WIERT: Web Information Extraction via Render Tree

Zimeng Li[1]*, Bo Shao[2], Linjun Shou[2], Ming Gong[2], Gen Li[2], Daxin Jiang[2]†

[1]School of Computer Science and Engineering, Beihang University, Beijing, China
[2]Microsoft STCA
zimengli@buaa.edu.cn, {bo.shao,lisho,migon,geli,djiang}@microsoft.com

Abstract

Web information extraction (WIE) is a fundamental problem in web document understanding, with a significant impact on various applications. Visual information plays a crucial role in WIE tasks as the nodes containing relevant information are often visually distinct, such as being in a larger font size or having a brighter color, from the other nodes. However, rendering visual information of a web page can be computationally expensive. Previous works have mainly focused on the Document Object Model (DOM) tree, which lacks visual information. To efficiently exploit visual information, we propose leveraging the render tree, which combines the DOM tree and Cascading Style Sheets Object Model (CSSOM) tree, and contains not only content and layout information but also rich visual information at a little additional acquisition cost compared to the DOM tree. In this paper, we present WIERT, a method that effectively utilizes the render tree of a web page based on a pretrained language model. We evaluate WIERT on the Klarna product page dataset, a manually labeled dataset of renderable e-commerce web pages, demonstrating its effectiveness and robustness.

(a) A web page segment rendered with CSS.

American Beauty (1999)

Directed by Sam Mendes

Genres - Comedy Drama | Sub-Genres - Tragi-comedy , Coming-of-Age , Family Drama | Release Date - Sep 15, 1999 (USA) | Run Time - 121 min. | Countries - United States | MPAA Rating - R

- AllMovie Rating
  9
- User Ratings (714)
- Your Rating

(b) The same one rendered without CSS.

Figure 1: Comparison between rendering with CSS files and without CSS files. The web page segment is selected from the WebSRC dataset(Chen et al. 2021b).

Introduction

The rapid development of the internet has resulted in the creation of an enormous amount of information in the form of web pages. Extracting information from web pages is a long-standing problem in web document understanding(Hao et al. 2011; Lin et al. 2020; Wang et al. 2022; Hwang et al. 2020) and plays an important role in various fields such as knowledge base construction(Lockard, Shiralkar, and Dong 2019; Xie et al. 2021), retrieval systems(Zhang and Vines 2004), question answering(Chen et al. 2021a), recommender systems(Leksin and Nikolenko 2013) and self-driving web browsers(Iki and Aizawa 2022).

A typical web page consists of various web documents, such as HTML, CSS, JavaScript, and resource files like images. Each of these documents has different formats and serves distinct roles in determining the overall semantics of the web page. CSS files contribute to the rendering process and appearance of a web page, and their absence can lead to difficulties in web page understanding.

Web page information extraction (WIE) involves extracting informative attributes from a given web page, which can be predetermined or unspecified. Attribute extraction, which involves extracting information such as university names, addresses, and phone numbers from university introduction pages, is a common form of WIE. Unlike information extraction from plain text documents, WIE involves understanding the complex layout and semantic information of web pages. This paper focuses on addressing the problem of attribute extraction from web pages.

Previous works have attempted to solve the WIE problem, with wrapper induction and its variants being classical methods for this task. However, these methods are difficult to generalize to unseen domains and are sensitive to page layout changes, limiting their performance on complex WIE tasks due to the small scale of annotated labeled data. Recent deep learning-based methods have significantly improved WIE performance, leveraging pre-trained language models from large scale unsupervised data to model DOM

*Work done when the first author was an intern at Microsoft STCA.
†Corresponding author.

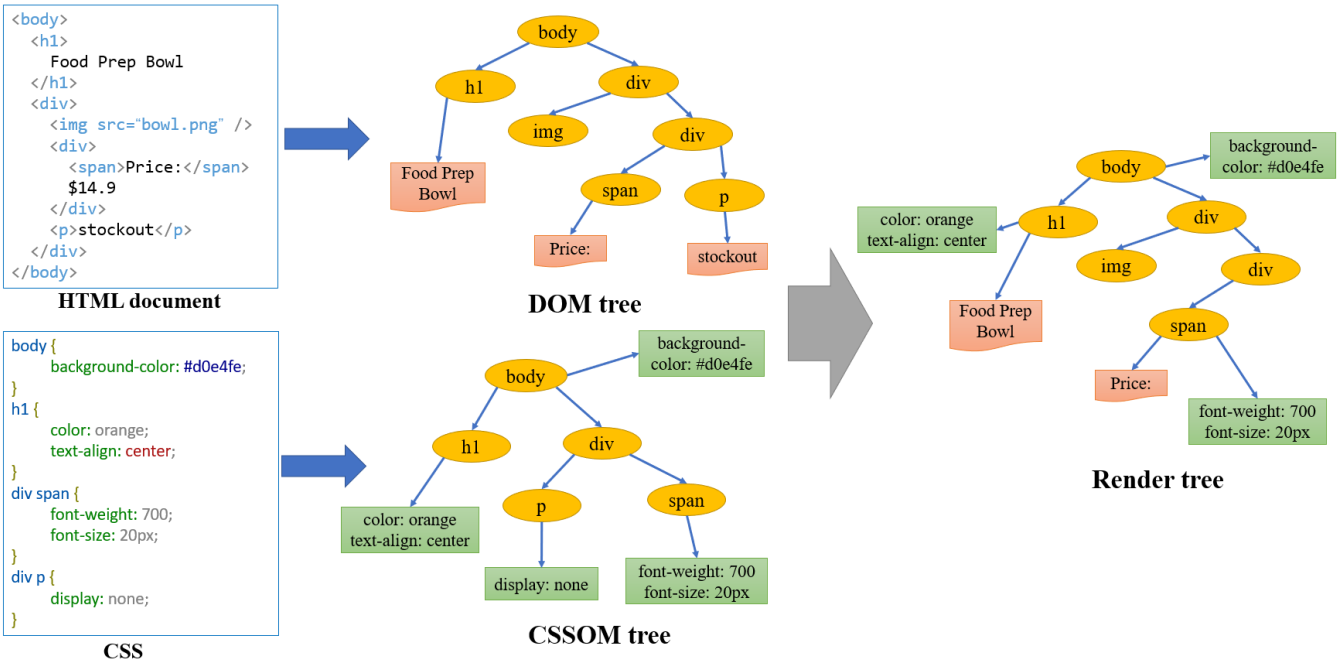


Figure 2: The generative process of render tree. The yellow ellipses represent element nodes. The pink trapezoids with curve side represent text nodes. The green squares represent style sheets. The blue thin arrows represent associations between element nodes and text nodes or style sheets.

tree structure information(Li et al. 2022; Deng et al. 2022; Wang et al. 2022).

Nevertheless, several challenges still exist in WIE. HTML documents of web pages are too large to be fed into deep learning models due to computational resource limitations. Previous transformer-based works used BERT or RoBERTa as backbones, which have a maximum input length of 512 tokens(Li et al. 2022; Deng et al. 2022). Since the number of tokens in HTML documents typically exceeds ten thousand, previous works split the documents into segments, leading to a loss of global semantic modeling. Additionally, previous works have largely ignored other useful information in web pages such as CSS styles, and have also struggled with the problem of sparse labeling.

To address these challenges, we propose a new method for WIE called WIERT (Web Information Extraction via Render Tree). This approach utilizes the render tree of a web page based on a language model pre-trained on long documents. We leverage the render tree, which is the combination of the DOM tree and the Cascading Style Sheets Object Model (CSSOM) tree, to take advantage of visual information efficiently. The render tree contains content, layout information, and rich visual information with little additional acquisition cost compared to the DOM tree, making it an effective solution for WIE.

In summary, this paper makes the following contributions:

1. We propose a novel transformer-based model, WIERT, for web page information extraction that effectively encodes the semantic and visual information of web documents via the render tree.
2. To improve the representation of web pages, we use CSS style embedding to encode rich visual information in web pages and introduce three training strategies: informative node tagging, relationship with informative node prediction, and token type classification.
3. Our experiments demonstrate the effectiveness and robustness of our method by achieving state-of-the-art results on the Klarna product information extraction task(Hotti et al. 2021).

Related Work

With the development of Internet, web data is exploding, leading that web page information extraction is a hot issue now. Many works try to solve web page information extraction problems. Wrapper induction and its variants are the main classical methods(Arjona Fernández et al. 2007; Chang et al. 2006; Kushmerick, Weld, and Doorenbos 1997; Laender et al. 2002; Turmo, Ageno, and Català 2006; Gregg and Walczak 2006). They summarize a set of extraction rules by analyze the structure of DOM trees, then apply this set of rules on other web pages directly.

Classical WIE methods focus on mining templates of web pages, which only involve shallow semantics. In order to conduct information extraction from the perspective of deep semantics, Modern WIE methods are based on deep learning techniques, which has been proven can understand patterns of big data. According to the backbone network architectures, modern WIE methods could be divided into two categories: graph neural network based methods(Hwang et al.

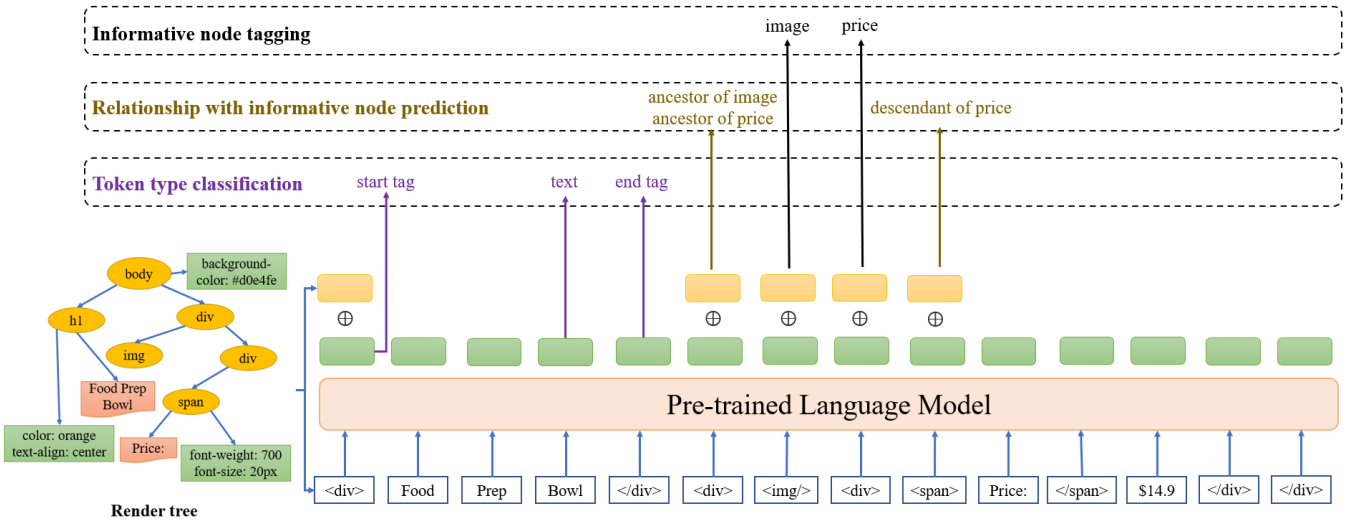


Figure 3: The overall model architecture of our proposed WIERT. The green rectangles with rounded corners represent output vectors of the pre-trained language model. The yellow rectangles with rounded corners represent style embedding vectors. $\oplus$ is vector concatenation operator. The top three longitudinal bars represent training objectives.

2020; Kumar et al. 2021) and transformer based methods(Li et al. 2022; Deng et al. 2022; Wang et al. 2022).

Graph neural network based methods regard the DOM tree as a graph(Hwang et al. 2020; Kumar et al. 2021). The DOM elements are nodes and the parent-child relationships are edges. Most works add edges between siblings nodes to indicate the tree structure. Then a graph neural network is used to generate the hidden representations for each element node, which are fed into a classifier to judge whether the node contains key information.

Transformer-based methods first serialize a HTML document as a sequence, then use a transformer model to process it. To let models learn the DOM tree structure of web documents, transformer based methods usually design various modules to encode DOM trees. MarkupLM(Li et al. 2022) adds a xpath embedding layer at the bottom of language model to encode the DOM structure as fixed length vector. DOM-LM(Deng et al. 2022) utilizes multiple embedding layers (i.e., depth embedding) to represent the DOM structure. WebFormer(Wang et al. 2022) designs several DOM structure based attention mechanisms.

Preliminaries

DOM Tree, CSSOM Tree and Render Tree

A well-rendered webpage require a set of component files, including HTML, CSS, JS documents, and other resource files such as images. While receiving these raw files, a modern browser builds the DOM tree from HTML document and the CSSOM tree from CSS document. The latter encodes CSS rules into a tree structure. Then the browser combines them into a render tree to attach style sheets to the corresponding element nodes. The style sheets on render tree are computed, which means that these are the final styles for element nodes. This generative process is shown in Fig 2. After that, the browser computes coordinates for each node and paints pixel on the screen. It is worth noting that this final step is only directly related to the render tree.

The structure of the render tree is similar to that of the DOM tree, except that each element node in the render tree has a style sheet attached to it. The render tree provides more detailed visual information than the DOM tree, making it crucial for web information extraction tasks and closer to the essence of the web page. Additionally, the render tree only includes element nodes that are displayed on the web page, unlike the DOM tree, which may contain element nodes that are not displayed. This means that the render tree is smaller than the DOM tree, making it more efficient to encode. Because of these reasons, we have chosen to use the render tree for solving WIE problems.

WIE Problem Definition

In this study, we approach the problem of extracting information from web pages by framing it as a node tagging task for the render tree elements. First, we obtain the render tree of a web page, which we serialize to a sequence through a preorder traversal. This serialized render tree comprises text and element nodes, and we subsequently transform it into a token sequence that can be input into a pretrained language model that we have modified for this purpose. The process involves removing redundant spaces and tokenizing the content of text nodes and adding them to the token sequence. For element nodes, we add start and end tag tokens if the tag is not self-closing or a corresponding self-closing tag token if it is self-closing.

The resulting token sequence, denoted as $[t_1, t_2, \ldots, t_L]$, comprises four types of tokens: text tokens, start tag tokens, end tag tokens, and self-closing tag tokens. We anticipate that the self-closing and start tag tokens will effectively summarize the information contained in their respective element nodes and serve as proxies for web page information extrac-

tion after encoding by a language model.

Render Tree Encoder

Fig 3 shows the overall architecture of our proposed WIERT model, a render tree encoder initialized by a transformer-based pre-trained language model. First the render tree is serialized into a sequence composed of text tokens and tag tokens by preorder traversal. Then we feed the sequence into a transformer encoder and get the sequence of output vectors. On the other side, the style embedding module convert style sheets of element nodes into fixed length vectors. For each element node, we concatenate its corresponding encoder output and style embedding vector to generate the final render tree element node representation.

In this sector, we explain how to modify a pre-trained language model to a render tree encoder.

Tag Token Embedding

Since pretrained language models are trained on plain text documents, their word embedding layer cannot encode tag tokens. Here we add a new embedding layer to encode tag tokens. Because tag names are usually words with clear semantics, we can initialize the tag token embedding layer with word embeddings of tag names.

Formally, the input embeddings of the token sequence $[t_1, t_2, \ldots, t_L]$ is processed as

$$w_i = \begin{cases} e_w(t_i) & t_i \; is \; a \; text \; token \\ e_t(t_i) & t_i \; is \; a \; tag \; token \end{cases}$$

where w_i is the input embedding to the transformer encoder of the i-th token, e_w is original word embedding layer of the pretrained model and e_t is our additional tag token embedding layer. Then the embedding sequence $[w_1, w_2, \ldots, w_L]$ is fed into the encoder and the output sequence $[h_1, h_2 \ldots, h_L]$ are generated.

Style Embedding

When accessing a web page, its HTML and CSS source code are not typically read. Rather, the visual content rendered from the web documents by a browser is examined. However, rendering images of web pages can be computationally intensive and resource-consuming. Consequently, we introduce a cheaper alternative to images of web pages, namely style sheets applied to the render tree.

A style sheet comprises a set of style property-value pairs as shown in Figure 2. Each style property can impact the appearance of element nodes in various ways, such as text size, text weight, and background color.

For each element node in a render tree, its corresponding style sheet is represented as $k_i : v_i{}_{i=1}^{m}$, where m is browser-specific. For each style property, a style embedding layer embeds v_i into a vector. The resulting vectors are concatenated together to create the style embedding of the element node:

$$s := [e_{s_1}(v_1) : \ldots : e_{s_m}(v_m)]$$

where s is an abbreviation for style embedding, : denotes the concatenation operator, and e_{s_i} is the embedding mapping of the i-th style property for $i = 1, \ldots, m$. The style embedding and the encoder output of the i-th element node, namely s_i and h_i, respectively, are concatenated to facilitate web page information extraction.

Model Training

Informative Node Tagging

Since we model the information extraction problem as render tree element node tagging problem, the output representations of start tag token and self-closing token are considered to represent the whole corresponding subtree. The index set $I = \{k_1, k_2, \ldots, k_m\}$ refers to indices of start tag tokens and self-closing tag tokens. We train the model using node-wise cross entropy loss:

$$l_{INT} = \sum_{i \in I} ce(\text{MLP}([h_i : s_i]), y_i)$$

where the function ce means cross entropy function, and y_i is the label of the element node identified by the token i, taking value from $\{1, 2, \ldots, C, C+1\}$. The positive integer C is the number of information types. We add a class called $C+1$ to represent that a node doesn't express any valuable information of interest.

Relationship with Informative Node Prediction

The training objective INT focuses solely on candidate tokens, specifically start tag tokens and self-closing tag tokens. As a result, error signals cannot propagate through other tokens, which may hinder the efficiency of the training process. In other words, the main training objective fails to teach the model to learn the structure of information. To address this issue, the traditional solution involves adding a random conditional field on top of the backbone model.

To overcome this limitation, we have implemented a classifier for each attribute to determine the relationship between an element node and the node with the corresponding attribute. These relationships can be categorized into four types: ancestor, precedent, self, and other. The loss function is

$$l_{RINP} = \sum_{i \in I} \sum_{j=1}^{C} ce(\text{MLP}([h_i : s_i], y_{ij})$$

where y_{ij} is the label of token i on label j, taking value from $\{self, ancestor, descendant, other\}$.

Token Type Classification

All input tokens could be divided into four classes, including text tokens, start tokens, end tokens and self-closing tokens. Tokens of different classes act different roles. Text tokens is responsible for the formation of text content in web page. HTML tag tokens are responsible for type setting, building non-textual content such as images and giving web pages more fruitful semantic information.

The token type classification task is viewed as four-class classification task. The loss functions of token type classification task is

$$l_{TTC} = \sum_{i=1}^{L} ce(\text{MLP}([h_i : s_i]), y_i)$$

where y_i is the token type of the i-th token, ce represents the cross entropy loss.

Finally, we train the following loss

$$l = \lambda_1 l_{INT} + \lambda_2 l_{RINP} + \lambda_3 l_{TTC}$$

where positive numbers λ_1, λ_2 and λ_3 are weight factors to leverage the influence from three training objectives, which can vary as training process go.

Experiments

Dataset

Klarna product page dataset The Klarna product page dataset contains 51,701 manually labeled product pages from 8,175 real e-commerce websites(Hotti et al. 2021). In each web page, at most 5 kinds of information are annotated and each kind is only attached to at most one DOM node.

The Klarna dataset is multilingual. In order to utilize pretrained language models (e.g. BERT, BigBird), only English subsets are used in our experiments. Tab 2 show the scale of English part of the Klarna dataset. As we can see, the Klarna dataset provided a official train/test split. In our experiments, we keep the official test set to measure generalization performance and split the official train set into a new train set and a validation set without overlapping according to the ratio of $9:1$.

Competitive Methods

We compare the results of our proposed WIERT and its variants with some strong baseline methods from (Hotti et al. 2021):

FreeDOM-EXT FreeDOM-EXT adapts the state of the art FreeDOM architecture(Lin et al. 2020) by adding some style features. FreeDOM is a two-stage method where a neural network first computes node embeddings based on local and contextual information, and then uses embedding distances and semantic relatedness between node pairs in the second stage.

TransformerEncoder consists of a single-layer multi-headed attention encoder stack fed with a sequence consisting of the features of the local node stacked with the features of its neighbors.

GCN-mean is the best baseline model, which is a multi-layer GCN model similar to GraphSage(Hamilton, Ying, and Leskovec 2017). GCN-mean models a DOM tree as a graph, each element is a node and the parent-son relationship on the tree is edge. GCN-mean apply average pooling to integrate information from neighbor nodes.

GCN-GRU is inspired by the local embedding module in the DOM-Q-Net algorithm(Jia, Kiros, and Ba 2019). The encoding of node is computed by feeding a Gated Recurrent Unit (GRU) with the local features of the node and an average encoding of the neighborhood of the node.

LSTM-BU A child-sum tree-LSTM model. For each element node, its hidden state is the sum of the hidden states of the children, and the cell state is computed based on the cell states of the children.

Besides baselines from (Hotti et al. 2021), we introduce a strong baseline model which is well known and behaves well in various web document understanding tasks recently:

MarkupLM encodes the Xpath of each DOM node and add the Xpath embeddings to the word embeddings, then fed into a pretrained language model. Because Markup LM is designed for modelling textual contents of HTML documents, we only use it to recognize two textual informative fields: name and price.

Last we also compare our WIERT with a baseline model generated from itself:

Baseline use the same model architecture as WIERT, but is only trained on informative node tagging task, i.e. $\lambda_1 = 1$ and $\lambda_2 = \lambda_3 = 0$, and doesn't contain style embedding layer at the top.

Implementation Detail

We use a pretrained Big Bird model to initialize our WIERT model, which was pretrained on a large corpus of long plain documents with a maximum input length of 4096(Zaheer et al. 2020). To handle discrete properties, we map each value to a one-hot vector. For continuous properties, we discretize the values into intervals of $(-\infty, a_1], \ldots, (a_n, +\infty)$ and treat them as discrete properties. For color properties, which are represented by four continuous values (i.e., red, green, blue, and alpha), we treat them as four continuous properties. We introduce only a few parameters in the embedding layers, and each style property is encoded by a style embedding layer. The details of the used style properties are listed in Table 1.

Usually the token sequences generated from render trees are too long to be fed into model. To address this issue, we adopt a simple method of splitting an entire token sequence into several adjacent but disjoint segments with lengths less than model's limitation. During both the training and testing stages, we use this method to produce token sequences from the render tree structure. For all experiments, we set the batch size to 16 and use an initial learning rate of 5×10^{-5}, which decays to 85% after each epoch. Through coarse hyperparameter tuning, we set the weights of three losses as $\lambda_1 = 1, \lambda_2 = 0.2, \lambda = 0.1$.

All experiments are conducted on eight V100 GPUs. When training WIERT and its variants, we first train the newly added parameters, which are randomly initialized, for up to two epochs. Then, we train all parameters for up to 30 epochs. The training results with the minimum error rate on the validation set are chosen to report.

Results and Discussion

Performance Comparison Table 3 shows the results of WIERT and competitive methods. We use predictive accuracy of each attribute and average of them as evaluation metrics. The predictive accuracy is the quotient of the number of correctly predicted pages over the number of all pages. As we can see, WIERT achieves the best performance among other baselines.

Compare to the baseline, we observe that WIERT shows significantly superior, which illustrate the effectiveness to model render tree of our propose strategies. Compared to MarkupLM and other non-pretrained methods, such as FREEDOM-EXT, GCN-*, LSTM-BU, Our models achieve over 10% improvement, which shows the contribution of the

style name	explanation
display	the display behavior (the type of rendering box) of an element
visibility	whether or not an element is visible
font-size	the size of a font
font-weight	how thick or thin characters in text should be displayed
font-style	the font style for a text
color	the color of text
background-color	the background color of an element
background-clip	how far the background (color or image) should extend within an element
opacity	the opacity level for an element
text-align	the horizontal alignment of text in an element
direction	the text direction/writing direction within a block-level element
text-decoration-line	the kind of text decoration to use (like underline, overline, line-through)
text-decoration-style	the style of the text decoration (like solid, wavy, dotted, dashed, double)
float	whether an element should float to the left, right, or not at all
outline-color	the color of an outline
outline-width	the width of an outline
outline-style	the style of an outline
caption-side	the placement of a table caption
border-collapse	whether table borders should collapse into a single border or be separated
border-width	the width of an element's border
box-sizing	how the width and height of an element are calculated
resize	if (and how) an element is resizable by the user

Table 1: Style properties used in experiments.

	Train	Test
US	7,687	3,316
GB	7,741	3,403
Total	15,428	6,719

Table 2: The number of pages in the English part of the Klarna dataset.

pretrained model and furthermore, our methods effectively adapt language model to web page information extraction task.

Ablation Study The ablation study was conducted to evaluate the effectiveness of different strategies in improving the performance of the proposed WIERT model. As shown in Table 4, we analyzed the contributions of style features and training objectives to the overall accuracy of the model.

Firstly, we investigated the impact of CSS style features on the performance of the model by removing them from the input. The results showed a decrease in the average accuracy of 0.007, indicating that CSS information plays an important role in the identification of certain types of entities, such as "buy," "image," and "price," which contain rich visual information.

Secondly, we examined the effect of the relationship with informative node prediction (RINP) task on the model's accuracy. The results showed a more significant decrease in the average accuracy of 0.013, suggesting that the structure of the DOM tree is crucial for identifying most of the attributes from the candidates within a page.

Finally, we analyzed the impact of the token type classification (TTC) task on the model's accuracy by removing it from the training objectives. The results indicated a decrease in the average accuracy of 0.007, which demonstrates the effectiveness of the TTC task in improving the performance of the model.

Overall, the ablation study highlights the importance of style features, the RINP task, and the TTC task in enhancing the performance of the WIERT model for web information extraction tasks.

Case Study Key information on a web page often uses eye-catching style, i.e. larger font sizes, bold font weights, or bright font colors, to grab readers' attention. In this section, we have randomly selected a few web pages where key information is presented with significant style features. We will compare the performance of two models: one trained with style features and the other without.

In Figure 4, we compare the results of the same web page predicted by both models. We use a red bounding box to highlight the predicted product price attribute. The model trained without style features incorrectly marks an element node, while the model trained with style features correctly marks the intended node. It is evident that the product price is emphasized using a larger font size and brighter font color, which are captured by the style encoding module. Consequently, the model trained with style features is able to easily identify the correct element node.

Conclusion

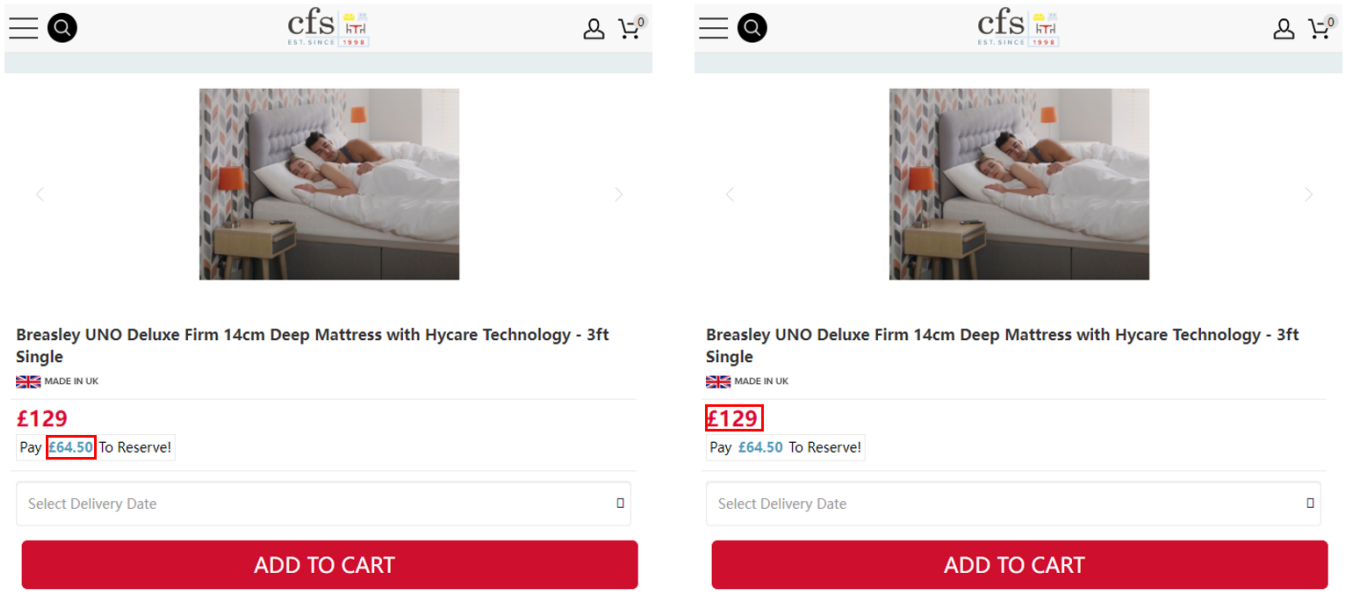
Web information extraction is a crucial task in web document understanding with numerous commercial applications. In order to effectively understand web content, visual information is essential. However, previous approaches have

method	buy	cart	image	name	price	avg
FREEDOM-EXT	.840	.488	.343	.835	.728	.667
Transformer encoder	.814	.495	.369	.787	.611	.610
GCN-mean	.911	.671	.497	.811	.653	.709
GCN-GRU	.837	.620	.373	.791	.607	.646
LSTM-BU	.897	.645	.423	.733	.563	.652
Markup LM	-	-	-	.664	.696	.680
Baseline	.941	**.786**	.556	.887	.926	.819
WIERT	**.952**	.780	**.574**	**.889**	**.934**	**.826**

Table 3: Predictive accuracy on the Klarna product dataset. The columns with best performance are bold.

TTC	RINP	CSS	buy	cart	image	name	price	avg
✓	✓	✓	**.952**	.780	**.574**	**.889**	**.934**	**.826**
✓	✓		.941	.786	.556	.887	.926	.819
✓		✓	.937	.773	.550	.887	.918	.813
	✓	✓	.949	**.787**	.551	**.889**	.919	.819

Table 4: Ablation study on the Klarna product dataset. "TTC" means token type classification. "RINP" means relationship with informative node prediction. "CSS" means CSS style embeddings.



(a) Extracted by model trained without CSS styles

(b) Extracted by model trained with CSS styles

Figure 4: A web page in the klarna product dataset. The red bounding boxes mark the product name attribute generated by model without style features (left) and model with style features (right).

mainly focused on modeling the DOM tree, which lacks significant visual information.

In this paper, we propose WIERT, an effective method for web information extraction that leverages the render tree of a web page. The render tree is a combination of the DOM tree and the CSSOM tree, and contains not only content and layout information but also rich visual information. This approach comes with only a small additional acquisition cost compared to DOM tree based methods, but provides substantial benefits in terms of efficiency and effectiveness.

Our experiments demonstrate that the visual information contained in render trees significantly improves the performance of web information extraction tasks. WIERT provides an innovative solution for utilizing the visual information of web pages and can be applied to a variety of web understanding tasks.

References

Arjona Fernández, J. L.; Corchuelo Gil, R.; Ruiz Cortés, D.; and Toro Bonilla, M. 2007. From Wrapping to Knowledge. *IEEE Transactions on Knowledge and Data Engineering*,

19(2): 310–323.

Chang, C.-H.; Kayed, M.; Girgis, M. R.; and Shaalan, K. F. 2006. A survey of web information extraction systems. *IEEE transactions on knowledge and data engineering*, 18(10): 1411–1428.

Chen, Q.; Lamoreaux, A.; Wang, X.; Durrett, G.; Bastani, O.; and Dillig, I. 2021a. Web question answering with neurosymbolic program synthesis. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, 328–343.

Chen, X.; Zhao, Z.; Chen, L.; Ji, J.; Zhang, D.; Luo, A.; Xiong, Y.; and Yu, K. 2021b. WebSRC: A Dataset for Web-Based Structural Reading Comprehension. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021*, 4173–4185. Association for Computational Linguistics.

Deng, X.; Shiralkar, P.; Lockard, C.; Huang, B.; and Sun, H. 2022. DOM-LM: Learning Generalizable Representations for HTML Documents. *CoRR*, abs/2201.10608.

Gregg, D. G.; and Walczak, S. 2006. Adaptive web information extraction. *Commun. ACM*, 49(5): 78–84.

Hamilton, W.; Ying, Z.; and Leskovec, J. 2017. Inductive representation learning on large graphs. In *Advances in neural information processing systems*.

Hao, Q.; Cai, R.; Pang, Y.; and Zhang, L. 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*, 775–784.

Hotti, A.; Risuleo, R. S.; Magureanu, S.; Moradi, A.; and Lagergren, J. 2021. The Klarna Product Page Dataset: A Realistic Benchmark for Web Representation Learning. *arXiv preprint arXiv:2111.02168*.

Hwang, W.; Yim, J.; Park, S.; Yang, S.; and Seo, M. 2020. Spatial dependency parsing for semi-structured document information extraction. *arXiv preprint arXiv:2005.00642*.

Iki, T.; and Aizawa, A. 2022. Do BERTs Learn to Use Browser User Interface? Exploring Multi-Step Tasks with Unified Vision-and-Language BERTs. *CoRR*, abs/2203.07828.

Jia, S.; Kiros, J.; and Ba, J. 2019. DOM-Q-NET: Grounded RL on Structured Language. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Kumar, A.; Morabia, K.; Wang, J.; Chang, K. C.; and Schwing, A. G. 2021. CoVA: Context-aware Visual Attention for Webpage Information Extraction. *CoRR*, abs/2110.12320.

Kushmerick, N.; Weld, D. S.; and Doorenbos, R. B. 1997. Wrapper Induction for Information Extraction. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence, IJCAI 97, Nagoya, Japan, August 23-29, 1997, 2 Volumes*, 729–737. Morgan Kaufmann.

Laender, A. H. F.; Ribeiro-Neto, B. A.; da Silva, A. S.; and Teixeira, J. S. 2002. A Brief Survey of Web Data Extraction Tools. *SIGMOD Rec.*, 31(2): 84–93.

Leksin, V. A.; and Nikolenko, S. I. 2013. Semi-supervised tag extraction in a web recommender system. In *International Conference on Similarity Search and Applications*, 206–212. Springer.

Li, J.; Xu, Y.; Cui, L.; and Wei, F. 2022. MarkupLM: Pre-training of Text and Markup Language for Visually Rich Document Understanding. In Muresan, S.; Nakov, P.; and Villavicencio, A., eds., *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics, ACL 2022*, 6078–6087. Association for Computational Linguistics.

Lin, B. Y.; Sheng, Y.; Vo, N.; and Tata, S. 2020. FreeDOM: A Transferable Neural Architecture for Structured Information Extraction on Web Documents. In *The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 1092–1102. ACM.

Lockard, C.; Shiralkar, P.; and Dong, X. L. 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*, 3047–3056.

Turmo, J.; Ageno, A.; and Català, N. 2006. Adaptive information extraction. *ACM Comput. Surv.*, 38(2): 4.

Wang, Q.; Fang, Y.; Ravula, A.; Feng, F.; Quan, X.; and Liu, D. 2022. WebFormer: The Web-page Transformer for Structure Information Extraction. In *Proceedings of the ACM Web Conference 2022*, 3124–3133.

Xie, C.; Huang, W.; Liang, J.; Huang, C.; and Xiao, Y. 2021. WebKE: Knowledge Extraction from Semi-structured Web with Pre-trained Markup Language Model. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, 2211–2220.

Zaheer, M.; Guruganesh, G.; Dubey, K. A.; Ainslie, J.; Alberti, C.; Ontañón, S.; Pham, P.; Ravula, A.; Wang, Q.; Yang, L.; and Ahmed, A. 2020. Big Bird: Transformers for Longer Sequences. In *Advances in Neural Information Processing Systems, NeurIPS 2020*.

Zhang, Y.; and Vines, P. 2004. Using the web for automated translation extraction in cross-language information retrieval. In *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, 162–169.