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

Yu Li
lijingyu68@gmail.com

Bryce Wang
Stanford University
brycewang2018@gmail.com

Xinyu Luan
xinyuluan0320@gmail.com

Abstract

We present XPath Agent, a production-ready XPath programming agent specifically designed for web crawling and web GUI testing. A key feature of XPath Agent is its ability to automatically generate XPath queries from a set of sampled web pages using a single natural language query. To demonstrate its effectiveness, we benchmark XPath Agent against a state-of-the-art XPath programming agent across a range of web crawling tasks. Our results show that XPath Agent achieves comparable performance metrics while significantly reducing token usage and improving clock-time efficiency. The well-designed two-stage pipeline allows for seamless integration into existing web crawling or web GUI testing workflows, thereby saving time and effort in manual XPath query development. The source code for XPath Agent is available at `https://github.com/eavae/feilian`.

1 Introduction

Web scraping [3] 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 [6] has introduced significant complexities to automated web scraping. These complexities stem from the enhanced capabilities and dynamic nature of HTML5, which require more sophisticated methods to accurately extract and interpret data. To address these challenges, researchers have developed a variety of tools, such as Selenium [5]. Which offers a set of application programming interfaces (APIs) to automate web browsers, enabling the extraction of data from web pages. However, program Selenium to extract data from web pages is a time-consuming and error-prone process. The most crucial part is how to locate the target information on the web page. XPath queries provide a solution to this problem by allowing users to navigate the HTML structure of a web page and extract specific elements. But, programming XPath queries is a challenging task, especially for non-technical users.

In addition to web crawling, XPath is also essential for web GUI testing. Web GUI testing involves interacting with web elements to verify that the user interface behaves as expected. XPath queries are used to locate elements on a web page, such as buttons, input fields, and links, which are then interacted with during the testing process. By automating these interactions, testers can efficiently validate the functionality and appearance of web applications across different browsers and devices. The ability to generate accurate and robust XPath queries is crucial for ensuring comprehensive test coverage and reliable test results.

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[4] 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 one time task execution, AutoScraper[2] adopt a simplified technique which only involves text content of webpages. By focusing on the hierarchical structure of HTML and traversing the webpage, it con-

struct a final XPath using generated action sequences. Such XPath is generalizable and can be applied to multiple webpages. Which significantly reduce the time required when execution.

But, the above approaches are not efficient in generating XPath queries. We introduced a more effective approach to generate XPath queries using LLMs which could simply integrate into existing web crawling or web GUI testing workflows.

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 distract 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 to generate XPath queries.

1.2 Our Contributions

In summary, our contributions are as follows:

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 Generative Information Extraction

Large Language Models (LLMs) are increasingly being used for generative information extraction (IE), where they directly generate structured knowledge from text—such as entities, relations, and events—offering an alternative to traditional discriminative methods (Xu et al., 2024). LLMs are especially advantageous in low-resource settings and support multitasking formats, which enhances their adaptability across various IE tasks. These tasks are typically categorized into Named Entity Recognition (NER), Relation Extraction (RE), and Event Extraction (EE), with a rigorous comparison of models' performance in each area (Xu et al., 2024).

Recent universal IE frameworks employ both natural language-based LLMs (NL-LLMs) and code-based LLMs (Code-LLMs). NL-LLMs, like UIE and InstructUIE, use natural language prompts to generate structured information. In contrast, Code-LLMs, such as Code4UIE and CodeKGC, leverage code-based schemas, offering more precise knowledge representation (Gan et al., 2023; Wei et al., 2023; Bi et al., 2024). In addition to text-based IE, advancements in LLMs have facilitated information extraction from web data. These models now enable processing of entire web pages post-crawling, extracting structured information such as product details and prices without manual, rule-based configurations (Ahluwalia et al., 2024). This web data extraction capability utilizes LLMs' proficiency in interpreting complex HTML structures, enhancing flexibility across dynamic online environments. However, challenges remain in ensuring factual accuracy and managing the computational demands of large models (Xu et al., 2024). Models like NeuScraper have addressed some of these challenges by integrating neural networks for direct HTML text extraction, yielding more accurate results and offering a promising alternative to traditional web scraping methods (Xu et al., 2024).

2.2 LLMs and XPaths for Information Extraction

As a specific generation technique of large models, generating Xpath for web information retrieval is also an efficient method for automating web information extraction. This approach leverages LLMs' understanding of document structures to create XPath queries that dynamically adapt to minor variations in web

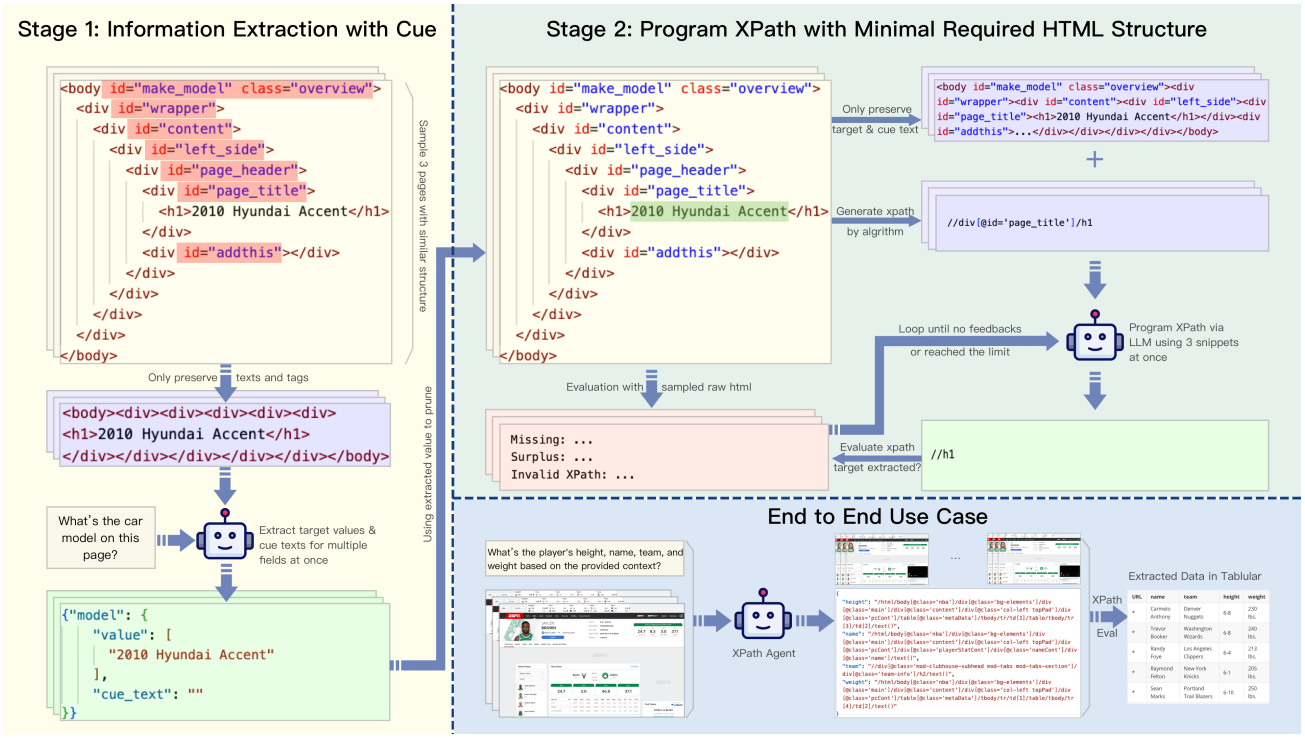

Figure 1: XPath Agent of two stages pipeline. The first stage is Information Extraction, which extracts target information and cue text from sanitized web pages (the red are sanitized). The second stage is XPath Programming, which generates XPath queries based on condensed html (the greens are target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

page layouts, increasing the scalability of extraction systems for structurally similar websites. 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 (AUTOSCRAPER, Huang et al., 2024). Specifically, AUTOSCRAPER employs a progressive generation phase to traverse HTML structures and a synthesis phase to refine reusable action sequences across similar web pages, enhancing scalability and efficiency in dynamic environments.

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 by extracted information. The whole process based on seeded web pages, which is 3 in our implementation. Figure 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 LLM not only extract questioned information but also cue texts. Secondly, we sanitized the web page to reduce the complexity of the web page and make the LLM focus on the target information with contextual semantic.

Cue texts are the indicative texts that signals the upcoming target information. For example, for "price: $100.00", the cue text is "price:". Those texts are important in some case, especially when no way or hard to directly programming XPath queries to extract target information "$100.00". In such case, treat "price:" as an anchor, using XPath's ancestor, sibling, or descendant axis traverse to the target information is the only way. In order to let the context still be condensed, we prompt LLM to response cue texts simultaneously.

Sanitizing web page is a process to remove unnecessary information from a page. In HTML, the most

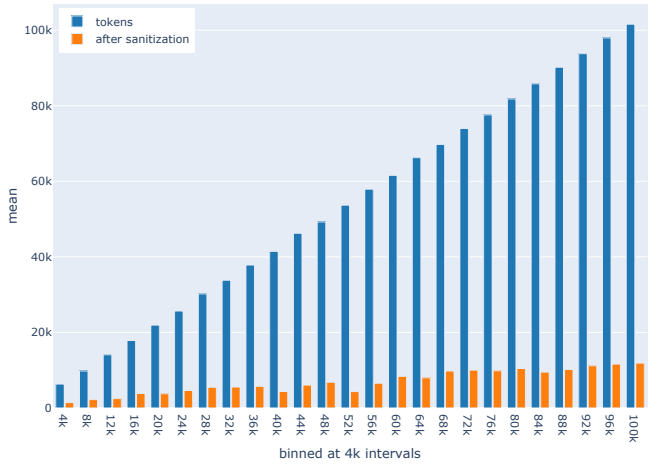

Figure 2: Token Stats Analysis with Algorithm 1. As page size grow, the size after sanitization increased slowly (sampled 128 pages for each category from SWDE dataset, around 10k pages totally).

meaningful parts are texts and tags. The texts are the target information we want to extract, and the tags are the structure of the web page which tells the relationship between texts especially the priority of which answer is more likely to be the target information. The purpose of sanitizing the web page is to reduce the complexity of the web page and make the LLM focus on the target information. We designed an algorithm to sanitize the web page, which is shown in Algorithm 1. We also employed minify-html[7] to further reduce the size of the web page.

The algorithm 1 traverse the HTML tree in a depth-first manner. It removes the invisible or empty nodes, and all attributes. It's efficient and can be easily implemented in any programming language. In our sampled web pages, it can help us reduce the size of the web page to 10% 20% on average.

In our implementation, we prompt LLM to extract all information at once on a single page with JSON format. So, multiple fields or multiple values for a single field might be extracted. We treat all extracted values are relevant and passing them to the next stage. The prompt for the Information Extraction stage is shown in Section A.

3.2 Program XPath

Program XPath queries is a process to generate XPath queries based on the condensed html by extracted information. In order to let LLM have more context to program a robust XPath query, we condensed the html by the extracted information and prompt with 3 seeded web pages at once. The algorithm is shown in Algorithm 2.

The condenser based on the extracted 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

which is the target information and cue texts. A distance function is used to identify the most relevant nodes so that we can keep them in the condensed html. During the condense process, we keep the target nodes and replace other nodes' children with "...".

3.3 Static XPath Generation

Program a robust XPath query is a challenging task. Which requires to balance between rigidity and flexibility. The rigidity means the XPath query should strictly follow the specific structure of the web page. The flexibility means the XPath query should be pruned to be generalizable across different web pages. In our early experiments, we discovered that the XPath query generated by LLM is not sticky to the structure of the web page. So, we designed a static XPath generation algorithm to guide LLMs.

The static XPath generation algorithm propagate from the target node to root node. It generates the XPath query in a bottom-up manner. Unlike naive XPath generation algorithm. We add more attributes (in our case, we include class and id) to the XPath query. It makes the XPath query richer.

3.4 Conversational XPath Evaluator

LLM are lacks of the environment to evaluate, correct or improve the XPath queries. So, we designed a evaluator to evaluate LLM generated XPath.

The XPath evaluator is a function, which execute the XPath query on seeded web pages and feedback the result to LLM. The result include 3 parts: what are missing, what are redundant, and correctness of

the XPath query. The LLM can based on feedback to improve the XPath query.

In our implementation, we limited the number of feedback loop to 3. At the end of the loop, we take the best XPath query based on evaluation result as the final result. The prompt for the Program XPath stage is shown in Section B.

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
```

4 Experiments

4.1 Experimental Setup

4.1.1 Models

We use DeepSeek and ChatGPT-4.0 as the primary large language models in our experiments.

4.1.2 Dataset

We use the SWDE [1] (Structured Web Data Extraction) dataset, which includes 90 of websites across 9 domains, in total 20414 web pages.

4.1.3 Experimental Parameters

We use DeepSeek-Chat as the main model for our XPath Agent. The Number of Seeds is 3 initial seeds are provided to guide query generation and the Sample Size: 32 web pages are sampled per task to evaluate the model's adaptability and generalizability.

4.2 Evaluation Metrics

For evaluation, we employ the metrics of precision, recall, F1 score, and accuracy. we utilized a set matching method to calculate the metrics for multi-label classification tasks where the labels are unordered. This involves comparing each predicted label set with the corresponding ground truth label set while disregarding the order of the labels. First, Both the ground truth and predictions are converted into sets to ignore any order. Accuracy is then defined such that a prediction is counted as correct if the predicted set exactly matches the ground truth set. We count correctly predicted labels as True Positives. Incorrectly predicted labels that are not in ground truth are classified as False Positives, while the labels that should have been in predicted set but were missed are designated as False Negatives. Additionally, since our data does not contain blank answers (with only a very few exceptions that have been removed), our cases do not have true negatives. Finally, precision, recall, F1 score, and accuracy are calculated by the following formula.

$$presision = \frac{TurePositives}{TurePositives+FalsePositives}$$

$$recall = \frac{TurePositives}{TruePositives+FalseNegatives}$$

$$F1 = \frac{2\times precision\times recall}{precision+recall}$$

$$accuracy = \frac{TruePositives}{TruePositives+FalsePositives+FalseNegatives}$$

5 Results and Analysis

5.1 Statistical Analysis

DeepSeek is strong in precision, making it good at avoiding irrelevant data. GPT 4o is highly effective at capturing a wide range of relevant content, ensuring that few important elements are missed. Claude 3.0 strikes a balanced approach, effectively combining both precision and recall for solid overall performance of F1 score. Claude 3.5 stands out as the most balanced and effective model, excelling across all key areas of accuracy, precision, recall, and overall performance. It provides the best mix of identifying relevant data while minimizing errors.

5.2 Comparative Analysis

The AutoCrawler [2] framework focuses on generating web crawlers for extracting specific information

Model	Accuracy	Precision	Recall	F1
DeepSeek V2.5	0.5794	**0.6764**	0.8017	0.7337
GPT 4o	0.5793	0.6423	**0.8553**	0.7336
Claude 3.0	0.6091	0.6796	0.8546	**0.7571**
Claude 3.5	**0.6191**	**0.6916**	**0.8551**	**0.7647**

Table 1: Experimental Results

from semi-structured HTML. It is designed with a two-phase approach: the first phase uses a progressive generation framework that leverages the hierarchical structure of HTML pages, while the second phase employs a synthesis framework that improves crawler performance by learning from multiple web pages.

In comparison to XPath Agent, AutoCrawler presents a different approach, emphasizing a sequence of XPath actions rather than just the extraction of XPath from snapshots. This difference may influence performance in various metrics, such as F1 score. AutoCrawler's focus on refining action sequences based on learning from past errors might offer advantages in terms of robustness and adaptability to dynamic web structures. However, your XPath Agent, by isolating XPath extraction tasks, might achieve greater precision in structured environments where precise element identification is crucial.

5.3 Error Analysis

TODO

6 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.

References

[1] Marah Abdin, Jyoti Aneja, Hany Awadalla, Ahmed Awadallah, Ammar Ahmad Awan, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Jianmin Bao, Harkirat Behl, Alon Benhaim, Misha Bilenko, Johan Bjorck, Sébastien Bubeck, Martin Cai, Qin Cai, Vishrav Chaudhary, Dong Chen, Dongdong Chen, Weizhu Chen, Yen-Chun Chen, Yi-Ling Chen, Hao Cheng, Parul Chopra, Xiyang Dai, Matthew Dixon, Ronen Eldan, Victor Fragoso, Jianfeng Gao, Mei Gao, Min Gao, Amit Garg, Allie Del Giorno, Abhishek Goswami, Suriya Gunasekar, Emman Haider, Junheng Hao, Russell J. Hewett, Wenxiang Hu, Jamie Huynh, Dan Iter, Sam Ade Jacobs, Mojan Javaheripi, Xin Jin, Nikos Karampatziakis, Piero Kauffmann, Mahoud Khademi, Dongwoo Kim, Young Jin Kim, Lev Kurilenko, James R. Lee, Yin Tat Lee, Yuanzhi Li, Yunsheng Li, Chen Liang, Lars Liden, Xihui Lin, Zeqi Lin, Ce Liu, Liyuan Liu, Mengchen Liu, Weishung Liu, Xiaodong Liu, Chong Luo, Piyush Madan, Ali Mahmoudzadeh, David Majercak, Matt Mazzola, Caio César Teodoro Mendes, Arindam Mitra, Hardik Modi, Anh Nguyen, Brandon Norick, Barun Patra, Daniel Perez-Becker, Thomas Portet, Reid Pryzant, Heyang Qin, Marko Radmilac, Liliang Ren, Gustavo de Rosa, Corby Rosset, Sambudha Roy, Olatunji Ruwase, Olli Saarikivi, Amin Saied, Adil Salim, Michael Santacroce, Shital Shah, Ning Shang, Hiteshi Sharma, Yelong Shen, Swadheen Shukla, Xia Song, Masahiro Tanaka, Andrea Tupini, Praneetha Vaddamanu, Chunyu Wang, Guanhua Wang, Lijuan Wang, Shuohang Wang, Xin Wang, Yu Wang, Rachel Ward, Wen Wen, Philipp Witte, Haiping Wu, Xiaoxia Wu, Michael Wyatt, Bin Xiao, Can Xu, Jiahang Xu, Weijian Xu, Jilong Xue, Sonali Yadav, Fan Yang, Jianwei Yang, Yifan Yang, Ziyi Yang, Donghan Yu, Lu Yuan, Chenruidong Zhang, Cyril Zhang, Jianwen Zhang, Li Lyna Zhang, Yi Zhang, Yue Zhang, Yunan Zhang, and Xiren Zhou. Phi-3 technical report: A highly capable language model locally on your phone, 2024.

[2] 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.

[3] 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.

[4] 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.

[5] Renu Patil and Rohini Temkar. Intelligent testing tool: selenium web driver. *International Research Journal of Engineering and Technology (IRJET)*, 4(06), 2017.

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

[7] wilsonzlin. minify-html, 2020.

Appendix

A Information Extraction Prompt

The prompt for the Information Extraction stage in the following format:

```
Extract the information and cues from the given context that are required by the
    question, and present the results in JSON format. When presenting the results,
    ensure character-level consistency with the extracted text, do not make any
    modifications. The given context may be incomplete or may not have the answer, so,
    the final JSON conclusion must not include any fields that have not been mentioned.

When there are multiple similar expressions, prioritize them according to the
    following rules (from high to low):
1. The label of the target text is more important in HTML semantics.
2. The target text is completely within a tag, rather than within a sentence or
    paragraph.
3. The target text is closer to other fields to be extracted.
4. If these expressions can complement each other, please extract them all.

Cue Text (cue_text, from high to low):
1. Cue Text: In HTML, the indicative text that signals the upcoming extraction of the
    target text, such as ‘Phone number‘ or ‘Address:‘.
2. When there is no cue text, use an empty string.

# Question:
{{ query }}

# Context:
‘‘‘html
{{ context }}
‘‘‘

# Answer Format (ignore the format requirements in the ‘Question‘, strictly follow the
     answer format of cue_text and value):
Thought: ...(Your thoughts, about fields mentioned in the context and their cues)...
Conclusion: ...(Strictly follow the JSON example format to answer: {{ json_example }})
    ...

# Your Answer:
```

B Program XPath Prompt

The system prompt for the Program XPath stage in the following format:

```
You are a pro software engineer, your task is reading the HTML code that user sent,
    and then response **one** Xpath (wrapped in JSON) that can recognize the element in
     the HTML to extract ‘target value‘.

Here’re some hints:
1. Do not output the xpath with exact value or element appears in the HTML.
2. Reference to the ‘target value‘ and the generated the xpath (if exists) to get more
     context.
3. When using text predication, always using ‘contains(., ’value’)‘ instead of ‘text()
    =’value’‘.
4. If the target xpath ends with ‘text()[n]‘, where n is not 1, please do not ignore
    it.
5. If cue text exist, using cue text and cue xpath to compose a new xpath might be a
    better idea.
6. String functions are allowed, such as ‘starts-with()‘, ‘ends-with()‘, ‘substring-
    before()‘, ‘substring-after()‘. Use it in caution, since it can only be used on ‘
    text()‘ node.

Please always response in the following Json format:
{
  "thought": "", # a brief thought of how to confirm the value and generate the xpath
  "xpath": "" # a workable xpath to extract the value in the HTML
}
```

The feedback prompt for the Program XPath stage in the following format:

```
Following the feedbacks to refine the xpath you provided:

1. Extend the xpath to include the missing information if ‘Missing‘.
2. Restrict the xpath to exclude the irrelevant information if ‘Surplus‘.
3. Correct the xpath grammer if ‘Invalid‘.
4. Response same xpath if no better solution.

{% for feedback in feedbacks %}
#### Evaluated on Fragment {{ feedback.id }}:
Extracted (JSON encoded): ‘{{ feedback.extracted | tojson }}‘
Feedback Message: ‘{{ feedback.message }}‘
{%- endfor %}
```