XPath Agent: An Efficient XPath Programming Agent Based on LLM for Web Crawler

Li Yu
lijingyu68@gmail.com

Bryce Wang
Stanford University
brycewang2018@gmail.com

Hazul
Research Dept.
Science City, Sci 88088
yvo@science.rdept.net

Abstract

We introduce XPath Agent, a production-ready XPath programming agent tailored for web crawling tasks. A standout feature of XPath Agent is its capability to automatically program XPath queries from a set of sampled web pages. To illustrate its efficacy, we benchmark XPath Agent against a state-of-the-art XPath programming agent across a suite of web crawling tasks. Our findings reveal that XPath Agent excels in F1 score with minimal compromise on accuracy, while significantly reducing token usage and increase clock-time efficiency. The well designed 2 stage pipelines makes it readily integrable into existing web crawling workflows, thereby saving time and effort in manual XPath query development.

1 Introduction

Web scraping [2] automates data extraction from websites, vital for modern fields like Business Intelligence. It excels in gathering structured data from unstructured sources like HTML, especially when machine-readable formats are unavailable. Web scraping provides real-time data, such as pricing from retail sites, and can offer insights into illicit activities like darknet drug markets.

The advent of HTML5 [4] has introduced significant complexities to automated web scraping, exacerbating issues of fragmentation and protocol diversity that have long challenged the development of web standards. These complexities stem from the enhanced capabilities and dynamic nature of HTML5, which require more sophisticated methods to accurately extract and interpret data. As traditional web scraping techniques struggle to keep pace with these advancements, there is a growing need for innovative solutions to navigate the intricacies of modern web technologies.

The development of Large Language Models (LLM) has emerged as a promising avenue. LLMs, with their advanced natural language processing capabilities, offer a new paradigm for understanding and interacting with web content. AutoWebGLM[3] demonstrated significant advancements in addressing the complexities of real-world web navigation tasks, particularly in simplifying HTML data representation to enhancing it's capability. By leveraging reinforcement learning and rejection sampling, AutoWebGLM enhanced its ability to comprehend webpages, execute browser operations, and efficiently decompose tasks.

Instead of fine-tuning LLMs, AutoScraper[1] adopt a simplified technique which involves traversing the webpage and constructing a final XPath using generated action sequences. By focusing on the hierarchical structure of HTML and leveraging similarities across different web pages, its significantly reduces the complexity and computational overhead.

Scrolling, navigating, back and forward through webpages to program XPath queries is a nature way for human. But for LLM, this process is time-consuming and computationally expensive. We expect LLM can do better and efficient.

1.1 Motivation

We assuming there are 3 core reasons why LLMs are not efficient in generating XPath queries. Firstly, LLMs are not designed to generate XPath queries. Secondly, web pages are lengthy and complex, full of task unrelated information. Those information dis-

In: W. Aigner, G. Schmiedl, K. Blumenstein, M. Zeppelzauer (eds.): Proceedings of the 9th Forum Media Technology 2016, St. Pölten, Austria, 24-11-2016, published at http://ceur-ws.org

tract the LLMs from generating the correct XPath queries. Thirdly, LLMs are context limited. A good XPath query should be generalizable across different web pages. However, LLMs can only generate XPath queries based on the context they have seen. So, a shorter and more task-related context is more likely to generate a better XPath query.

Based on the above insights, we propose a novel approach to generate XPath queries using LLMs. We aim to reduce the number of steps required to generate a well-crafted XPath query, reduce the computational overhead, and improve the generalizability of the XPath queries generated by LLMs.

In order to increase the efficiency of XPath query generation, we also employed LangGraph. Which is a graph-based tool set which we can define the whole pipeline in a graph-based manner and execute it in parallel. Which significantly reduce the time required to generate XPath queries.

1.2 Our Contributions

We based each of the above motivations and make the following contributions:

1. We designed a two stage pipeline, which we can employ a weaker LLM to extract target information. And a stronger LLM to program XPath.
2. We proposed a simple way to prune the web page, which can reduce the complexity of the web page and make the LLM focus on the target information.
3. We discovered that extracted cue texts from 1st stage significantly improve the performance of the 2nd stage.
4. We benchmarked our approach against a state-of-the-art same purpose agent across a suite of web crawling tasks. Our findings reveal that our approach excels in F1 score with minimal compromise on accuracy, while significantly reducing token usage and increase clock-time efficiency.

2 Related Work

2.1 LLMs and Direct Webpage Information Extraction

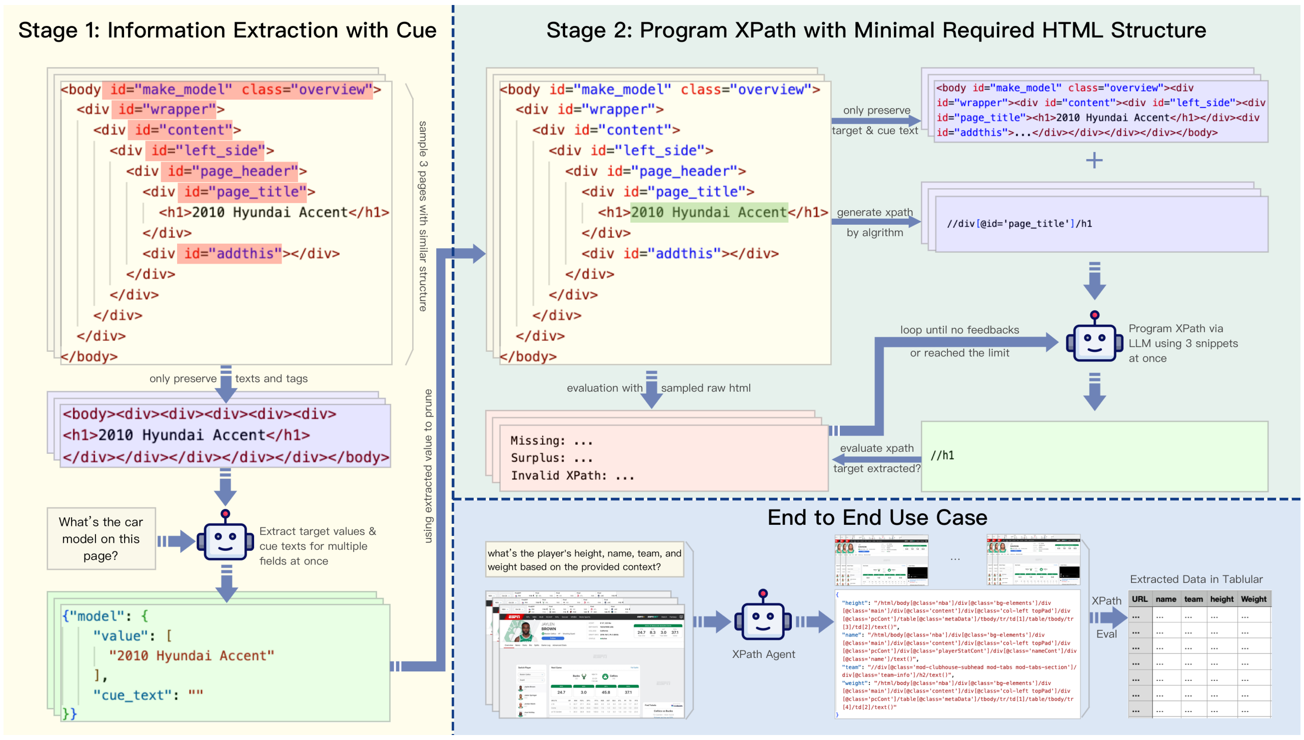
Large Language Models (LLMs) have demonstrated remarkable abilities in replicating human-like task execution, particularly in web data extraction. Recent advancements have enabled LLMs to process entire web pages post-crawling, facilitating direct extraction of structured information such as product details and prices. This method capitalizes on the LLMs' capabilities to interpret complex HTML structures without manual rule-based configurations, enhancing adaptability across dynamic web environments (Ahluwalia et al., 2024). However, challenges remain, particularly in ensuring factual accuracy and avoiding excessive reliance on large, computationally intensive models (Xu et al., 2024). Despite these hurdles, this approach offers a promising alternative to traditional web scraping, as demonstrated by models like NeuScraper, which achieve improved accuracy by integrating neural networks for direct HTML text extraction (Xu et al., 2024).

2.2 LLMs and Webpage Content Simplification

An alternative approach to direct information extraction involves simplifying web content through LLMs before targeting specific data points. This method filters out redundant HTML elements, reducing the size and complexity of web data while retaining the relevant information for extraction. By employing techniques such as chunking and semantic classification, LLMs like those used in RAG models can focus on high-value segments, improving processing efficiency (Ahluwalia & Wani, 2024) Studies have shown that simplifying web pages using LLMs significantly enhances their performance in dynamic environments, especially when dealing with varied and noisy HTML structures (Deng et al., 2023). Moreover, this approach aligns with efforts to reduce the cognitive load on models, particularly in contexts requiring rapid, scalable information extraction (Sàrl and Godin, 2023).

2.3 LLMs and XPaths for Information Extraction

Generating generalizable XPath queries using LLMs has emerged as a robust method for automating web information extraction across structurally similar websites. This approach leverages LLMs' understanding of document structures to create XPath queries that dynamically adapt to minor variations in web page layouts, increasing the scalability of extraction systems. Tools like TREEX, which integrate decision tree learning, allow for the synthesis of XPaths that balance precision and recall across multiple web pages, even those with unseen structures (Omari et al., 2024). This technique significantly reduces the need for manual intervention and facilitates the creation of highly efficient and reusable extractors for tasks such as price comparison and product aggregation across e-commerce platforms (Huang et al., 2024).


Stage 1: Information Extraction with Cue
sample 3 pages with similar structure
only preserve texts and tags
What's the car model on this page?
Extract target values & cue texts for multiple fields at once
using extracted value to prune
Stage 2: Program XPath with Minimal Required HTML Structure
only preserve target & cue text
generate xpath by algrithm
loop until no feedbacks or reached the limit
Program XPath via LLM using 3 snippets at once
evaluation with sampled raw html
evaluate xpath target extracted?
End to End Use Case
what's the player's height, name, team, and weight based on the provided context?
XPath Agent
XPath Eval
Extracted Data in Tablular
URL
name
team
height
Weight

```
<body id="make_model" class="overview">
  <div id="wrapper">
    <div id="content">
      <div id="left_side">
        <div id="page_header">
          <div id="page_title">
            <h1>2010 Hyundai Accent</h1>
          </div>
          <div id="addthis"></div>
        </div>
      </div>
    </div>
  </div>
</body>
```

```
<body><div><div><div><div><div>
<h1>2010 Hyundai Accent</h1>
</div></div></div></div></div></body>
```

```
{"model": {
    "value": [
      "2010 Hyundai Accent"
    ],
    "cue_text": ""
}}
```

```
<body id="make_model" class="overview"><div
id="wrapper"><div id="content"><div id="left_side"><div
id="page_title"><h1>2010 Hyundai Accent</h1></div><div
id="addthis">...</div></div></div></div></body>
```

```
//div[@id='page_title']/h1
```

```
Missing: ...
Surplus: ...
Invalid XPath: ...
```

```
//h1
```

```
{
  "height": "/html/body[@class='nba']/div[@class='bg-elements']/div
  [@class='main']/div[@class='content']/div[@class='col-left topPad']/div
  [@class='pcCont']/table[@class='metaData']/tbody/tr/td[1]/table/tbody/tr
  [3]/td[2]/text()",
  "name": "/html/body[@class='nba']/div[@class='bg-elements']/div
  [@class='main']/div[@class='content']/div[@class='col-left topPad']/div
  [@class='pcCont']/div[@class='playerStatCont']/div[@class='nameCont']/div
  [@class='name']/text()",
  "team": "//div[@class='mod-clubhouse-subhead mod-tabs mod-tabs-section']/
  div[@class='team-info']/h2/text()",
  "weight": "/html/body[@class='nba']/div[@class='bg-elements']/div
  [@class='main']/div[@class='content']/div[@class='col-left topPad']/div
  [@class='pcCont']/table[@class='metaData']/tbody/tr/td[1]/table/tbody/tr
  [4]/td[2]/text()"
}
```

Figure 1: XPath Agent of two stages pipeline. The first stage is Information Extraction, which extracts target information and cue text from sanitized web pages (the red are sanitized). The second stage is XPath Programming, which generates XPath queries based on condensed html (the green are kept) and generated XPath.

3 Methodology

In this section, we present the methodology of our approach, which consists of two stages: Information Extraction (IE) and XPath Programming. The IE stage extracts target information from sanitized web pages, while the XPath Programming stage generates XPath queries based on the condensed html and extracted information. 1 illustrates the two-stage pipeline of XPath Agent. For each stage, we provide a detailed description of the process and the algorithms used.

3.1 Information Extraction with Cue Text

The Information Extraction (IE) stage aims to extract target information. Not like traditional IE, we discovered 2 key insights. Firstly, we prompt the LLM to not only extract questioned information but also extract cue texts. Cue texts are the texts that are close to the target information and can help the LLM to generate better XPath queries. Secondly, we sanitize the web page before feeding it to the LLM. Sanitizing the web page can reduce the complexity of the web page and make the LLM focus on the target information.

Algorithm 1: IE HTML Sanitizer

```
Input: Root node of HTML tree $root\_node$
Output: Sanitized HTML tree
$left\_stack \leftarrow [root\_node]$;
$right\_stack \leftarrow []$;
while $left\_stack$ is not empty do
    $node \leftarrow left\_stack.pop()$;
    $right\_stack.append(node)$;
    $left\_stack \leftarrow$
      $left\_stack + list(node.iterchildren())$;
end
while $right\_stack$ is not empty do
    $node \leftarrow right\_stack.pop()$;
    if $is\_invisiable\_or\_no\_text(node)$ then
        $node.getparent().remove(node)$;
    end
    else
        $node.remove\_attributes()$;
    end
end
```

Algorithm 2: HTML Condenser

```
Input: root: HTML root node;
target_texts: List of target texts to keep;
d: Distance function between two texts;
Output: root: Condensed HTML root node;
target_texts ← [];
distances ← {};
eles ← {};
foreach ele, text in iter_with_text(root) do
    foreach target_text in target_texts do
        distance ← d(text, target_text);
        if distance < distances[text] then
            distances[text] ← distance;
            eles[text] ← [get_xpath(ele)];
        end
        else if distance == distances[text] then
            eles[text].append(get_xpath(ele));
        end
    end
end
targets ← concat(values(eles));
foreach xpath, ele in iter_with_xpath(root) do
    if is_outside(xpath, targets) then
        remove_children(ele);
        replace_text_to(ele, "...");
    end
end
```

3.2 XPath Program

4 Experiments

...

5 Conclusion

Third level headings must be flush left, initial caps and bold. One line space before the third level heading and 1/2 line space after the third level heading.

Fourth Level Heading

Fourth level headings must be flush left, initial caps and roman type. One line space before the fourth level heading and 1/2 line space after the fourth level heading.

5.1 Citations In Text

...

5.1.1 Footnotes

Indicate footnotes with a number[1] in the text. Place the footnotes at the bottom of the page they appear on. Precede the footnote with a vertical rule of 2 inches (12 picas).

5.1.2 Figures

All artwork must be centered, neat, clean and legible. Do not use pencil or hand-drawn artwork. Figure number and caption always appear after the the figure. Place one line space before the figure, one line space before the figure caption and one line space after the figure caption. The figure caption is initial caps and each figure is numbered consecutively.

Make sure that the figure caption does not get separated from the figure. Leave extra white space at the bottom of the page to avoid splitting the figure and figure caption.

Below is another figure using LaTeX commands.

5.1.3 Tables

All tables must be centered, neat, clean and legible. Do not use pencil or hand-drawn tables. Table number and title always appear before the table.

One line space before the table title, one line space after the table title and one line space after the table. The table title must be initial caps and each table numbered consecutively.

5.1.4 Handling References

Use a first level heading for the references. References follow the acknowledgements.

[1]This is a sample footnote

Table 1: Sample Table

A	B	1
C	D	2
E	F	3

5.1.5 Acknowledgements

Use a third level heading for the acknowledgements. All acknowledgements go at the end of the paper.

References

[1] Wenhao Huang, Zhouhong Gu, Chenghao Peng, Zhixu Li, Jiaqing Liang, Yanghua Xiao, Liqian Wen, and Zulong Chen. Autoscraper: A progressive understanding web agent for web scraper generation, 2024.

[2] Moaiad Ahmad Khder. Web scraping or web crawling: State of art, techniques, approaches and application. *International Journal of Advances in Soft Computing & Its Applications*, 13(3), 2021.

[3] Hanyu Lai, Xiao Liu, Iat Long Iong, Shuntian Yao, Yuxuan Chen, Pengbo Shen, Hao Yu, Hanchen Zhang, Xiaohan Zhang, Yuxiao Dong, and Jie Tang. Autowebglm: A large language model-based web navigating agent, 2024.

[4] Raúl Tabarés. Html5 and the evolution of html; tracing the origins of digital platforms. *Technology in Society*, 65:101529, 2021.