/xpath_

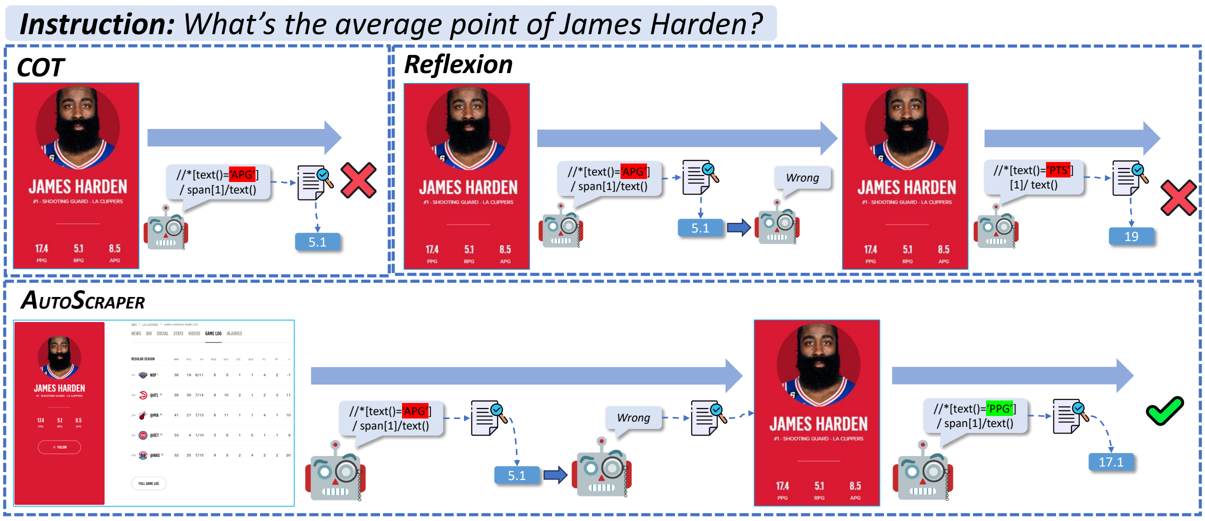

Figure 4: Comparison with the other two baselines.

	Good case	Bad case
Question	*Here's a webpage on detail information with detail information of an NBA player. Please extract the **height** of the player.*	*Here's a webpage with detailed information about a university. Please extract the **contact phone number** of the university.*
Case	*//div[@class='gray200B-dyContent']/ b[**contains(text(),'Height:')**]/following-sibling::text()*	*//div[@class='infopage']//h5[**contains (text(), '703-528-7809')**]*

Table 7: Examples of XPath fragility. The green focuses on the common information across different webpages, while the red focuses on specific information of seed webpages.

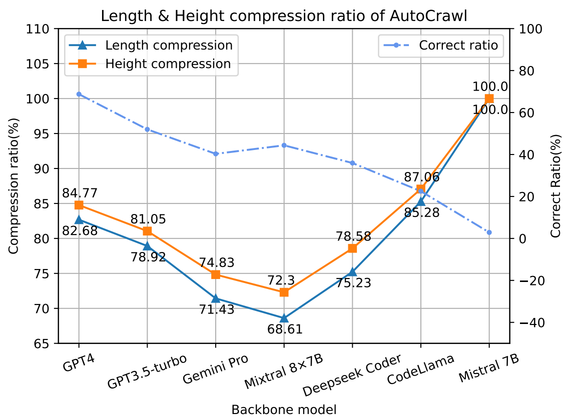

Figure 5: The curve on length and height compression ratio in SWDE dataset.

stronger LLMs capability, the lower the proportion of bad cases with AUTOSCRAPER . However, it should be noted that the current SoTA LLM GPT-4-Turbo still suffers from an XPath fragility problem, which indicates that relying entirely on LLMs to generate reliable XPath still has some distance to go.

Models	Contains	Equal(=)
GPT4	0.61%	2.90%
GPT-3.5-Turbo	9.33%	9.78%
Gemini Pro	10.62%	14.29%
Mixtral 8×7B	12.88%	8.55%
Deepseek-Coder	11.63%	7.55%
CodeLlama	18.75%	14.29%
Mistral 7B	18.18%	33.33%

Table 8: Bad case ratio in two types of predicate.

C Dataset Statistic

Table 11, 12, 13 shows the detailed statistic about the semi-structure web information extraction dataset SWDE, EXTENDED SWDE and DS1.

D Prompt List

Table 14 shows the task prompt we design for each attribute for SWDE.

syntax.asp

Models	Method	Executable Evaluation						IE Evaluation		
		Correct(↑)	Prec	Reca	Unex.(↓)	Over.	Else	Prec	Reca	F1
Closed-source LLMs										
GPT-3.5-Turbo	COT	34.49	3.48	4.53	56.10	0.35	1.05	87.96	42.16	40.58
	Reflexion	43.90	1.74	2.09	49.13	0.35	2.79	**93.46**	49.58	48.66
	AutoScraper	**45.30**	4.18	8.01	**35.89**	0.35	6.27	83.60	**60.84**	**56.69**
Gemini Pro	COT	34.49	2.09	6.62	49.13	0.35	7.32	81.09	46.55	42.40
	Reflexion	34.15	2.09	6.97	51.57	0.35	4.88	**84.43**	45.19	41.66
	AutoScraper	**35.89**	5.23	10.10	**42.86**	0.35	5.57	83.74	**52.75**	**47.73**
GPT4	COT	55.05	2.44	7.32	30.31	0.35	4.53	84.11	67.31	64.04
	Reflexion	**63.76**	3.83	5.57	20.91	0.35	5.57	**86.00**	76.50	74.50
	AutoScraper	63.07	3.48	5.92	**16.72**	0.35	10.45	81.29	**78.77**	**74.77**
Open-source LLMs										
CodeLlama	COT	9.01	1.29	2.15	85.84	0.00	1.72	87.22	12.62	11.21
	Reflexion	**13.73**	1.72	3.00	**80.26**	0.00	1.29	89.41	**17.76**	**16.01**
	AutoScraper	11.16	0.00	1.72	85.84	0.00	1.29	**92.49**	13.29	12.52
Mixtral 8×7B	COT	31.36	1.05	4.88	58.19	0.35	4.18	**86.83**	40.16	37.25
	Reflexion	29.62	1.05	4.18	62.02	0.35	2.79	83.44	36.44	33.64
	AutoScraper	**40.07**	3.83	9.41	**39.37**	0.35	6.97	81.63	**57.10**	**51.57**
Deepseek-coder	COT	**38.33**	3.83	6.62	**47.74**	0.35	3.14	81.32	**48.52**	**44.80**
	Reflexion	36.24	3.48	3.83	51.92	0.00	4.53	83.53	45.03	43.64
	AutoScraper	37.63	2.44	5.92	50.52	0.35	3.14	**86.91**	47.09	44.33

Table 9: The executable evaluation and IE evaluation of LLMs with three frameworks in Extended Swde dataset. We examine 6 LLMs, including 3 closed-source LLMs and 3 open-source LLMs.

Models	Method	Executable Evaluation						IE Evaluation		
		Correct(↑)	Prec	Reca	Unex.(↓)	Over.	Else	Prec	Reca	F1
Closed-source LLMs										
GPT-3.5-Turbo	COT	32.65	4.08	8.16	53.06	0.00	2.04	90.56	43.54	41.16
	Reflexion	36.73	8.16	4.08	51.02	0.00	0.00	**95.56**	44.22	43.75
	AutoScraper	**48.98**	4.08	0.00	**44.90**	0.00	2.04	94.90	**51.70**	**52.38**
Gemini Pro	COT	17.72	2.53	3.80	75.95	0.00	0.00	90.82	22.88	22.10
	Reflexion	20.25	10.13	1.27	65.82	0.00	2.53	88.83	26.93	27.66
	AutoScraper	**43.04**	15.19	3.80	**34.18**	0.00	3.80	**93.76**	**55.97**	**56.92**
GPT4	COT	50.60	9.64	6.02	30.12	0.00	3.61	93.60	65.75	64.73
	Reflexion	50.60	10.84	4.82	33.73	0.00	0.00	**96.85**	62.65	63.50
	AutoScraper	**57.83**	15.66	4.82	**16.87**	0.00	4.82	92.88	**74.95**	**75.52**
Open-source LLMs										
CodeLlama	COT	2.70	2.70	5.41	89.19	0.00	0.00	78.72	10.62	9.19
	Reflexion	8.82	0.00	5.88	85.29	0.00	0.00	**94.12**	14.41	12.69
	AutoScraper	**13.51**	0.00	5.41	**81.08**	0.00	0.00	84.12	**18.92**	**17.39**
Mixtral 8×7B	COT	17.72	6.33	0.00	74.68	0.00	1.27	**94.81**	21.15	22.01
	Reflexion	22.78	6.33	1.27	69.62	0.00	0.00	94.15	28.03	28.20
	AutoScraper	**36.71**	11.39	6.33	**43.04**	0.00	2.53	91.59	**48.52**	**48.23**
Deepseek-coder	COT	25.30	9.64	2.41	60.24	0.00	2.41	**92.47**	34.71	35.65
	Reflexion	22.89	6.02	3.61	65.06	0.00	2.41	90.21	31.43	32.04
	AutoScraper	**39.76**	10.84	6.02	**42.17**	0.00	1.20	90.43	**51.39**	**50.28**

Table 10: The executable evaluation and IE evaluation of LLMs with three frameworks in Ds1 dataset. We examine 6 LLMs, including 3 closed-source LLMs and 3 open-source LLMs.

Domain	Attribute	Website	Num	Domain	Attribute	Website	Num
Auto	model	aol	2000	**Movie**	ttitle	allmovie	2000
	price	autobytel	2000		director	amctv	2000
	engine	automotive	1999		genre	boxofficemojo	2000
	fuel_economy	autoweb	2000		mpaa_rating	hollywood	2000
		carquotes	2000			iheartmovies	2000
		cars	657			imdb	2000
		kbb	2000			metacritic	2000
		motortrend	1267			msn	2000
		msn	2000			rottentomatoes	2000
		yahoo	2000			yahoo	2000
Book	title	abebooks	2000	**NBAPlayer**	name	espn	434
	author	amazon	2000		team	fanhouse	446
	isbn_13	barnesandnoble	2000		height	foxsports	425
	publisher	bookdepository	2000		weight	msnca	434
	pub_date	booksamillion	2000			nba	434
		bookorders	2000			si	515
		buy	2000			slam	423
		christianbook	2000			usatoday	436
		deepdiscount	2000			wiki	420
		waterstone	2000			yahoo	438
Camera	model	amazon	1767	**Restaurant**	name	fodors	2000
	price	beachaudio	247		address	frommers	2000
	manufacturer	buy	500		phone	zagat	2000
		compsource	430		cuisine	gayot	2000
		ecost	923			opentable	2000
		jr	367			pickaretaurant	2000
		newegg	220			restaurantica	2000
		onsale	261			tripadvisor	2000
		pcnation	234			urbanspoon	2000
		thenerd	309			usdiners	2000
Job	title	careerbuilder	2000	**University**	name	collegeboard	2000
	company	dice	2000		phone	collegenavigator	2000
	location	hotjobs	2000		website	collegeprowler	2000
	date_posted	job	2000		type	collegetoolkit	2000
		jobcircle	2000			ecampustours	1063
		jobtarget	2000			embark	2000
		monster	2000			matchcollege	2000
		nettemps	2000			princetonreview	615
		rightitjobs	2000			studentaid	2000
		techcentric	2000			usnews	1027

Table 11: Detail statistic of SWDE dataset.

Domain	Website	# Attributes
Movie	allmovie	20
	amctv	13
	hollywood	12
	iheartmovies	8
	imdb	34
	metacritic	17
	rottentomatoes	10
	yahoo	10
NBAPlayer	espn	10
	fanhouse	14
	foxsports	10
	msnca	12
	si	12
	slam	12
	usatoday	5
	yahoo	9
University	collegeprowler	18
	ecampustours	14
	embark	23
	matchcollege	15
	usnews	19

Table 12: Detail statistic of EXTEND SWDE dataset.

Domain	Attribute	Website
Book	title author price	abebooks alibris barnesandnoble fishpond infibeam powells thriftbooks
E-commerce	title price	amazoncouk bestbuy dabs ebay pcworld tesco uttings
Hotel	address price title	agoda expedia hotels hoteltravel javago kayak ratestogo venere
Movie	actor genre title	123movieto hollywoodreporter imdb mediastinger metacritic rottentomatoes themoviedb yidio

Table 13: Detail statistic of DS1 dataset.

Domain	Task prompt	Prompt
Auto	Here's a webpage with detailed information about an auto.	Please extract the model of the auto. Please extract the price of the auto. Please extract the engine of the auto. Please extract the fuel efficiency of the auto.
Book	Here's a webpage with detailed information about a book.	Please extract the title of the book. Please extract the author of the book. Please extract the isbn number of the book. Please extract the publisher of the book. Please extract the publication date of the book.
Camera	Here's a webpage with detail information of camera.	Please extract the product name of the camera. Please extract the sale price of the camera. Please extract the manufacturer of the camera.
Job	Here's a webpage with detailed information about a job.	Please extract the title of the job. Please extract the name of the company that offers the job. Please extract the working location of the job. Please extract the date that post the job.
Movie	Here's a webpage with detailed information about a movie.	Please extract the title of the movie. Please extract the director of the movie. Please extract the genre of the movie. Please extract the MPAA rating of the movie.
NBAPlayer	Here's a webpage with detailed information about an NBA player.	Please extract the name of the player. Please extract the team of the player he plays now. Please extract the height of the player. Please extract the weight of the player.
Restaurant	Here's a webpage with detailed information about a restaurant.	Please extract the restaurant's name. Please extract the restaurant's address. Please extract the restaurant's phone number. Please extract the cuisine that the restaurant offers.
University	Here's a webpage on detailed information about a university.	Please extract the name of the university. Please extract the contact phone number of the university. Please extract the website url of the university. Please extract the type of the university.

Table 14: Prompts for crawler generation task in SWDE dataset.