Understanding HTML with Large Language Models

Izzeddin Gur, Ofir Nachum, Yingjie Miao, Mustafa Safdari, Austin Huang
Aakanksha Chowdhery, Sharan Narang, Noah Fiedel, Aleksandra Faust
Google DeepMind
{izzeddin,ofirnachum,yingjiemiao,msafdari,austinvhuang
chowdhery,sharannarang,nfiedel,faust}@google.com

Abstract

Large language models (LLMs) have shown exceptional performance on a variety of natural language tasks. Yet, their capabilities for HTML understanding – i.e., parsing the raw HTML of a webpage, with applications to automation of web-based tasks, crawling, and browser-assisted retrieval – have not been fully explored. We contribute HTML understanding models (fine-tuned LLMs) and an in-depth analysis of their capabilities under three tasks: (i) Semantic Classification of HTML elements, (ii) Description Generation for HTML inputs, and (iii) Autonomous Web Navigation of HTML pages. While previous work has developed dedicated architectures and training procedures for HTML understanding, we show that LLMs pretrained on standard natural language corpora transfer remarkably well to HTML understanding tasks. For instance, when fine-tuned on data from the MiniWoB benchmark, LLMs successfully complete 50% more tasks using 192x less data compared to the previous best supervised model. We create and open-source a large-scale HTML dataset distilled and auto-labeled from CommonCrawl.

1 Introduction

Web crawling (Olston et al., 2010), form-filling (Diaz et al., 2013; Gur et al., 2021), or information retrieving web agents (Nogueira and Cho, 2016) are important for both automating and assisting users in web-based tasks. These and similar applications rely on models that can search for specific content or controls on a web page as well as navigate a website autonomously. Since a web page in its raw form is represented as an HTML-based text sequence, the success of models for web-based tasks relies on their ability to understand HTML semantics, structure, and embedded interactions.

The predominant approach to web automation and HTML understanding is to train specialized models, i.e., gathering application-specific datasets and designing neural network (NN) architectures to leverage inductive biases of the HTML’s structure; see, e.g., Liu et al. (2018); Toyama et al. (2021); Gur et al. (2019, 2021); Humphreys et al. (2022). However, both dataset collection and neural architecture design are expensive, time-consuming, and require highly-specialized, domain-specific knowledge.

Meanwhile, in the natural language processing (NLP) literature, large language models (LLMs) have emerged as a solution to the difficulties of dataset collection and specialized NN design (Kaplan et al., 2020; Bommasani et al., 2021). A popular paradigm in NLP is to take an off-the-shelf LLM – pretrained on a large text corpus via an unsupervised and task-agnostic learning objective – and either fine-tune or prompt the LLM on a small task-specific dataset. This paradigm has shown exceptional performance on a variety of NLP tasks (Xue et al., 2020; Brown et al., 2020; Austin et al., 2021). Whether LLMs can be applied to HTML understanding – especially given the much larger context and sequence lengths – remains an under-explored question.

In this paper, we investigate whether LLMs can be applied to HTML understanding to produce better-performing, more sample-efficient HTML understanding models and without the need for custom NN architecture design. To that end, we present a suite of three benchmarking tasks for HTML understanding that capture the essence of these applications and require understanding both structure and content. First, we devise Semantic Classification as a task that requires a model to classify a given HTML element into one of a set of categories, such as address, email, password etc., with application to automated form-filling. Second, we present Description Generation, a label-extraction

1See visualizations of the results at https://sites.google.com/view/llm4html/home.
task where a model is given an HTML snippet and is asked to produce a natural language description. For instance for an email field, the description might be “Please enter your email address.” Note that in the majority of web pages, this connection between input elements and description content is only implicit in the raw HTML code and inferring such links is a prerequisite for higher-level navigation objectives. The third task is Autonomous Web Navigation (Shi et al., 2017). A model is presented with an HTML page paired with a natural language command and must apply appropriate actions on a sequence of HTML pages to satisfy the command. See Figure 1 for a simplified example of these tasks.

With these benchmark tasks in hand, we evaluate the transfer capabilities of a variety of pre-trained LLMs (Table 1), varying in architecture (encoder-only, encoder-decoder, or decoder-only), model size (from 24.6M to 62B parameters), and training data corpora (both including and excluding pretraining NLP and HTML corpus). While prior work universally pre-parses the HTML as input to the model (Gur et al., 2021; Liu et al., 2018; Nakano et al., 2021), ours uses raw, unprocessed HTML. Our results show that LLMs demonstrate a remarkable level of HTML understanding across all tasks, with up to $192 \times$ more sample-efficiency than models trained from scratch, and achieving a new SoTA for supervised learning on the MiniWoB benchmark suite (Shi et al., 2017). The encoder-decoder architectures with bi-directional attention show the best performance across the board even when their pretraining does not include HTML.

The broader objective of this research is to advance the integration of LLMs with autonomous web agents. It has only been in the last year that researchers have begun to utilize LLMs outside of NLP and integrate them as core capabilities in autonomy ((Lu et al., 2021; Ahn et al., 2022)). In this context, LLMs are reasoning engines for sequential decision making agents interacting with environments. We believe these contributions expand the scope of language models and connect their unique capabilities with autonomous agents for the web. We provide a new perspective on machine learning for HTML understanding and web automation, showing that pretrained LLMs can achieve significant performance on such tasks, reducing the need for specialized architectures and training protocols. To encourage further research in this direction, we open sourced 2 model weights for agents used in the WoB environment and our dataset for description generation.

2 Related Work

HTML Understanding Autonomous web navigation has been a popular application for neural network models, and a variety of works propose simulated websites for training web-based agents, with application to task fulfillment (Yao et al., 2022; Gur et al., 2021; Burns et al., 2022; Mazumder and Riva, 2020; Shi et al., 2017; Liu et al., 2018) as well as information retrieval or question-answering (Adolphs et al., 2021; Nogueira and Cho, 2016). Simulated websites provide an easy way to evaluate models online, and for this reason we use the existing MiniWoB benchmark (Shi et al., 2017) for our web navigation setting. However, it is still important to have a mechanism for evaluating models on a wide variety of real-world websites. This was the key motivation for generating our own dataset for the description generation task, which is distilled and auto-labeled from CommonCrawl and is a key contribution of our paper.

Alongside these benchmarks, many works have developed models for web navigation and related subtasks (Pasupat et al., 2018; Bommasani et al., 2021; He et al., 2021; Gur et al., 2021; Humphreys 2

2https://console.cloud.google.com/storage/browser/gresearch/webllm
et al., 2022; Liu et al., 2018; Jia et al., 2019). These works often rely on specialized neural network architectures that leverage inductive biases of HTML structure, or on preprocessing of HTML to make it easier to input to a model ((Li et al., 2021a,b)). In contrast, our work takes a minimalist approach, providing HTML in text form with minimal processing and using widely-adopted transformer networks.

**LLMs and HTML** Works that explore the intersection of LLMs and HTML generally fall into two categories. The first category uses LLMs to assist web navigation (Nakano et al., 2021; Yao et al., 2022), and typically relies on a custom preprocessing to map the context and structure of a web page to natural language, thus severely restricting what HTML pages the model can parse. The second category pretrains LLMs on a large corpora of HTML text (Aghajanyan et al., 2021). However, these works typically restrict the model evaluation to standard NLP tasks, e.g., summarization and question/answering as opposed to tasks more relevant to HTML understanding and web automation. Our work can be thought of as the reverse: We keep the pretraining of LLMs unchanged and focus on the mechanisms for transferring the pretrained LLMs to HTML-relevant tasks.

### 3 Canonical Tasks for HTML Understanding

We devise three canonical tasks to study HTML understanding capabilities of LLM-based web agents. These tasks require correctly interpreting both structure and content to varying degrees to make predictions, with autonomous navigation being the most challenging capability of the three.

**Autonomous Web Navigation.** This task evaluates how well a model navigates multi-page websites as a sequential decision-making problem (Shi et al., 2017; Liu et al., 2018). At the beginning of an episode, the agent is given a natural language instruction, e.g. *Enter the username “lyda” and the password “N22t” into the text fields and press login.* The agent applies actions to a sequence of HTML pages, where each action is of the form `function(selector, text)`. The function is one of `click` or `type`, `selector` is an integer pointer that uniquely identifies an element, and `text` is a text to input if the `type` functionality is activated. An episode terminates when either the page reaches a terminal state (e.g., the ‘sign in’ button is clicked) or the maximum number of steps is reached.

**Semantic Classification.** Many HTML understanding applications require a model that can classify HTML elements into standardized categories. For example, in automated form-filling (Diaz et al., 2013; Gur et al., 2021), it is useful to identify a ‘submit button’ across many websites (e.g., shopping, flight booking, utility application) with various button representations (e.g., position, color, or text). Thus, we formulate Semantic Classification as classifying elements into role categories. Take the example HTML in Figure 1 which includes two `input` elements and a `submit` button. Let’s pick the first `input` as an element of interest to be classified by the system, also called a salient element. The system should classify this element as `username`, since it appears on a login page and it has a `label` with Email Address which is typically associated with the username in form-filling applications. To solve this, the system can aggregate information from multiple sources in the page – the label that says Enter Email Address, the `input` attributes (`type=“email”` and `id=“uName”`), or even the ordering of other elements in the page such as ‘password’ and ‘sign in’.

**Description Generation.** Motivated by applications in accessibility-minded web browser control (Jorgensen and Binsted, 2005), we formulate description generation as an extractive problem where the goal is to locate the textual description of an element in the HTML and generate it as output. For instance, the description of the salient element in Figure 1 is Enter Email Address; when rendered, this label will appear above the ‘email’ input field. HTML provides a large amount of flexibility, and so in general a descriptive text that appears alongside a specific element when rendered can be very far from that element when looking at the HTML plaintext. Thus, this task evaluates a model’s ability to understand the structure of HTML as it would appear to a user, despite not having access to the rendered web page directly.

### 4 Datasets

Each of our canonical tasks requires a separate dataset, with the description generation task using a newly contributed, auto-labelled dataset based on CommonCrawl.

**Autonomous Web Navigation.** We use the 12K demonstrations included in the publicly available MiniWoB benchmark (Shi et al., 2017), which encompass 62 website applications ranging from...
email forwarding to social media interactions. Each demonstration is a sequence of (instruction, HTML, action) tuples. Every element in a MiniWoB demonstration is accompanied by a reference number unique within its respective pages. This number can be used as an element selector, making the action space unified across all tasks and time steps. For instance, the action in Figure 1 would be type(ref=5, "username@email.com"), where 5 refers to the index of the input when counted from top-to-bottom. As model input, we concatenate the natural language instruction and HTML into a single text input sequence. Similarly, we treat the action as a text sequence for the model to predict.

**Semantic Classification.** We use a dataset of 28K labelled examples, containing 66 different categories, of the form (HTML, element, category), previously used in the context of environment generation (Gur et al., 2021). The dataset consists of HTMLs from real-world shopping websites and categories relevant to form-filling during payment and checkout on these websites.

**Description Generation.** For this task, we derive a dataset from CommonCrawl.3 CommonCrawl does not include renderings or annotations that would reveal what text in the HTML is associated with which elements. Instead, we infer descriptions of various elements by exploiting a special attribute in the HTML schema known as for. As an example in Figure 1, the first label in the HTML has a for attribute with value uName, which is the id of the element described by label; in this case, the id is that of the first input in the page. This annotation does not affect the rendering of the page and is typically used for accessibility purposes. We utilize the information given by these for attributes to create a large-scale dataset4 to study description generation.

Specifically, we collected 100 WARC (from April 2019) files from the CommonCrawl project and extracted all HTML label1s that have a for attribute. Removing non-Unicode and alphanumeric text in HTML label1s results in a 400K example dataset. We balance the distribution of labels, effectively downsampling the dataset to 85K samples. Each example is represented as (HTML, element, description), where HTML is the HTML plaintext of the page, element is the element whose id attribute matches that appearing in the label1’s for attribute, and description is the text inside the label1 element (see example in Figure 1). More details of the dataset can be found in Appendix A.2.

<table>
<thead>
<tr>
<th>Task</th>
<th>Dataset</th>
<th>Size</th>
<th>Input</th>
<th>Architecture</th>
<th>Output</th>
<th>Task Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Autonomous Web Navigation</td>
<td>MiniWoB Demos (Shi et al., 2017)</td>
<td>12K</td>
<td>Page</td>
<td>Enc-Dec-Dec</td>
<td>Text</td>
<td>Dictionary</td>
</tr>
<tr>
<td>Semantic Classification</td>
<td>Annotated Shopping Webpages (Gur et al., 2021)</td>
<td>28K</td>
<td>Snippet</td>
<td>All</td>
<td>Text</td>
<td>Category</td>
</tr>
<tr>
<td>Description Generation</td>
<td>CommonCrawl (new)</td>
<td>85K</td>
<td>Snippet</td>
<td>Enc-Dec-Dec</td>
<td>Text</td>
<td>Text</td>
</tr>
</tbody>
</table>

Table 1: Task, dataset, and model summary. All models receive raw HTML. Autonomous Web Navigation receives the entire HTML, while the other tasks receive HTML snippets extracted given salient element.

5 Pre-Processing

In treating HTML as token sequences, we minimize any HTML tree pre-processing prior to model input. We thus provide HTML as raw text (i.e., sequences of text tokens) and only apply a snippet extraction pre-processing for pages which are too large to fit into the typical LLMs context windows.

**Snippet Extraction.** Real HTML pages can grow extremely large, reaching thousands of elements, far beyond the context window of the largest LLM that we studied (1920 tokens in PaLM (Chowdhery et al., 2022)). LLMs typically truncate such long sequences, which can be detrimental to HTML understanding as HTMLs are not linearly structured. We take an element-centric approach and extract HTML snippets (a small portion of HTML code) surrounding a salient element (Figure 5). A simple heuristic, which controls the tree’s width and depth, guides the process: Start with a salient element and traverse its ancestors in the HTML tree until a stopping condition is satisfied. As we traverse up, we estimate the height of the tree and the increased number of descendants of the new root. We stop when either metric violates a pre-defined limit and take the resulting sub-tree as the snippet. We mark the salient element using a special attribute, called target, to distinguish it from other elements. We perform the snippet extraction

---

3http://commoncrawl.org

4https://console.cloud.google.com/storage/browser/gresearch/webllm/datasets/descgen
for the semantic classification and description generation datasets, and keep the full HTML pages in MiniWoB because these pages are typically much smaller than real-world HTML.

**HTML un-Parsing.** We provide the models with the unparsed plaintext HTML in the form of a sequence of tokens. This canonical representation does not require specific model architectures such as hierarchical networks (Liu et al., 2018; Gur et al., 2021) and can be fed into any LLM. We transform all datasets by converting every HTML page or snippet into a sequence. For MiniWoB, we additionally concatenate (action history, instruction, HTML) tuples into a single sequence.

6 Model Training

We study a variety of transformer-based LLMs (Vaswani et al., 2017) with different sizes and architectures for HTML understanding tasks (Table 1). In the rest of the text, we prefix models fine-tuned for Autonomous Web Navigation, Description Generation, and Semantic Classification with WebN-, WebD-, and WebC-, respectively. For instance, WebD–T5-3B is the three billion parameter T5 model (Raffel et al., 2020) fine-tuned for the Description Generation task. The rest of this section elaborates on training details.

**Encoder-Decoder and Decoder-only Models.** We train encoder-decoder models, i.e., T5 (Raffel et al., 2020), and decoder-only models, i.e., LaMDA (Thoppilan et al., 2022) and PaLM (Chowdhery et al., 2022), with text input and text output. Inputs are raw HTML pages or snippet texts; similarly, outputs are categories, natural language descriptions, or actions represented as text. Namely, for Semantic Classification we use the textual representation of categories, similar to previous classification problems in NLP (Raffel et al., 2020). For Autonomous Web Navigation, actions are converted into text by first converting them into key and value pairs and then concatenating the pairs.

Many websites in MiniWoB require multiple interactions, such as click-button-sequence or click-checkboxes, where each interaction might cause a subtle change in the website state. For instance, after clicking on a checkbox in the click-checkboxes website, its value flips from positive to negative or the other way around, which is not always reflected in LLMs’ predictions and leads to action repetitions. We solve this issue by augmenting tuples in the dataset with a sequence of past actions, (action history, instruction, HTML, action), and allowing LLMs to learn from past experience.

**Encoder-only Models.** We train encoder-only models, i.e., BERT (Devlin et al., 2018), with text input and categorical output. We keep semantic categories as discrete one-hot classes. To train encoder-only models, we add a new classification layer after the final encoder layer to produce a distribution over semantic categories. In addition to the typical BERT models, we study MobileBERT (Sun et al., 2020), distilled from BERT-large with inverted bottlenecks, and Albert-XL (Lan et al., 2020), with parameter sharing and embedding split.

7 Results

We now present the results of fine-tuned LLMs for HTML understanding. We compare the models’ performance with the existing baselines where possible (autonomous web navigation) and against other LLM architectures and training regimes (all tasks). Sections 7.1, 7.2, and 7.3 evaluate task-specific performance, while Section 7.4 assesses the performance across all the tasks.

**Metrics:** For autonomous web navigation we evaluate models’ Success Rate, which is averaged over 100 episodes per task. For the other tasks, we use Accuracy to measure exact match between prediction and ground truth. In the description generation task, we additionally provide evaluations using alternative ‘soft’ text evaluation metrics, BLEU and ROUGE-1, measuring the similarity between predicted and ground truth text.

7.1 Autonomous Web Navigation Results

For Autonomous Web Navigation we fine-tune two WebN- encoder-decoder architectures (WebN-T5-large and WebN-T5-3B) on 12k demonstrations from human-annotated real websites. We evaluate the models on MiniWob (Liu et al., 2018) benchmark, and compare with specialized architectures trained using supervised learning (SL) on 2.4 million human expert demonstrations CC-Net (SL) (Humphreys et al., 2022), and two RL models bootstrapped with SL, CC-Net (SL) (CC-Net (SL & RL) (Humphreys et al., 2022), and WGE (SL & RL) (Liu et al., 2018)). Additionally, we compare with the decoder-only architecture (WebN-Lambda-1B) and perform an ablation study on the impact of including the action history in the input.
Figure 2: a) WebN–T5* performance compared to the previous SOTA models on MiniWoB benchmark. WebN–T5-3B improves the task success 16% while using 192 times less data, compared to the best supervised learning (SL) model, CC-Net (SL). LLMs performance is only surpassed by works utilizing RL, requiring orders of magnitude more online experience interaction with websites. b) LLMs with and without pretraining on Autonomous Web Navigation task. Those with pretraining (denoted by the ‘WebN-’) prefix show a 2.5–4.5x performance improvement.

**Comparison to SoTA.** Since previous works report success on only a subset of websites in MiniWoB, we evaluate on 48 out of 62 websites that are common across all models. Table 8 in the Appendix reports fine-grained results while Figure 2a presents results averaged over all websites. Compared to CC-Net (SL) which is trained on all 2.4M demonstrations, WebN-T5-3B improves the success 16% while only training on 12K publicly-available demonstrations, yielding over 192x improvement in sample-efficiency. We find that all choices of LLMs outperform previous SL models. Notably, WebN-T5-3B significantly improves on websites requiring multiple-action sequences such as click_checkboxes or websites requiring entering text such as login_user (Table 8). We observe that the performance of LLMs is only surpassed by previous works utilizing RL, which require orders of magnitude more online experience interaction. Extending our fine-tuned LLMs to an RL setting is a promising avenue for future work.

**Action history ablation.** Across all LLMs we consistently observe a decrease in success, on average 6.4%, when past actions are excluded from the inputs (Figure 2a). Action history helps with websites that require entering multiple texts, as well as understanding minor changes that could be difficult to detect (e.g. click_checkboxes and multi_layout). multi_layout requires entering 3 different texts in the website where the layout is randomized at each episode, yet, surprisingly, even the (relatively smaller) WebN-T5-large model without action history outperforms the CC-Net (SL) model; illustrating that incorporating action history is not the only contributing factor for the better success.

**7.2 Semantic Classification Task Results**

To evaluate the Semantic Classification task, we compare the T5 encoder-decoder architecture’s three size variants (WebC-T5-base, WebC-T5-large, and WebC-T5-3B) fine-tuned on 22K real, human-labeled training websites. We compare with a fine-tuned encoder only architectures (WebC-*BERT*), three fine-tuned decoder-only architectures (WebC-LaMDA and PaLM), and both encoder-decoder and decoder-only models trained on human labeled websites from scratch. Results are presented in Table 2, where we find that all WebC-LLMs perform well and significantly better than the same architectures without pretraining.

![Image of a table showing LLM performance on the Semantic Classification task.](image-url)
## 7.3 Description Generation Task Results

For Description Generation we split the CommonCrawl dataset based on URL top-level domains to test LLMs’ capabilities to generalize to unseen HTML. We fine-tune encoder-decoder architectures (WebD–T5*) and decoder-only models (WebD–LaMDA*), with results presented in Table 3. We also evaluate a strong heuristic baseline which simply finds the description closest to the salient element in the HTML text (Closest Description).

### Accuracy and Similarity Performance

We show results of our evaluations in Table 3. All models achieve high scores across all metrics, achieving $\approx 84\%$ on the accuracy in terms of exact match and a higher non-exact match score based on BLEU and ROUGE-1 ($\approx 91\%$). This difference indicates that the models are capable of locating the descriptions, but not always generating the exact output.

## 7.4 HTML Understanding LLMs’ Performance Analysis Across Tasks

We now analyze our results in aggregate to derive our main conclusions.

### 7.4.1 Pretraining Effect: Pretraining on Large Text Corpora Matters

Fine-tuned pretrained LLMs outperform LLMs trained on HTML-only data, improving the performance by more than 34.1% on the Autonomous Web Navigation (Table 2b), and 10% to 12.7% on the Semantic Classification task (Table 2).

Since Autonomous Web Navigation is the most difficult task, the improved performance is an encouraging evidence of the value of LLMs in HTML understanding tasks. Specifically, we observe that LLMs without pretraining are comparable to fine-tuned pretrained models only on websites that require simple text matching. In contrast, for websites such as click_checkboxes, text matching is harder and we find that pretraining is key to good performance. We also found that without pretraining, model outputs were frequently in an incorrect format such as invalid dictionaries or invalid refs with non-integer values. This suggests that the large corpora used for pretraining helps models to learn general HTML structure.

---

### Table 3: Summary of results for Description Generation

<table>
<thead>
<tr>
<th>Data Size</th>
<th>New descendants (%)</th>
<th>Height</th>
<th>Test (%)</th>
<th>Dev (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>25</td>
<td>3</td>
<td>87.7</td>
<td>90.3</td>
</tr>
<tr>
<td>25</td>
<td>25</td>
<td>4</td>
<td>88.6</td>
<td>89.2</td>
</tr>
<tr>
<td>50</td>
<td>50</td>
<td>3</td>
<td>88.4</td>
<td>90.0</td>
</tr>
<tr>
<td>50</td>
<td>50</td>
<td>4</td>
<td>89.3</td>
<td>89.2</td>
</tr>
<tr>
<td>300</td>
<td>300</td>
<td>5</td>
<td>87.8</td>
<td>75.8</td>
</tr>
<tr>
<td>500</td>
<td>500</td>
<td>7</td>
<td>75.8</td>
<td>74.5</td>
</tr>
</tbody>
</table>

(a) Data Size, Accuracy

(b) Accuracy over training data size. Using only 1000 labeled examples (4.4% of all training dataset), WebC-T5-3B outperforms T5-3B (full data without pretraining) which is trained on all available labeled data (approximately 30k examples), and outperforms WebC-PaLM-8B which is an order of magnitude larger.

---

## Figure 3

(a) Effect of snippet extraction parameters on WebC-T5-3B. Increases above 50% in new descendants and height of 4. Large increases in both parameters lead to large snippets and decrease in accuracy.

(b) Accuracy over training data size. Using only 1000 labeled examples (4.4% of all training dataset), WebC-T5-3B outperforms T5-3B (full data without pretraining) which is trained on all available labeled data (approximately 30k examples), and outperforms WebC-PaLM-8B which is an order of magnitude larger.
<table>
<thead>
<tr>
<th>Model Name</th>
<th>Test</th>
<th>Dev</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Accuracy(%)</td>
<td>BLEU</td>
</tr>
<tr>
<td>WebD-T5-large</td>
<td>83.2</td>
<td>90.2</td>
</tr>
<tr>
<td>WebD-LaMDA-1B</td>
<td>83.3</td>
<td>87.5</td>
</tr>
<tr>
<td>WebD-T5-3B</td>
<td>84</td>
<td>90.8</td>
</tr>
<tr>
<td>Closest Description</td>
<td>57.4</td>
<td>24.4</td>
</tr>
</tbody>
</table>

Table 3: Description generation accuracy of LLMs.

Figure 4: Accuracy per classification category of the WebC-T5-3B model on the development dataset.

7.4.2 Architecture Effect: T5-based Models Perform Best Across All Tasks

Encoder-decoder T5 based models perform better across all three tasks. On the Autonomous Web Navigation task, encoder-decoder (WebN-T5) architectures are better or comparable to WebN-LaMDA-1B (Figure 2a). On the Semantic Classification, the smallest encoder-decoder model (WebC-T5-base) performs comparably to much larger decoder-only models (WebC-LaMDA-1B or WebC-PaLM-8B) and the largest encoder-only model (WebC-BERT-large) which has 85M more parameters (Table 2). We also observe that decoder-only PaLM-8B performs worse than much-smaller encoder-decoder T5-large when trained only on HTML data. Finally, on the Description Generation encoder-decoder architecture has higher BLEU score.

One possible explanation for the strong performance of T5-based models is the encoder-decoder architecture of these models. Namely, T5 models utilize an encoder with a bidirectional attention mechanism, not present in the LaMDA and PaLM decoders. The bidirectional attention mechanism can process HTML pages from both ends, potentially overcoming the loss of information when tree-structured HTML pages are converted into a fixed linear text sequences.

7.4.3 Model Size Effect: Size (Sub-linearly) Matters

Across the tasks it appears that the architecture plays an important role in the model performance. Model size and performance are also positively correlated, although they reach diminishing returns. For instance, the model performance is roughly $O(\log \log n)$ with respect to model size on Semantic Classification (Figure 6 in Appendix). On the Autonomous Web Navigation task, performance grows slowly with the model size (Table 8), while on the Description Generation it plateaus (Table 3).

7.4.4 Error Analysis

We manually examined 50 errors of WebC-T5-3B model over the development set (Table 4) and assigned them into one of the 9 error types that we devised. We found that 32% of the errors are due to lack of information in the HTML snippets, which is mainly the result of lost information during snippet extraction process. Annotation errors or email/username ambiguity make up 30% of the errors. These can’t be improved without revising the annotated data or adding extra information to resolve the ambiguity. We also found that the model sometimes picks a more general category, or a nearby text misleads the model; the latter usually happens when the HTML snippet is long where majority of the elements are noise.

7.4.5 Few-Shot Prompting

In Table 5, we present few-shot prompting performance of a 540B PaLM model. We probe the model using a prompt template `<html> Role: <category> with 1 example per category and generate categories using greedy-decoding. In our
### Table 4: Types of errors over 50 manually examined examples. 32% of errors are due to lack of information in HTML snippets, 30% of errors are related to annotations or can’t be improved due to ambiguity (email/username), and the remaining errors are incorrect predictions by the model.

<table>
<thead>
<tr>
<th>Error Type</th>
<th>Percentage of Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not enough information in the HTML snippet</td>
<td>30</td>
</tr>
<tr>
<td>Incorrect annotation (ex: &quot;unknown_role&quot; instead of &quot;organization&quot;)</td>
<td>12</td>
</tr>
<tr>
<td>Annotation tool translates user selection incorrectly</td>
<td>8</td>
</tr>
<tr>
<td>Email/Username ambiguity</td>
<td>10</td>
</tr>
<tr>
<td>More general category (ex: &quot;header&quot; instead of &quot;cart_header&quot;)</td>
<td>8</td>
</tr>
<tr>
<td>Immediate neighboring text misleads</td>
<td>8</td>
</tr>
<tr>
<td>Incorrect date formatting (ex: &quot;mm&quot; instead of &quot;mmm&quot;)</td>
<td>4</td>
</tr>
<tr>
<td>No information in the HTML snippet</td>
<td>2</td>
</tr>
<tr>
<td>Others</td>
<td>18</td>
</tr>
</tbody>
</table>

Table 5: Few-shot prompting performance with different pre- and post-processing steps.

<table>
<thead>
<tr>
<th>Model Name</th>
<th>Test</th>
<th>Dev</th>
</tr>
</thead>
<tbody>
<tr>
<td>PaLM-540B</td>
<td>64.2</td>
<td>60.3</td>
</tr>
<tr>
<td>- w/o Example Cleaning</td>
<td>57.9</td>
<td>57.2</td>
</tr>
<tr>
<td>- w/o Category Rewriting</td>
<td>52.1</td>
<td>50.7</td>
</tr>
<tr>
<td>- w/o Dictionary Mapping</td>
<td>45.6</td>
<td>45.1</td>
</tr>
</tbody>
</table>

Preliminary experiments, we found that few-shot prompting achieves only 45.6 accuracy, much lower than a model fine-tuned on the same data. We found two common problems – the model is not able to canonicalize predictions into categories and many of the examples are dropped due to context length.

We developed post-processing methods to alleviate the canonicalization problem and pre-processing methods to reduce lengths of examples. Adding a dictionary-based mapping on predictions – a manually curated paraphrase dictionary – improves the performance to 52.1. We also tried rewriting predictions by changing the order of tokens around "_" such as name_first to first_name which further improved the performance to 57.9. Finally, we cleaned examples in the prompt by removing certain elements such as "svg", "path", "img", and "iframe" and also removing class attribute from every element; this pre-processing step gives 64.2.

## 8 Conclusion

We presented canonical tasks and fine-tuned LLMs for HTML understanding. The comprehensive evaluations and analyses over a range of architectures, dataset sizes, and baselines yields practical findings and highlights current limitations of these models. We find that a) pretraining is critical for the performance and can reduce labeled data requirements, improving sample efficiency up to 200x; b) model architecture is the second-most important factor, and T5 models with bidirectional attention and encoder-decoder architecture perform the best across the board; c) given a choice, model size should be evaluated in the context of the model’s training and inference performance, as the model size sub-linearly correlates with its performance. Finally, the proposed HTML understanding tasks highlight the relatively short context window that limits current LLMs, suggesting possibilities for future research that incorporate or eliminate this constraint.

## 9 Limitations

Our experimental results are limited to relatively short context windows. While HTML documents can have 10s of thousands of tokens, LLMs that we studied have only 2K context windows. We developed heuristics to extract HTML snippets to fit into context windows which is a promising future direction. We are also limited by the MiniWoB simulator for our web navigation experiments. While MiniWoB serves a wide variety of simulated websites, they still lack many of the complexities of real websites.

## References


Armen Aghajanyan, Dmytro Okhonko, Mike Lewis,


Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2023. Toolllm: Facilitating large language models to master 16000+ real-world apis.


Qiaoyu Tang, Ziliang Deng, Hongyu Lin, Xianpei Han, Qiao Liang, Boxi Cao, and Le Sun. 2023. Toolalpaca: Generalized tool learning for language models with 3000 simulated cases.


A Appendix

A.1 Brief Background on HTML as Semi-Structured Text Data

HTML is a markup language, used to organize web page **structure** and **content**. Consider the example HTML page in Figure 1. This web page includes two adjacent input elements, one for e-mail and another for password, with their corresponding labels on a separate branch of the page. These inputs and labels are one of many possible **elements** that serve as HTML building blocks.
Each element has a set of attributes – key and value pair – that describe the element’s content, such as style and human-readable text. When rendered in a browser, these attributes will be responsible for how the element is shown and where it is positioned. In the example in Figure 1, the first input has three attributes, tag="input", type="email", and id="uName", that identify the element as an email input with an identifier (“uName”) that can be accessed programmatically.

A.2 Dataset and Pre-Processing Details
Examining the description distribution, we found the original 400K dataset to be very skewed; only 20 descriptions (such as Email and Password) were covering 50% of the dataset. We sub-sampled the dataset so that each unique description has at most 10 data points. We also found that for attributes are almost always defined for HTML label. This could cause a model to overfit and just find the label element in the HTML and ignore everything else. To avoid this sort of ‘cheating’ we replace the tags of HTML label by randomly sampling from {div, span, a, label}. These tags are also frequently used to inject text in HTML but they are very rarely used with for attributes. Finally, we removed examples where there are only a single text in the HTML since models can trivially generate descriptions by finding the only text in the HTML, which biases model weights and evaluation metrics. After this final step, we have a total of 85K labeled examples.

We didn’t apply any special filtering to keep only text related information. We did minimal pre-processing to filter some of the attributes from excessively long inputs. For semantic classification, we applied no filtering on extracted snippets. For description generation, we filtered “class” and “style” attributes as we found them to increase the length of the HTML documents significantly. In MiniWoB, we applied no filtering and used the original observation space provided by the environment.

A.2.1 Snippet Generation
In Figure 5, we give a high-level overview of our snippet generation procedure.

A.3 Action Space for the Autonomous Agent
We set the number of tokens that our models can generate to 20. Our vocabulary consists of 32K tokens, which gives an initial estimate of 32000\(^{20}\) candidate generations at each step. While we don’t constrain the vocabulary of our models during generation, it is important to note that output tokens in our dataset typically come from input HTML documents or user instructions (with formatting tokens such as "\{", ":", ":\}" and special "click", "type" tokens being exceptions). The models easily learn these implicit constraints and assign much lower mass to tokens that are not in HTML document or instruction. Additionally, they also quickly learn the desired parsable format ("<action type>, <target HTML element identifier>, <instruction substring for typing actions>") where the target HTML element specified in the action output is actually present in the input HTML (so it can be acted upon). We estimated the average statistics of the public demonstrations that we used to train our models and found that there are 54 elements in a given HTML document and 35 instruction tokens, on average. So our action space is approximately \(2 \times 54 \times 35^2\) where there are 2 action types and \(35^2\) number of substrings in an instruction.

A.4 Sample Episodes from MiniWoB
See Table 6 for an example episode of web navigation inferred by a fine-tuned LLM.

A.5 Detailed MiniWoB Results
See Table 7 for detailed performance of various models on MiniWob.

A.6 Resource Requirements
See Table 8.

A.7 Structure Dependence Ablation Study
We conducted an ablation study to examine the sensitivity of model performance to preserving structural information. To do so, we evaluate the model’s performance on HTML input with critical structure components removed. We kept the order of elements and their attributes fixed while corrupting the nesting structure by removing closing tags.

Removing closing tags corresponds to a valid traversal (BFS) and keeps the order of elements the same as the text based input.

As a simple example:

```html
<div id="form"><div><input id="username"></div></div>
```

would be converted into:

```html
<div id="form"><input id="username"></div>
```
We evaluated the trained WebN-T5-3B model on the same set of synthetic websites from the MiniWoB benchmark with this aspect of structure removed from the HTML pages. WebN-T5-3B achieves a 45.4% success rate, 6% lower than before, suggesting that WebN-T5-3B is at least partially dependent on the DOM topology.

A.8 Additional Related Works

A.8.1 Task-specific Models
An alternative to LLMs is to adapt bespoke task-specific architectures tailored towards processing of structured documents and HTML ((Li et al., 2021b,a)).

StructuralLM ((Li et al., 2021a)) is an approach specifically tailored for document understanding (i.e., combinations of images and text), and thus makes several simplifying assumptions for its model that limit its applicability to HTML understanding (i.e., trees of elements with a richer structure and functionality). It is trained only on the textual content of a document - the markup information is ignored. For example, any input field or dropdown in a document would be missing from the model inputs. All of the tasks we study require knowledge of this information. For example, in autonomous navigation the model needs to interact with input elements (e.g., text, checkboxes, dropdowns) such as username and password in the login-user task in MiniWoB. Typically, a “type” action with a reference to an element and a text argument is generated by the model. Without knowing which input elements are available in the page, it is impossible to generate a reference to any input element.

While MarkupLM ((Li et al., 2021b)) is better tailored for understanding HTML pages, it has similar drawbacks as StructuralLM in that it focuses solely on text and structure of text while ignoring everything else in the markup. To illustrate our point better, we used the open source implementation of MarkupLM from the HuggingFace library ((Wolf et al., 2019)) to process the sample HTML snippet in Figure-1. The MarkupLM ignores all input elements, both username and password, and generates `<s>Email AddressEnter Password:Please enter your password.</s>` which is the text input to the MarkupLM Transformer (Figure 7). Classifying this text as username or password is not possible without the additional context on which input element is the salient element (in this context it is the username).

MarkupLM is also evaluated on NLP-like tasks such as QA or entity classification where understanding page content is paramount, whereas we focus on HTML understanding tasks such as autonomous navigation where both content and the page’s layout structure need to be understood.
We perform a quantitative evaluation of MarkupLM on our tasks to understand how significant these limitations are. We fine-tune the MarkupLM-base model on the semantic classification task, using the same setup as other WebC models but with the suggested hyperparameters from (Li et al., 2021b). We use the MarkupLM implementation from the HuggingFace library (Wolf et al., 2019). On development and test sets, MarkupLM-base achieves 65% and 66% accuracy, respectively. These results are more than 16% lower compared to similar size WebC-BERT-base results that we report in our work. This suggests that although domain specific models may be suitable for processing HTML for NLP tasks, the generality, flexibility, and sample efficiency LLMs provide advantages for autonomous navigation tasks.

A.8.2 Web Information Extraction

WebSRC (Chen et al., 2021) introduced a QA task from web documents. The authors manually curated a small set of seed questions (460 questions) which are used to collect paraphrases and do data augmentation. Similar to DescriptionGeneration, the problem is to find the relevant text in a web document. While DescriptionGeneration is focused more on elements and fully automated, WebSRC is a QA task that requires manual annotation.

WIERT (Li et al., 2023) studies information extraction from web documents. They focus on utilizing simplified render trees (only text, tag, and style information are kept) to classify DOM nodes into product categories. They traverse the render tree to generate a sequence of element tokens and encode the sequence with a pretrained language model; style nodes are independently encoded and concatenated to the element encodings. The resulting encodings are used in a multi-objective setup to train the model. In contrast, our models are trained in a unified setup with a single objective, and using original HTML documents with no special filtering required to extract informative elements.

A.8.3 Tool Use

More recently, tool-using agents emerged as a way to augment LLMs’ capabilities with external tools. ToolAlpaca (Tang et al., 2023) and TOOLLM (Qin et al., 2023) are two examples that study multi-step tool-use with iterated API calls. Actions as well-defined API interfaces can be useful to simplify and reduce the dimensionality of the interface between the model and environment. This in turn enables a degree of horizontal scaling and generalization as studied by these works. However, this flexibility is only achieved when the environment implements an API interface. Where there are none available, navigating web interfaces becomes crucial to unlock access to further information.
from transformers import MarkupLMProcessor
processor = MarkupLMProcessor.from_pretrained(f"microsoft/markuplm-base")
snippet = ''
<\div><\label class="form-label" for="uName">Email Address</\label>
<\label class="form-label" for="pass">Enter Password:</\label>
<\div><\input type="email" id="uName" target><\input type="password" id="pass"><\span class="hidden">Please enter your password.</\span></\div>''
encoding = processor(snippet)
print(processor.batch_decode(encoding["input_ids"]))

Figure 7: Code snippet to reproduce MarkupLM preprocessing result. Encoding the example HTML code with the MarkupLM preprocessor will ignore all input elements.
Table 6: A sample web page and corresponding episode using the T5-3B model. At each time step, previous actions, instruction, and HTML are concatenated into a single HTML text. Note that at the beginning of episode, there is no past actions and we simply concatenate instruction and HTML. Action is generated as a sequence of tokens which is later parsed into a dictionary. The ref in the action points to an element that has a ref attribute with the same value. For instance, at the beginning of episode, ref: 6 corresponds to an input with ref=6. At the end of the episode, the model clicks on the submit button and the episode terminates.

<table>
<thead>
<tr>
<th>Web page</th>
<th>HTML Text</th>
<th>Action Text</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Web page" /></td>
<td>select qw, mMyP, YMYQNa, DJIG, 11J3 and click Submit. body ref=&quot;1&quot;&lt;div id=&quot;area&quot; ref=&quot;3&quot;&gt;div id=&quot;boxes&quot; ref=&quot;4&quot;&gt;label ref=&quot;5&quot;&gt;input type=&quot;checkbox&quot; id=&quot;cb11&quot; ref=&quot;6&quot; value=&quot;False&quot;&gt;&lt;/input&gt;&lt;t class=&quot;text CLASS&quot; ref=&quot;None&quot;&gt;&lt;b ref=&quot;7&quot;&gt;label ref=&quot;9&quot;&gt;input type=&quot;checkbox&quot; id=&quot;ch11&quot; ref=&quot;8&quot; value=&quot;False&quot;&gt;&lt;/input&gt;&lt;t class=&quot;text CLASS&quot; ref=&quot;None&quot;&gt;&lt;b ref=&quot;10&quot;&gt;label ref=&quot;12&quot;&gt;input type=&quot;checkbox&quot; id=&quot;ch2&quot; ref=&quot;14&quot; value=&quot;False&quot;&gt;&lt;/input&gt;&lt;t class=&quot;text CLASS&quot; ref=&quot;None&quot;&gt;&lt;b ref=&quot;15&quot;&gt;input type=&quot;checkbox&quot; id=&quot;ch3&quot; ref=&quot;16&quot; value=&quot;False&quot;&gt;&lt;/input&gt;&lt;t class=&quot;text CLASS&quot; ref=&quot;None&quot;&gt;button id=&quot;subbtn&quot; class=&quot;secondary-action&quot; ref=&quot;17&quot;&gt;Submit&lt;button id=&quot;div&quot;/}}{{/div}}/body&gt;</td>
<td>{action: click, ref: 6}</td>
</tr>
<tr>
<td></td>
<td>(action: click, ref: 10) Select qw, mMyP, YMYQNa, DJIG, 11J3 and click Submit. body ref=&quot;11&quot;&lt;div id=&quot;area&quot; ref=&quot;3&quot;&gt;div id=&quot;boxes&quot; ref=&quot;4&quot;&gt;label ref=&quot;5&quot;&gt;input type=&quot;checkbox&quot; id=&quot;cb11&quot; ref=&quot;6&quot; value=&quot;True&quot;&gt;&lt;/input&gt;&lt;t class=&quot;text CLASS&quot; ref=&quot;None&quot;&gt;&lt;b ref=&quot;7&quot;&gt;label ref=&quot;9&quot;&gt;input type=&quot;checkbox&quot; id=&quot;ch11&quot; ref=&quot;8&quot; value=&quot;False&quot;&gt;&lt;/input&gt;&lt;t class=&quot;text CLASS&quot; ref=&quot;None&quot;&gt;&lt;b ref=&quot;10&quot;&gt;label ref=&quot;12&quot;&gt;input type=&quot;checkbox&quot; id=&quot;ch2&quot; ref=&quot;14&quot; value=&quot;False&quot;&gt;&lt;/input&gt;&lt;t class=&quot;text CLASS&quot; ref=&quot;None&quot;&gt;&lt;b ref=&quot;15&quot;&gt;label ref=&quot;16&quot;&gt;input type=&quot;checkbox&quot; id=&quot;ch3&quot; ref=&quot;19&quot; value=&quot;False&quot;&gt;&lt;/input&gt;&lt;t class=&quot;text CLASS&quot; ref=&quot;None&quot;&gt;button id=&quot;subbtn&quot; class=&quot;secondary-action&quot; ref=&quot;19&quot;&gt;Submit&lt;button id=&quot;div&quot;/&gt;{{/div}}/body&gt;</td>
<td>{action: click, ref: 10}</td>
</tr>
<tr>
<td></td>
<td>(action: click, ref: 12) Select qw, mMyP, YMYQNa, DJIG, 11J3 and click Submit. body ref=&quot;12&quot;&gt;div id=&quot;area&quot; ref=&quot;3&quot;&gt;div id=&quot;boxes&quot; ref=&quot;4&quot;&gt;label ref=&quot;5&quot;&gt;input type=&quot;checkbox&quot; id=&quot;cb11&quot; ref=&quot;6&quot; value=&quot;True&quot;&gt;&lt;/input&gt;&lt;t class=&quot;text CLASS&quot; ref=&quot;None&quot;&gt;&lt;b ref=&quot;7&quot;&gt;label ref=&quot;9&quot;&gt;input type=&quot;checkbox&quot; id=&quot;ch11&quot; ref=&quot;8&quot; value=&quot;False&quot;&gt;&lt;/input&gt;&lt;t class=&quot;text CLASS&quot; ref=&quot;None&quot;&gt;&lt;b ref=&quot;10&quot;&gt;label ref=&quot;12&quot;&gt;input type=&quot;checkbox&quot; id=&quot;ch2&quot; ref=&quot;14&quot; value=&quot;False&quot;&gt;&lt;/input&gt;&lt;t class=&quot;text CLASS&quot; ref=&quot;None&quot;&gt;&lt;b ref=&quot;15&quot;&gt;label ref=&quot;16&quot;&gt;input type=&quot;checkbox&quot; id=&quot;ch3&quot; ref=&quot;19&quot; value=&quot;False&quot;&gt;&lt;/input&gt;&lt;t class=&quot;text CLASS&quot; ref=&quot;None&quot;&gt;button id=&quot;subbtn&quot; class=&quot;secondary-action&quot; ref=&quot;19&quot;&gt;Submit&lt;button id=&quot;div&quot;&gt;{{/div}}/body&gt;</td>
<td>{action: click, ref: 12}</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-------</td>
<td>------------</td>
</tr>
<tr>
<td>browse-simple</td>
<td>0.89</td>
<td>0.76</td>
</tr>
<tr>
<td>browse-video</td>
<td>0.94</td>
<td>0.84</td>
</tr>
<tr>
<td>click-test</td>
<td>0.87</td>
<td>0.99</td>
</tr>
<tr>
<td>grid-coordinate</td>
<td>0.87</td>
<td>0.98</td>
</tr>
<tr>
<td>highlight-text-2</td>
<td>0.79</td>
<td>0.87</td>
</tr>
<tr>
<td>identify-shape</td>
<td>0.8</td>
<td>0.77</td>
</tr>
<tr>
<td>moving-items</td>
<td>0.86</td>
<td>0.97</td>
</tr>
<tr>
<td>navigate-tree</td>
<td>0.99</td>
<td>0.99</td>
</tr>
<tr>
<td>read-table-2</td>
<td>0.54</td>
<td>0.87</td>
</tr>
<tr>
<td>use-slider</td>
<td>0.8</td>
<td>0.77</td>
</tr>
<tr>
<td>visual-attention</td>
<td>0.82</td>
<td>0.84</td>
</tr>
</tbody>
</table>

Note that these are normalized between 0 and 1.

Table 7: Success rate comparison of various models in MiniWoB tasks. Baseline results are borrowed from (Humphreys et al., 2022).
Table 8: Resource requirements and running time of LLMs.

<table>
<thead>
<tr>
<th>Model Name</th>
<th>Model Size</th>
<th>TPU version</th>
<th>Batch size</th>
<th>Input sequence length</th>
<th>Examples per sec (training)</th>
<th>Examples per sec (inference)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PaLM</td>
<td>62B</td>
<td>TPU v4</td>
<td>8</td>
<td>1920</td>
<td>9.313</td>
<td>30.51</td>
</tr>
<tr>
<td>PaLM</td>
<td>8B</td>
<td>TPU v4</td>
<td>32</td>
<td>1920</td>
<td>64.4</td>
<td>184.3</td>
</tr>
<tr>
<td>T5</td>
<td>3B</td>
<td>TPU v4</td>
<td>128</td>
<td>512</td>
<td>163.8</td>
<td>734.5</td>
</tr>
<tr>
<td>LaMDA</td>
<td>1B</td>
<td>TPU v2</td>
<td>128</td>
<td>512</td>
<td>363.1</td>
<td>1416</td>
</tr>
</tbody>
</table>