

Master Thesis

Navigating WebAI: Training Agents to Complete Web Tasks with Large Language Models and Reinforcement Learning

Lucas-Andreï Thil

Thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science of Artificial Intelligence
at the Department of Advanced Computing Sciences
of the Maastricht University

Thesis Committee:

Dr. M. Popa
Dr. G. Spanakis

Maastricht University
Faculty of Science and Engineering
Department of Advanced Computing Sciences

July 3rd, 2023

Abstract

Recent advancements in language models have demonstrated remarkable improvements in various natural language processing (NLP) tasks. Supervised learning (SL) approaches have achieved impressive performance while utilizing significantly less training data compared to previous methods. However, these SL-based models fall short when compared to reinforcement learning (RL) approaches, which have shown superior results. In this paper, we propose a novel approach that combines SL and RL techniques over the MiniWoB benchmark.

Furthermore, we address important limitations of previous works that claimed their models could understand HTML content. Our analysis reveals that these models tend to memorize the distributions of target elements rather than truly understanding the underlying HTML structure. To address this issue, we propose several methods aimed at rectifying this shortcoming and present a new baseline of results.

By integrating SL and RL techniques, we aim at leveraging the strengths of both approaches and overcome their individual limitations. Our experiments demonstrate that our proposed method outperforms previous SL approaches on some tasks using a fraction of the data, while also bridging the performance gap with RL models. Our best baseline models in SL achieve 43.58% average accuracy and combined with a multimodal RL approach scores 36.69% accuracy. The combination of these approaches presents a promising direction for future web navigation, and the study of individual tasks puts light on their limitations.

In conclusion, this paper introduces a hybrid approach that combines SL and RL techniques for language modeling over the MiniWoB benchmark. We highlight the limitations of previous works regarding the understanding of HTML content and propose effective methods to address this issue. The presented results showcase the effectiveness and limitations of our approach and set a new baseline for further exploration in the field of pre-trained language modeling for computer tasks.

1 Introduction

The web has become an integral part of our lives, with individuals dedicating a significant amount of time to perform various tasks, including professional activities and leisure pursuits. From administrative tasks to social interactions, the Internet supports a vast array of daily services. However, navigating the complex and ever-evolving landscape of the web can often be challenging and time-consuming. Users are faced with the daunting task of sifting through extensive web pages, clicking on relevant links, and interacting with different elements to accomplish their goals.

In response to this challenge, the field of AI has made significant advancements in developing intelligent agents capable of assisting users in web navigation. These agents employ various techniques, including machine learning, to understand user intentions and efficiently navigate the web on their behalf. However, traditional approaches often rely on large amounts of training data, making them less suitable for scenarios where limited examples are available or when facing new and unseen web environments.

We define a paradigm in AI for web navigation, as a model able to solve computer tasks in a few-shot manner (without being trained on many examples) over unseen environments, having an efficient amount of parameters to allow fast inference times as web navigation requires multi-step action sequences. This task remains tedious and only models such as GPT-4 achieved human-level results with few-shot abilities through iterative prompting mechanisms [1], but this is far from smaller and specialized models that can perform inference from a single device.

To achieve efficient and effective web navigation, our approach prioritizes models with an optimal number of parameters. By having an efficient parameter count, the model can perform faster inference, swiftly processing and navigating webpages. This speed is crucial in web navigation, where users expect quick responses and seamless interactions. Ideally, models should be efficient enough to run on a single device. In this work, our focus is on such models, aiming to understand their limitations and establish robust benchmarks to scale larger architectures in the future.

In this project, we are aiming at using HTML content directly combined with stylesheets and scripts, and visual inputs containing all the information of the environment through the Miniwob benchmark. We use the collection of 13k recorded episodes from the benchmark and combine a hierarchical planning approach to fine-tune T5 models over the navigation task and boost them through an RL approach.

Through this work, we aim to contribute to the advancement of AI-driven web navigation, enhancing user experiences and streamlining interactions in the complex online world. By developing smaller-scale models and demonstrating an efficient usage of techniques, we hope to contribute to more intelligent and adaptable web navigation agents that can fulfill the needs of users in their daily tasks.

1.1 Related Concepts

In this report, we will use several terms that we define here in Layman’s terms for a wider audience.

- **Machine learning (ML):** Machine learning is a type of artificial intelligence where computers learn from data and experiences to improve their performance on a specific task.
- **Supervised learning (SL):** Supervised learning is a machine learning technique where the computer is given labeled examples as input and learns to make predictions or classify new, unseen data based on those examples.

- **Reinforcement learning (RL):** Reinforcement learning is a type of machine learning where an agent learns to make decisions or take actions by interacting with an environment and receiving feedback or rewards for its actions.
- **Language Models:** Language models are computer programs that have been trained to understand and generate human language. They learn the patterns, grammar, and semantics of a given language by analyzing large amounts of text data. Language models can be used to perform various tasks such as generating text, answering questions, or translating languages.
- **Large Language Models (LLMs):** Large language models are advanced versions of language models that have been trained on massive amounts of text data. They are capable of processing and understanding vast amounts of information and generating human-like responses. These models use sophisticated algorithms and deep learning techniques to improve their language understanding and generation capabilities. Large language models, such as GPT-3, have billions of parameters and can produce coherent and contextually relevant text across a wide range of topics.
- **Web navigation:** Web navigation refers to the process of moving around and interacting with websites or web pages. It involves clicking on links, submitting forms, scrolling, and accessing different sections or content on the web.

1.2 HTML Markup

HTML (Hypertext Markup Language) markup is a language used to structure and format content on web pages. In layman’s terms, think of HTML as a set of instructions that tells a web browser how to display and organize different elements like text, images, headings, links, and more on a webpage, which is rendered by the browser.

Technically, HTML consists of a series of tags enclosed in angle brackets (“<” and “>”) in the fashion of a dictionary, that defines the structure and semantics of different elements on a webpage. For example, the `< h1 >` tag represents a top-level heading, the `< p >` tag represents a paragraph of text, and the `< img >` tag represents an image.

HTML works in conjunction with other markups or scripts such as CSS (Cascading Style Sheets) to control the appearance and layout of the elements, or Javascript to run front-end functions to cite a few. CSS provides rules for defining colors, fonts, sizes, positioning, and other visual aspects of the webpage, and Javascript allows it to perform dynamic changes on the page or run code through an interpreter directly on the loaded page on the client’s side. These elements are all part of the original HTML code, and in the context of training an agent to navigate the web from its content, it is crucial to account for them.

1.3 Research Questions

Based on the literature review, and the information gathered about the context of web environments, we can devise a series of research questions that this thesis will try to answer and precise its limitations.

The first one regards how well can agents trained in user interactions navigate different web interfaces. A second one corresponds to the evaluation of such models, where we investigate a set of metrics able to describe their performances and how clearly outline their weaknesses. A third research path corresponds to what are the best model architectures and machine learning techniques to consider and how to train them. Finally, we will explore the limitations of these models and complete by investigating the related risks.

2 Related Work

In recent years, significant advancements have been made in the field of AI-driven web navigation, with researchers exploring different approaches to improve the performance and capabilities of intelligent agents. This section provides an overview of the existing methods and techniques that have contributed to the development of this project.

2.1 OpenAI Universe

We begin by discussing the release of OpenAI’s Universe in 2016, which aimed to create a general-purpose agent capable of playing video games and interacting with interfaces [2][3], and among which focused also on web navigation over Miniwob, a framework containing over 80 tasks embedded in a web page. Miniwob is a reinforcement learning environment that acts as a sandbox for training AI agents in performing tasks from a given instruction. It allows testing and developing agents in relation to the complexities of web navigation tasks in a controlled setting.

2.2 Neural Network Oriented Approaches

Several works devised neural-network-oriented models for solving web navigation tasks. The most notable is from Humphrey et al. which presented their state-of-the-art multimodal model, CC-Net, which combines RL with supervised learning using behavioral cloning (BC) [4]. This approach demonstrated remarkable efficiency and effectiveness in web navigation tasks, achieving state-of-the-art results on the Miniwob Benchmark, but required a training dataset of 2.4 millions examples. The model leverages the strengths of both RL and SL to optimize performance and adaptability.

Another approach that has shown promise is Workflow Guided Exploration, proposed by Liu et al. [5]. This method incorporates policy constraints to limit the range of possible action types during the reinforcement learning phase of training. By constraining the agent’s actions, the model converges more effectively and achieves better performance while minimizing the number of demonstrations in the dataset.

Older works focused on using more traditional approaches such as traditional neural network approaches and heavily focused on processing the contents they have been faced with, and designed specifically over their environment such as Pasupat et al. [6] who focused exclusively on instruction mapping methods. He et al.[7] outlined the limitations between multimodal environments combining diverse UI representations, Document Object Models (DOM), and action planning from a general utterance.

Multimodality over the observed environment and language comprehension are two difficult areas to combine in a single manner for a navigation task. These previous works managed to identify some of their limitations and others showed greater success, but they still suffer from generalization issues.

The tradeoff between exploration and exploitation is an important concept in reinforcement learning that many of these problems suffer from, requiring a lot of data to converge to an optimal solution. This allows them to obtain great results near human capabilities in the case of CC-Net, but they require large amounts of data large the tasks they are being trained on, and more efficient models need to be investigated.

In figure 1 we can observe the average accuracy of different models over the Miniwob benchmark based on the training technique used, SL, RL, a combination of both, or using few-shots prompting examples.

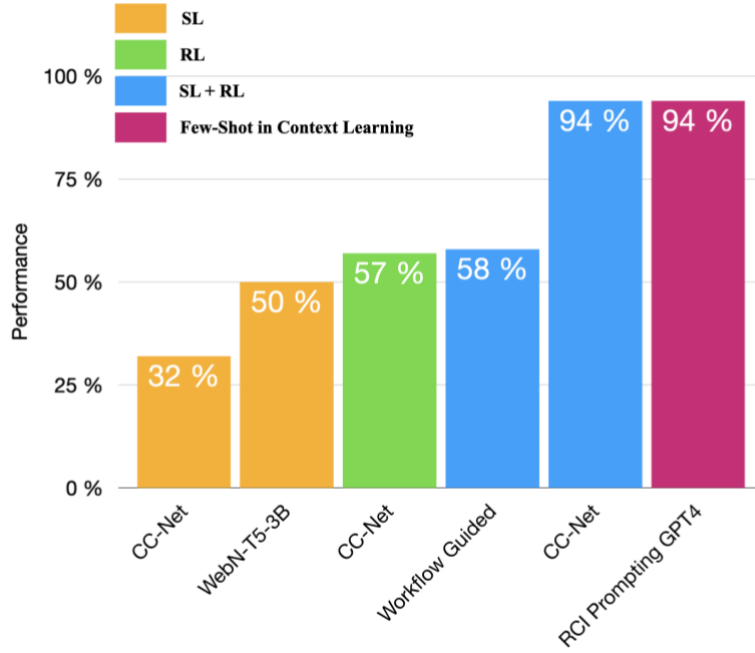


Figure 1: **Comparison of Existing Models Regarding the Web Navigation Task over the Miniwob Benchmark.** Average comparison over the tasks proposed on the Miniwob benchmark between different training techniques and architectures.

2.3 Large Language Models

The first works utilizing LLMs in web navigation tasks were done by Nakano in ‘WebGPT: Browser-assisted question-answering with human feedback’ [8] which leverages a GPT-3 [9] model as a browsing assistant for answers generation. Their main limitation is that they were not using raw web content, but devised a text-based browser that their model could interpret. Their tasks were limited as they were tasked to retrieve information on the web rather than solving general navigation problems, but they opened the way towards integrating language models in browsers. One step forward, Yao et al. in ‘WebShop’ [10] created an online e-commerce environment in order to ground language models in an interactive setting where the agent would be tasked with natural language instruction to devise an action plan to retrieve a desired article from the platform. They were also relaying from processing their environment to mapping it to natural language which heavily restricted the context window of their models, and made them not suitable for different environments.

Others attempted to pre-train an LLM directly over HTML content but suffered heavily in its benchmarking results over classical NLP tasks such as text summarization or question answering as pointed out by Aghajanyan et al. in their work [11]. This severely restricts the use of LLM in the context of web navigation, as a model pre-trained over HTML content will fail at understanding the instructions it is provided with.

One notable breakthrough in the field has been the introduction and scaling of large language models (LLMs), which have expanded the range of applications and target tasks that AI agents can tackle. Gur et al. demonstrated the potential of LLMs in understanding web content with their work on “Understanding HTML with Large Language Models” [12]. They leveraged a bi-directional attention encoder in the T5 architecture [13] and surpassed the performance of

previous supervised learning (SL) approaches by training their model with significantly fewer data (192x less). This highlights the effectiveness of LLMs in comprehending and navigating web pages.

Additionally, the work by Kim et al. showcased the potential of larger LLMs, such as GPT4 [1][14]. Through iterative prompting, they demonstrated that these models can achieve state-of-the-art baselines, further highlighting the importance of LLMs in web navigation tasks in a few-shot manner. We refer to it as 'few-shot' the ability of a model to accurately perform a prediction while being confronted to only a few examples, and by zero-shot when it has never been confronted with the current demonstration it is solving successfully.

Despite these advancements, there is still a need to develop more efficient and adaptable models for web navigation. The existing methods often rely on large amounts of training data, making them less suitable for scenarios where only limited examples are available or when encountering new and unseen web environments. Furthermore, fast inference times are crucial in web navigation tasks, as users expect swift responses and seamless interactions which implies the need for smaller capable models.

3 Methods

Our methodology aims to address the limitations observed in previous large language models (LLMs) applied to the web navigation task using the Miniwob benchmark. As a preliminary approach, we describe the environment and the dataset consisting of recorded demonstrations and its processing steps to make them suitable for our models before moving to the model design phases. In the first stage, we conduct a comprehensive analysis by comparing various sizes of T5-based models. We fine-tune these models using different approaches: initially following existing methodologies and then incorporating hierarchical planning techniques to overcome the identified limitations.

Moving to the second stage, we integrate the best-fine-tuned language model with a multi-modal neural network. This combination allows us to train the model using a combination of supervised learning (SL) and reinforcement learning (RL) approaches while integrating visual input components. By leveraging both SL and RL, we aim to enhance the model’s performance and adaptability in navigating web-based environments.

To gain further insights into the models’ behavior and understand the significance of the inputs, we conduct an ablation study. This study involves systematically removing or altering specific components of the models to observe their impact on performance. Through this analysis, we aim to gain a deeper understanding of the inner workings of the models and gain insights for improvements.

A key aspect of our methodology is to start by assessing the models’ performance at a small scale, identifying and correcting its limitations before integrating additional techniques to attempt to improve them.

3.1 Miniwob++ Benchmark

The Miniwob++ benchmark[5] is a framework for over a hundred different web-based environments. Derived originally from OpenAI’s World of Bits projects, it consists of multiple website demonstrations of varying difficulties in order to simulate a web exploration scenario. We used the thirteen thousand human-made demonstrations from the dataset provided by the Farama Foundation covering interactions recorded by AWS’ Mechanical Turk. This available data allows us to train the models in a supervised manner. Many existing research projects use the Miniwob benchmark, and thus it is a good choice to measure the capabilities of our models using it.

The Farama Foundation also manages gymnasium (GYM), previously owned by OpenAI [15], thus it makes it a suitable environment for introducing reinforcement learning (RL) techniques.

3.2 Datasets

The datasets consist of HTML, Document Object Models (DOM) elements parsed as dictionaries, where each element is attributed a reference number called ‘ref’ which is a unique integer value used to identify this element. It acts as a selector across the DOM tree.

In addition to the DOM, we have mouse interaction data that has been processed to reduce the action space to two elements: ‘click’ or ‘type_text’ also referred as ‘keydown’. Keydowns originally refer to an individually pressed keyboard key and its associated value. Contrary to Humphrey et al [4] we concatenate together the keys of all the adjacent typing actions targeting the same element, e.g instead of having to individually press the keys to type ‘Hello World’ consisting of a total of 11 characters and thus 11 actions, we use the full string as a single action where the string can be embedded from our vocabulary as seen in the original Worlds of Bits [3] as it is easier to create a categorical distribution from the vocabulary than over single-character

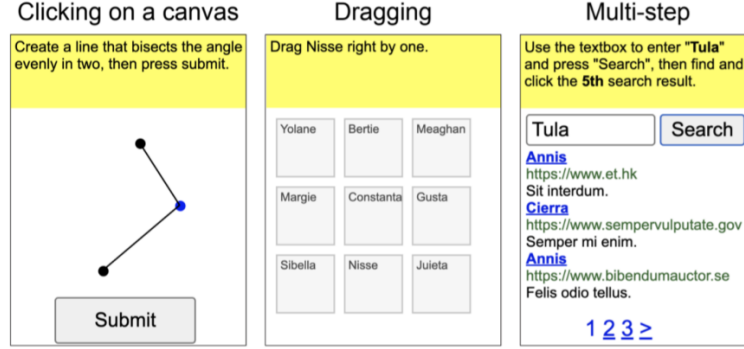


Figure 2: **Example of Miniwob Episodes.** Each opened episode is timed and alongside it, a discounted reward is computed. These episodes cover a wide range of tasks, and in our case, we select a subset of 40 episodes that are suited to work with language models in the fashion of Gur et al. [12].

sequences. Action types such as double-click, single clicks, and other click-based action except dragging or scrolling are turned into single-click actions. Therefore we only use two separate actions in the fashion of Gur et al [12].

Below in table 1 is an example of action data for both types.

history_episodes	utterance	actions	refs	text	complete
DOM tree	click submit	click	5	null	stop
DOM tree	Select Departure City Buffalo	keydown	133	buffalo, NY	continue

Table 1: **Example of episode individual action for the hierarchical dataset.**

Each experiment data from the Miniwob benchmark has a computed, linearly timed reward value that has been recorded along the episodes. We process the episodes to only keep successful ones, with non-negative reward values. Each episode also holds entries regarding the utterance, task name, and time to completion.

The dataset prepared for fine-tuning the different T5 models concatenates the tuples for the action history under the form (action_type, ref, text), the utterance, and the DOM while truncating and padding to obtain a sequence length of 512 tokens from the T5 tokenizer. An example of inputs for an episode can be seen in figures 3 and 4.

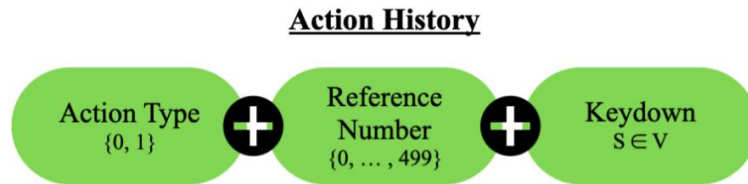


Figure 3: **Structure of Action History as Proposed by Gur et al. [12]**

The T5 model fine-tuned on the hierarchical task has been created by iterating through the episodes and identifying sub-tasks. Each experiment utterance can infer a certain amount of

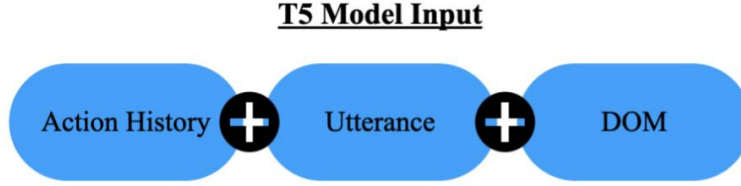


Figure 4: **Structure of T5 Input in its traditional form as Proposed by Gur et al. [12].** The sequence is tokenized before being padded or truncated to fit the 512-long input of the T5 architecture.

1. Infer Model Plan: Devise a plan for the following instruction: 'Click the menu button, and then find and click on the item labelled next'
Subtasks = ['Click menu button', 'find item labelled next', 'click on the next icon']
2. Loop through the subtasks utterances and infer model: {Action History, subtask utterance, DOM} $\rightarrow$ {Action output, reference, keydown string, boolean final state}
3. Perform action of the Miniwob environment: Perform action, check if the episode is terminated, and reward

Figure 5: **T5 Hierarchical Inference Process.** We first infer the model to devise a navigation plan from the initial utterance, then iterate through the subtask instructions individually to infer the current action at each time step while evaluating the state of the episode by means of the computed reward and terminal state.

actions, e.g. "click button ONE, then click SUBMIT" implies two actions that can be separated. Other more complicated actions are processed individually from the original dataset, and the performed actions are mapped to the corresponding sub-tasks. This secondary 'planning' dataset, is used to fine-tune the same hierarchical T5 model in dividing a task into a list of sub-tasks and proposing a plan to solve the navigation problem of the current episode. The goal is to translate the high-level instruction into an actionable plan for the model to follow, and to later improve upon during a RL phase [16][17]. These two last datasets are used to fine-tune a single model for planning and action tasks.

In figure 5, we show the exact classification steps to generate the sub-tasks in a supervised manner. We introduce a boolean indicating whether a sub-task is reaching a final step before moving to the next one.

3.3 Episode Processing for Action History Extraction

To extract the action history while iterating through the episodes, several steps are followed. This section outlines the key steps involved in processing the episodes and gathering the necessary information.

Removing Duplicate Actions

To ensure that the action history does not contain duplicate entries, each action step is checked for duplication with the previous action. If duplicates are found, only the last occurrence is retained under the assumption that it represents the correct action. For instance, when it comes

to typing actions, there may be multiple 'keydown' events. However, it is assumed that the agent directly pastes the text, necessitating only one action.

Additionally, any 'click,1' action corresponding to a click on the `< body >` element is discarded as it is considered unnecessary for the action history.

Keydown Actions for Targeted Elements

For keydown actions, only the last action for each targeted element is retained. To obtain the correct string to be inputted, the 'value' field of the targeted element in the last episode is examined. A function analyzes an instruction and target action to create a target text. Various cases are considered, such as entering time, sending texts or emails, booking flights, guessing random numbers, interacting with textboxes, retweeting or searching for users, switching tabs, responding to messages, and more. If none of the specific cases match, the episode is fixed manually or dropped if lacking clarity over its intentions.

Balancing Tasks

We observe an important unbalance in the distribution of tasks as seen in figure 17. This unbalance can force the model to specialize in a subset of tasks that are over-represented. To remedy this, we down-sample 150 episodes in these task suites for training and testing.

3.4 WebN-T5 Study

We will provide a comprehensive overview of the model architecture, showcasing its workings and how it reflects in the context of the navigation task. Understanding the limitations will set the foundation for our proposed method, which aims to address and overcome these challenges. We will demonstrate how we intend to leverage the model's strengths while mitigating its limitations, ultimately enhancing its performance in the navigation task from a T5-based architecture.

Architecture

The T5 model consists of an encoder-decoder structure. The encoder processes the input text, while the decoder generates the output. It is designed as a "text-to-text" model also referred to as a sequence-to-sequence manner [18], where both the input and output are represented as text. Before passing and retrieving text from the model, it needs to be tokenized with a dedicated tokenizer. A tokenizer is a function that maps the original text into a format readable by the T5 model which is a sequence of tokens. Each token is a sub-representation of a part of the original text, which can be an entire word, parts of a word, or single characters. The T5 architecture takes as inputs 512-long token sequences. If the input does not match this length, it is either padded or truncated. The outputs are reconstructed from the tokenizer, which converts the produced token into plain text.

The model has been pre-trained on a large corpus of text which allows us to reuse the trained weights in order to fine-tune it on a specialized task, here is web navigation where we train the model to predict a string sequence representing an action to perform. The goal as shown by Gur et al., is to recreate their T5 model fine-tuned over the navigation task that they name WebN-T5, and identify its limitations.

In order to train the model, a commonly used metric is the ROUGE [19], used as a loss function as the model is trained in a sequence-to-sequence manner. A loss function computes a signal to backpropagate through the model in order to update its parameters gradually, leading to a performance improvement over the data labels it is confronted with during training.

We fine-tune several models based on the original T5 model architecture [13]. Different sizes are investigated: T5-small, T5-base, and T5-large. Our target labels are composed of the action type 'click' or 'type', the reference number, and the text to type.

Limitations over selector

We could not load the weights of the models from 'Understanding Large Language Models' [12], therefore we attempted reproducing the creation and training steps based on the T5-architecture. An important factor is that the reference numbers attributed to the different elements of the DOM are attributed in an ordered manner. This poses a bias issue as the disposition of important elements of the pages is not uniformly distributed in the generated page or the DOM. Indeed, the most important areas of attention usually lie in the center of the screen, and most pages from the Miniwob benchmark usually follow similar designs. As seen in figure 6, we observe that the distribution of elements of interest does not follow a uniform distribution. This means that the model may train on memorizing these elements [20] based on their location, or even worse entirely memorize the distribution of the salient elements of the page. We divided an experiment where two categories of T5-based architectures are trained, one in the same fashion as the original paper with ordered references, and the other randomizing the references.

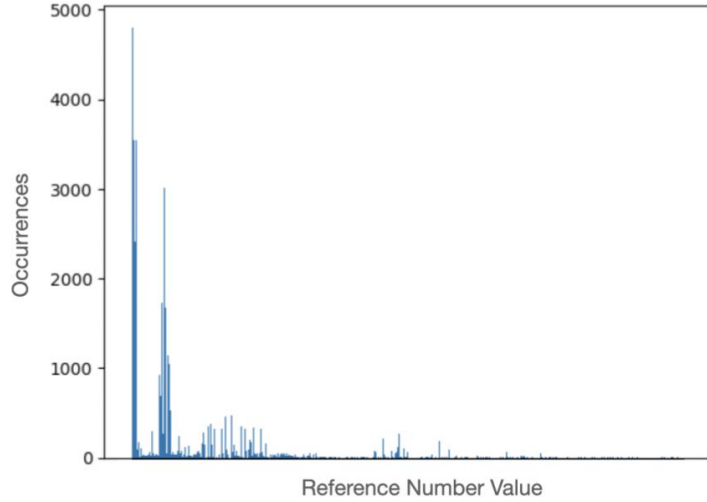


Figure 6: **Distribution of Ordered Reference Numbers in the Recorded Actions.** We can observe that the distribution of the target elements is concentrated among several locations, which can be linked to the salient elements in the DOM and displayed on the page. One of our claims is that previous works focused on learning these distributions by overfitting.

This issue is not only present in the previous language-based models but also in the other attempted models which are not randomizing these references. Humphrey et al. [4] argue against the possibility of their model memorizing every possible configuration of the episodes as being too many possible cases. We argue against it because the distribution of references remains largely unchanged over the randomization of the content in the episodes. Therefore, we re-put into consideration the claim of these models to fully understand HTML content or not memorize every possible configuration, where instead they learn the posterior distribution [21] of these reference numbers. To solve this, we randomize the reference numbers of all of our episodes to

force the models to base their predictions on the element features to mitigate the representation of these tokens.

Agent Gymnasium and Selenium

Two agents were created for the Miniwob benchmark. One uses Gymnasium’s provided RL environment for interactions, and another one is fully based on selenium [22] which can be updated easily with other algorithms for the web. There are no differences between the two, but the fully Javascript one using selenium can run in the background.

Serial Action Attribution vs Randomized

We train a T5-base model in the ordered-reference dataset provided by default in the Miniwob environment. This model is benchmarked over the raw environment a first time to measure its accuracy and a second time where the references of the environments are randomized. We investigate whether the model tries to memorize the target distribution of these references, and if so to which degree.

History vs No History

We performed an ablation study over various sizes of T5-based architectures to study the influence of adding action history. The studied model sizes are T5-small, T5-base, and T5-large, all fine-tuned over the classical navigation problem from Gur et al., and the other over the hierarchical task planning to compare the effects of action history on them.

3.5 WebN-T5 Hierarchical Planning

The goal is to train the agent on lower-level tasks during an episode. As observed on the original WebN-T5 [12], we observe that the agent seems to perform non-optimal actions despite the previously added action history on certain episodes. We propose to fine-tune the model for task division in a hierarchical manner from the original utterance. It breaks down the original task to propose a multi-step navigation plan to solve the problem.

Task Division

Various types of sub-actions can be identified such as clicking a submit button which is represented in the original utterances through the substrings 'Click Submit', or 'Press Submit'. These actions are transformed to be the same ones and saved into the dataset. Other clicks or selection-based actions are also identified in a similar fashion. We defined a general function targeting each Miniwob episode in order to produce a list of sub-tasks based on the observed interactions.

Actions such as 'Enter Time' are clearly separated by linking a typing action to the corresponding checkbox, other utterances such as 'Select' or 'Write' are also ramified with their exact targeted element.

The flight task has a JSON file attributed to the utterance and it is harder to reconstruct the exact action sequence based on the performed actions. We are able to reconstruct the actions by matching them with the target element if they correspond to a departure city field, date, search, destination, and seat selection. All episodes with actions in the reconstruction process that can not be mapped in a straightforward manner are dropped to avoid teaching the model actions that we can not properly match with a sub-task instruction. The Email forwarding episodes are also harder to pre-process into sub-tasks because of the variety of syntaxes in the original utterance.

Eg. "Tiffany wants the email you got from Vickie" or "Look for the email from Edin and forward it to Flora" are hard to divide into sub-tasks. A set of clearly identified sentence structures are selected to be divided into sub-tasks. It is possible that larger pre-trained language models are better at these tasks.

Here are two examples of the translation between miniwob utterances and an action sequence for the hierarchical planning task:

- Example 1:
 - Utterance: "Departure City": "Philadelphia", "Destination City": "Charlotte", "Ticket Type": "Return flight", "Departure Day": 4, "Returning Day": 26, "Passengers": 2
 - Action sequence: Select Departure City Philadelphia; Select Destination City Charlotte; Select the Departure Day to 4;
- Example 2:
 - Utterance: Expand the section below and click submit.
 - Action sequence: Expand the section below; click submit;

The social media tasks were entirely discarded as many episodes contain a single action whereas the utterance suggests multiple. This does not seem right and thus this task was entirely removed from the dataset.

The final dataset is composed of over eight thousand action episodes, and in table 17 we describe the number of examples per task. The second dataset regarding task planning instruction contains 10,960 episodes.

3.6 Language Model with Reinforcement Learning

The best-benchmarked methods over Miniwob are using an RL approach, usually combined with an SL one, but they require a lot of exploration (dataset composed of 2.4 million episodes for CC-Net [4]). We propose combining the previously fine-tuned T5 models over the hierarchical task with CC-Net inspired model to let it improve further in an RL manner. We use a PPO-based algorithm, V-MPO as an improvement loss.

Model Architecture

The model architecture is derived from the original CC-Net [4] which uses a multimodal approach taking additionally as inputs the predictions of the fine-tuned large T5 model, a screenshot of the current environment and the language information being the concatenation of the action history, utterance and DOM elements of the page. We combine it with our best fine-tuned T5-large hierarchical model in order to leverage the pre-trained and fine-tuned capabilities of our models as seen in figure 7.

The screenshot inputs of size (3, 210, 160) are fed through four RESNET blocks of stride 2x2x2x2, kernel size of 3x3 with a respective amount of output channels of (32, 128, 256, 512) in the manner of CC-Net. Language inputs are fed through a size 64 embedding layers using our original vocabulary which is then passed through a single transformer layer with four attention heads and a hidden dimension of 140 parameters.

The outputs of the three input layers are fed into a 2-layers multimodal block with 8 attention heads and 512 embedding dimensions. Two layers have been used instead of the previous 8 to deal with the vanishing/exploding gradient problem originally encountered, with the hypothesis that combining with the previous fine-tuned T5 model would not require as many layers and

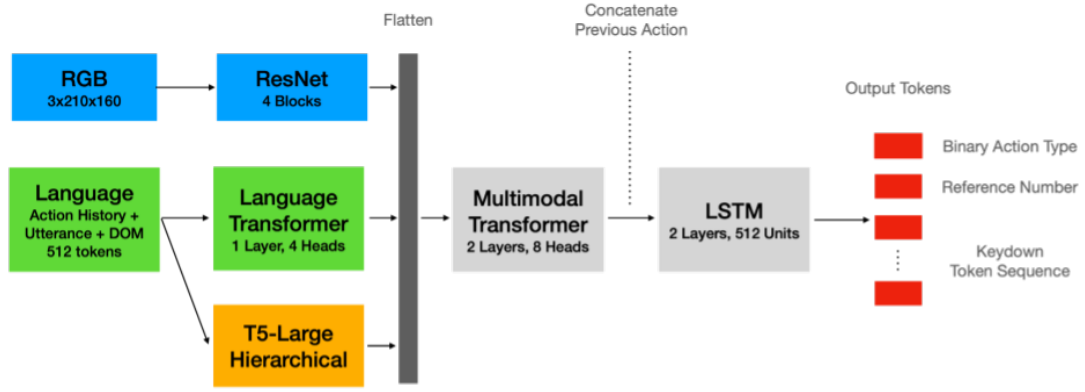


Figure 7: **The Combined Architecture of T5-large fine-tuned over the Hierarchical Task, and CC-Net Multimodal Abilities over an RL Approach.** In the manner of the original CC-Net model, visual inputs are passed through four ResNet layers. The language inputs comprising the action history, utterance, and DOM information are passed through a cross-attention transformer with one layer and four heads, and in parallel with our fine-tuned T5-large over the hierarchical task. The three outputs resulting from these blocks are flattened and concatenated before passing through a transformer block including two layers with eight heads. The output is concatenated with the tensor of the previous inference of the model and passed through a series of two LSTM layers with 512 units. The output tensor of size 10 predicts the binary action type, the target reference number of the action, and a token sequence of a string to predict if the action is of type keydown.

parameters. The outputs of the multimodal transformer are then concatenated with the previous action and fed into a two-layer LSTM block.

The final layer is composed of a binary variable for the action type, followed by a 500-long tensor representing the reference number of the html element, and a 8x1591 long tensor representing the vocabulary indexes of the target words during the keydown actions. The original paper used a size 64 tensor to represent its target tokens that would be indexed against a learnable hashtable. This approach is more compact, but was suffering more of the vanishing/exploding gradient problem during our experiments, therefore we have stayed with a one-hot encoding for this experiment.

Dictionary Mapping

A vocabulary of 1591 word tokens has been created containing all integers attributed to the different HTML elements, all characters from the Latin alphabet, special symbols present in our language, and DOM data. As not every word can fit as a token, we have chosen the sub-word division method in order to arrive at a vocabulary size of around 1500 words.

We use the T5 tokenizer for the T5 model but create a different one for the rest of the CC-Net architecture containing 1591 words. References are constrained in a 500 integer range.

3.7 Online vs Offline Phases

The accuracy of the model can only be measured in an online environment over the Miniwob benchmark. A supervised training approach is not sufficient for the model to learn as shown in WebN-T5-3B achieving 51.6% accuracy being the SL SOTA, while RL combined approaches such as CC-Net surpassing human-level performance. Therefore, it must be combined with an RL approach by alternating between SL and RL improvement phases. We refer to 'offline' as the supervised part of the training based on recorded episodes, and 'online' as the phases of the agent exploring its environment and learning from experience [23]. We alternate between the two phases, where the first offline one uses the already recorded Miniwob episodes, whereas the following offline ones are solely using the successfully recorded episodes during the online RL phases [24].

Gym Environment

Gymnasium offers a reinforcement learning environment over the Miniwob benchmark, with its action observation and action space. In this work, we propose a new preset in the fashion of previous ones [3] [5] [4] where we reduce the action space by retaining only two out of the fourteen available action types, namely 'CLICK_ELEMENT' and 'TYPE_TEXT'. The observation space remains unchanged, except during our processing steps, randomizing the reference number attributes and preparing the data for the different model input requirements.

A discounted reward is computed in relation to the time to completion of the episode. As we are inferring two separate models multiple times during each episode, from servers equipped with GPUs, the inference time can be too low to fully complete episodes requiring a lot of action steps. The majority of episodes have a total available time of ten seconds, we have decided to increase it to thirty seconds to ensure that the model can complete the tasks. Each successful episode produces a positive reward between 0 and 1, while failed episodes return a negative one. This reward helps in training the models in a reinforcement learning manner after the supervised learning phases.

V-MPO

In order to compute the loss of the agent during an online phase composed of multiple action steps, we use V-MPO, a PPO-based algorithm [25] [26]. V-MPO is a reinforcement learning algorithm that combines maximum likelihood policy optimization (MPO) and variational inference [27]. It tries to address the sparse exploration problem, and sample inefficiency in an environment.

Its high-level steps are as follows:

1. Observe environment and collect ‘trajectories’ from current policy (our T5+CC-Net based model)
2. Update a function-value network by minimizing the error between predictions and estimated values from the policy
3. Update the policy (our model) using variational inference from a newfound one that yields a high-estimated return. We estimate the return of the different state actions, perturb the current policy and compare estimated returns with the current policy. Loss is computed from the sum of a policy evaluation and policy improvement losses using KL-constraint to update the former towards the latter after n returns. We create a categorical distribution of each action (type, reference, text) from the output layer of our model for each element.
4. The computed loss is backpropagated through our current policy, becoming the old one for the next iteration. The new current policy to be perturbed at the start of an iteration will be based on the latest update.

The model is trained on all possible tasks in a mixed manner as it showed better results than training on each task individually [4]. We benchmark the model after the first offline phase to test its base architecture in a purely SL manner, then after the full training combining offline and online, we benchmark it again to show its improvements. The parameters used are presented in table 3.

A key modification is the sampling of actions from the created distribution. Indeed, we need to sample an action type, a reference number, and the string sequence in case the action type is keydown. Contrary to the original CC-Net which used a fixed-sized embedding indexed against an embedding table, where we assume they used cosine similarity from the output weights to derive a probability distribution, we used a normalized categorical distribution based on the activation weights of the final layer where each index represents the token position in our vocabulary. In other words, we normalize the activation weights of the tensor segments corresponding to our output types in order to derive a probability distribution from which we can sample actions during policy inference for exploration.

The T5 model is not updated in this approach as our CC-Net and T5 model has been trained in a two-stage process. When using V-MPO, we only update the CC-Net-based architecture which can pose limitations. Ultimately, we see the CC-Net part of this process as an RL boost to the original T5 model but it can pose limitations if the original inference of the T5 is severely flawed.

3.8 Choice of Loss Functions

Language-based, multi-modal, and reinforcement learning-oriented models may require different losses. In our development process, we were confronted to design choices that we detail in this section.

T5's Loss

The T5 model has been fine-tuned using the ROUGE loss metrics [28]. The ROUGE metrics, particularly ROUGE-N and ROUGE-L, measure the overlap between predicted and reference sequences, thereby assessing the model's coverage and fluency in generating appropriate action steps. However, there are certain drawbacks to consider. First, ROUGE metrics are primarily designed for text summarization or translation tasks, and applying them directly to action sequence prediction may not fully capture the nuances specific to web navigation, especially regarding the format of our labels. Additionally, ROUGE metrics do not account for the temporal order and structural dependencies inherent in action sequences[29]. Thus, although ROUGE can offer insights into the quality of generated action sequences, it may not capture all aspects of their correctness and effectiveness.

CC-NeT5's Custom Loss

A main design decision over the CC-NeT5 architecture is the nature of its output layer. As it has been decided to predict a binary action type, followed by a reference number and a string sequence composed of 8 tokens, several loss functions had to be considered and designed. This process was iterative, as we found that some formats of the output layer were making the model converge inefficiently, which impacted the design of the final decided loss function.

Our output was initially designed as follows:

- Action: A binary value represented as a tensor of size 1.
- Reference number: A tensor of length 64, indexed on the vocabulary size through embeddings.
- Keydown text: A tensor of size 8x64, indexed on the vocabulary size through embeddings.

Where both the reference number and the different keydown text were encoded in a size 64 individual tensor representing the vector of the corresponding value indexed in an embedding table. A mean-squared-entropy (MSE) (equation 1) loss function was used for these parts, but after a series of tests, the encoding seemed too sparse for the model to properly converge. That is why it has been decided to update the loss function to a cross-entropy (CE) (equation 2) loss in order to predict the reference numbers and the keydown text by means of a softmax function over each section of the output layer corresponding to the element to predict. Each of these sections has been increased to a size 64 tensor to one of our vocabulary size of 1591 elements. For further efficiency, the reference number final tensor has been reduced to a size of 500, where each index corresponds to the reference number to predict which is more than the total amount of uniquely encountered numbers. These indexes are computed by means of a softmax function. The resulting loss function to process this output is described in equation 3.

The final output layer of CC-NeT5 becomes:

- Action: A binary value represented as a tensor of size 1.
- Reference number: A tensor of length 500
- Keydown text: A tensor of size 8x1591 (8 times our vocabulary size).

The action type tensor is matched with its corresponding boolean value representing a 'click' or 'keydown' action, the reference number index is extracted through the softmax alongside the keydown text token index that is retrieved from our tokenizer.

Figure 8: **Description of Mean Square Entropy (MSE) loss (1), and Cross Entropy (CE) loss (2).**

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2 \quad (1)$$

$$CE = -\frac{1}{n} \sum_{i=1}^n (y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)) \quad (2)$$

Figure 9: **Final combined loss to use in the combination model of T5-hierarchical and CC-NeT to process our final output layer**, where v represents our vocabulary size. The first MSE computes the loss of the action type, followed by the cross entropy of the reference number, and finally by the sum of each CE corresponding to the token to predict.

$$CC - NeT5_{loss} = MSE(t_0, \hat{t}_0) + CE(t_{1:501}, \hat{t}_{1:501}) + \sum_{i=8}^8 (CE(t_{v \times i}, \hat{t}_{v \times (i+1)})) \quad (3)$$

3.9 Naming Conventions of Models

We provide different namings employed for our models throughout this work. The term "WebNT5" refers to the models introduced by Gur et al. in their study, which were fine-tuned using the recorded episodes of the miniwob benchmark. We specify in this paper the specific dataset on which the WebNT5 models were fine-tuned, such as ordered or randomized references, as well as the distribution of these references on which the model was benchmarked. When referring to models fine-tuned for the hierarchical task, we use the abbreviation "T5-Hierarchy" or "T5H." Each time, we indicate the size of both the WebNT5 and T5H models, such as "WebNT5-base" to denote the WebNT5 model fine-tuned from a T5-base model, or "T5H-large" for a T5H model fine-tuned from a T5-large model.

The original multimodal model proposed by Humphrey et al. is referred to as "CC-Net." To combine our best fine-tuned T5 model, specifically T5H-large, with an architecture inspired by CC-Net, we designate it as "CC-NeT5." Additionally, to acknowledge the inspiration from both components, we also denote it as "CC-Net5." CC-NeT5 was trained by combining supervised learning (SL) and reinforcement learning (RL), and we conducted benchmarking at two stages: after the initial SL phase and at the completion of the final RL phase. To differentiate between these results, we explicitly mention whether it underwent the "SL"-only training phase or if it followed the "SL+RL" approach by combining both methods.

4 Results

In this section, we provide an analysis of the outcomes achieved by the various trained models, including the findings from our ablation studies and notable observations. The models are benchmarked over a selection of episodes not necessitating pure mouse cursor movements, but only click and typing actions from the Miniwob benchmark in the manner of Gur et al. [12]. We measure the accuracy of each task over a hundred episodes and define it as the total average accuracy of the global performance of the model over the benchmarked episodes.

Our ablation study investigates the effects of language and visual inputs, as well as action history, allowing us to assess the significance of each input type in the overall performance.

Following this, we cover both the strong and weak points of these models over their benchmarks and compare them with similar existing models.

4.1 Model Performances

The best performing model found was the T5-large trained on the hierarchical task achieving a 43.58% average accuracy over all tasks, which is higher than the T5-base with 39.78%. The combination of T5-large hierarchy combined with the CC-Net-inspired architecture achieved a 36.39% accuracy after its BC (offline) phase. The final accuracy of that model after the RL phase is 33.86%, which is lower than the SL approach and we explain several possible reasons as to why it happens in its related sub-section. We present the average accuracies of our pure language models in figure 10, whereas the combination of SL and RL is described in the subsequent section. The total accuracies of each models over the task from the Miniwob benchmark are shown in table 4, 5 and table 6 in the appendix.

T5-Model Performance

The overall capabilities of the fine-tuned T5 language models are displayed in figure 10 and table 2. We can observe that T5-large fine-tuned over the hierarchical task achieves the best global performance with 43.49% accuracy with its action history, and with it is ablated achieves 43.58% which is not significantly different. T5-base fine-tuned over the hierarchical task achieves 39.77% precision.

More interestingly, the original WebN-T5 model from Gur et al. [12] have been reproduced without randomizing the references on the training data, and benchmarked both on ordered, and randomized references. We found a decrease in performance from 35.25% to 24.19% which implies that the model is memorizing the distribution of reference numbers from its training data in its parameters. To prove this claim, we re-fine-tuned the same original T5-base model with randomized references, and we see that the model is able to not only recover its performance but also beat the original WebNT5-base fine-tuned over ordered references, showing the importance of our randomization process.

We found small noticeable discrepancies between the performances of the different model sizes from the fine-tuning, mostly related to tasks requiring extensive comprehension of natural language tasks, such as 'click-checkboxes-soft' requiring the agent to select words similar to the ones given in the utterance. We observe that larger models perform better at these tasks due to the pre-training capabilities in text comprehension, such as in the experiment of 'click-checkboxes-transfer' where T5-large for the hierarchical task achieves 47% precision, and its T5-base equivalent is much lower at 15%. Another example showcased in figure 12 called 'click-checkboxes-soft' asks the agent to select words similar to the ones from the original utterance. We observe that larger models such as T5-large are more efficient at this task with an accuracy

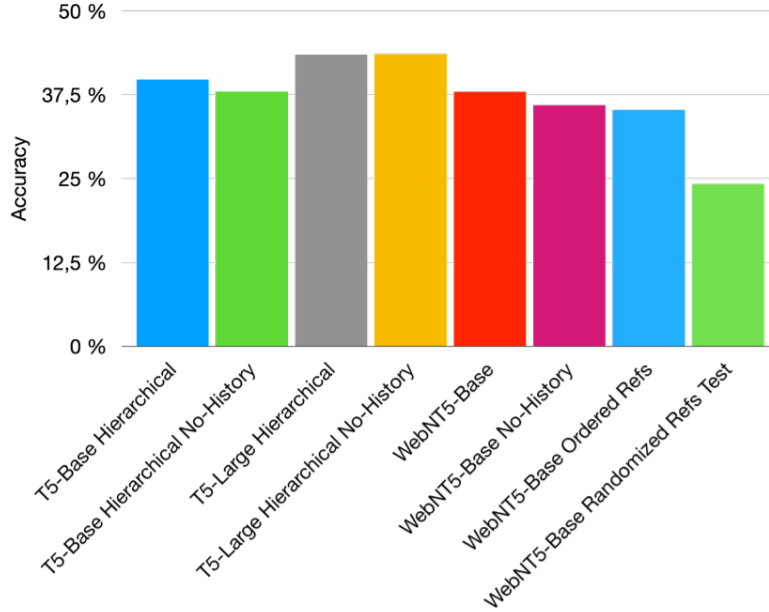


Figure 10: **Average accuracies of the different fine-tuned language models benchmarked over the Miniwob benchmark.** We first reproduced WebN-T5 trained over episodes with ordered references in the fashion of Gur et al. [12], and benchmarked it over Miniwob a first time with ordered and then randomized references and we observe a significant drop of performance when the references are randomized during testing, implying that the model is memorizing the posterior distribution of the references to predict; which was our hypothesis illustrated from the distribution of references in the original episodes shown in figure 6. Then WebN-T5 has been tuned over randomized references and we benchmarked it with and without the history of its actions and we found better results than on the ordered reference dataset. Finally, T5 has been fine-tuned over the hierarchical task and achieved better accuracies. We also observe that the model size plays an important role in the final average accuracy, and is less affected by the removal of action history due to the task division properties of the hierarchical planning.

of 21% compared to smaller models fine-tuned on this task such as T5-base with 8% accuracy. This shows that larger models are better at leveraging their pre-trained abilities over these tasks. Other tasks requiring less pre-trained capabilities have their model achieve less discrepancy between their results among all architectures, but we still observe an advantage towards larger models.

Other interesting examples such as 'use-spinner' illustrated in figure 14, have the models fail drastically. In this task, the agent is tasked with manipulating a spinner up and down to place it on the desired number, and can only do so by clicking an 'up' or 'down' button. Most models achieve 0% on this task which can be explained by the poor capabilities of language models in creating representations of their environments, and truly 'understanding' them.

Issues with original papers

The reproduction of the original work by Gur et al. [12] led to varying results. We did achieve high scores on most tasks meaning the reproduction was close to their work, but we could not reproduce the claim that their model could understand HTML over the web navigation task. Different model sizes have been fine-tuned following their method where the reference numbers are not randomized. The benchmark results are close to theirs, but when the reference numbers are shuffled we observe a significant drop in performance from 35.25% to 29.19% average accuracy. This shows that their original model tends to memorize the distribution of certain reference numbers based on the location of the page, and not on the likelihood of the element to be targeted.

A few Miniwob tasks show that claim is the 'click-button-sequence' task that achieves a 100% accuracy on the WebN5 models, but obtains 8% accuracy when the references are shuffled as seen in table 4. Our model recovers to 100% accuracy when trained on randomized data. Other tasks showing similar results are 'click-button' with 100% accuracy on the original model, then 61% when randomizing the references to finally recover at 76% when trained on randomized references.

A couple of tasks are notable in proving that claim, most notably 'click-color' which asks the user to choose the corresponding color out of the four displayed, where both models achieve respectively 25% and 21% accuracy for being tested on ordered and randomized references. This shows that the model does not perform better than random in selecting one out of four possible actions, and part of the 4% deviation from the expected 25% accuracy is explained by selecting unrelated elements on the page. The original model by Gur et al. [12] achieves a 25% accuracy while our best fine-tuned directly on randomized references scores 61% accuracy.

In short, the original model trained on ordered references seems to memorize the distribution of target elements rather than 'understanding' the environment as previously claimed. This issue is actually common in some of the previous works that are sensible to over-fitting such as CC-Net's model architecture.

4.2 History vs No History

The different models followed an ablation study over their action history to determine their importance, where the results are presented in both figure 10 and table 2. The T5-base model trained on random references sees an almost two points loss in performance when its action history is removed from 39.77% to 38.02%, Whereas the models fine-tuned over the hierarchical tasks are more resilient, with no difference for the T5-large (43.49% accuracy with action history, and 43.58% without) as seen in table 4 between the tasks. This can be explained by the model dividing its sequence action from the start and not needing to 'remember' in which state it is currently as it is already following that plan.

Click on the **white** colored box.

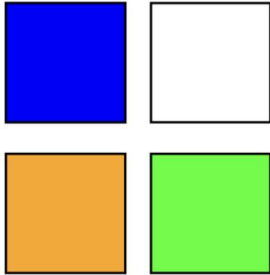


Figure 11: **Episode example of choosing one out of four colors displayed.**

Select words similar to unwise and click Submit.

- ☐ launch
- ☐ wicked
- ☐ stupid
- ☐ furious
- ☐ weird

Submit

Figure 12: **Episode example of checking boxes of words related to the one given in the original utterance.**

Figure 13: **Example of Miniwob episodes.** On figure 11, the episode consists in choosing one out of four colours displayed. The inner DOM properties of the colors are using RGB or CSS color short names such as 'light-blue', 'cyan' to express a color. Previous work in WebN-T5 has a 25% success rate which is the random expected value, while our T5 approach by randomizing the reference numbers during training achieves 50% accuracy in recognizing the right color and action, and up to 61% for our T5-base fine-tuned over the hierarchical task.

In the second example in figure 12, the agent is required to select the boxes related to words matching the one given from the original utterance. This task requires language knowledge and we observe that larger models fine-tuned on this task such as T5-large achieve better results (21% vs 8%) than smaller sizes such as T5-base.

These two examples showcase the importance of these models over language comprehension tasks, and when coupled with reference randomization, the model avoids memorizing the target references and leverages its pre-trained capabilities.

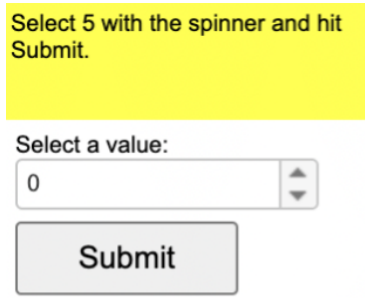


Figure 14: **Use Spinner to a desired value and click submit..** In this task example, the model is not able to derive the given utterance by manipulating the spinner to the desired value and hitting submit. Multiple actions are required which in the best-case scenario are equal to the desired number to input, and the model should be aware of when to stop and submit the result. This is a good example of the representative limitations of language models over their environments.

The model trained in the manner of WebNT5 also sees a loss of around two points for WebNT5-base from 37.94% to 35.99% accuracies. The original results by Gur et al. [12] found more important losses averaging at 6.4% on their models.

We can conclude that larger models are more resilient to the removal of action history, but this is partly explained by the nature of their fine-tuning where the hierarchical task is already dividing an action sequence to follow. It also seems that the randomization of references plays a role where the model becomes less dependent on the previous sequence of predicted numbers.

4.3 Hierarchical Planning Improvements

Fine-tuning a model over the hierarchical task achieved higher results than the compared WebN-based models. The following T5 sizes have been fine-tuned which are T5-base and T5-large, achieving respectively 42.1% and 43.49% accuracy. The original WebN achieved 46.4% accuracy over WebT5-large, and the reproduced T5-base size achieved 38.1% which shows that hierarchical planning is an important component of these models.

The ablation study over the action history outlined a 14% decrease in performance over the T5-base model for hierarchical planning and 0.2% over the T5-large. This is much less than the reported 6.4% reported by the original WebN-T5-large and WebN-T5-3B models showing that hierarchical planning is less sensitive to the action history as it tries to solve the episode by following an original plan step by step.

Nonetheless, we do observe significant rates of failures in complicated episodes, or when the environment changes as the agent follows the original plan as seen in the full benchmark table 4 over the fine-tuned T5 models.

4.4 Performance of Combining T5 and CC-Net

In this part, we cover the results of training the model inspired by CC-Net combining the output of our best T5 model being the T5-large fine-tuned over the hierarchical task. We first present the results over the initial supervised learning phase before covering the ones from the reinforcement learning phase, while showcasing the results of our ablation study over its inputs.

Results of Supervised Learning Phase

The initial performance of the model after the initial supervised phase achieves an average score of 36.69% which is significantly lower than our best T5-large fine-tuned over the hierarchical task. We observe that this drop is mostly explained by a failure over the 'enter-text' task. This can be explained by the keydown action not writing the correct text into the field, or sometimes predicting a clicking action instead.

Three types of ablation experiments have been performed. The first one covers removing the concatenation with the previous action before passing the tensors into our dual-LSTM layers where the models achieve an average performance of 24.86% accuracy. The removal of visual inputs from the model achieves a 23.69% accuracy. Surprisingly, the removal of visual elements achieved better results on some tasks than the original model which is intriguing, notably over the click-scroll-list task where it achieved 34% and the original one 0%. The final ablation study over this model consisted in removing the output from the T5 model which led to an average score of 0% which shows that the model is totally dependent on the language model it has been combined with.

The differences between the different ablations show that even if the model is mainly dependent on the T5-large fine-tuned over the hierarchical task, it did learn a multimodal representation of the environment by means of visual inputs and action history.

Results of Reinforcement Learning Phase

The average accuracy after the RL training phase is 33.86% which is lower than the SL approach at 36.69%. The ablation study revealed similar results as the SL phase, in which the ablation of visual inputs reduced the average model accuracy at 22.32%, of the previous action at 24.30%, and finally of the fine-tuned T5-model over the hierarchical task at 0%. The last ablation showed that the model remained heavily reliant on the language model during the training using reinforcement learning.

There could be several reasons why the RL baseline model may result in an overall lower performance compared to the original SL phase. The first one is related to the nature of the model itself. Indeed, in figure 15 we observe that the model is totally dependent on the T5-large fine-tuned over the hierarchical task, implying that the other weights are calibrated to work around it which is itself a difficult task towards improving the original results. Hyperparameter tuning is also an important aspect of it, and the performance of our model with RL is very sensitive to it. Even though we based our parameters on Humphrey et al. [4], we brought some modifications as shown in table 3 which are more fitting to our own case. The learning rate is much smaller as we used a single batch.

The nature of the RL environment plays an important role as well. RL involves a trade-off between exploration and exploitation. During RL training, the agent needs to explore different actions and learn from the feedback it receives. This exploration phase can lead to lower performance initially as the agent tries out different strategies. It may take time for the agent to learn the optimal policy through exploitation of the learned knowledge, and in the case of Miniwob the total amount of actions to sample from can be very large and noisy. Some episodes such

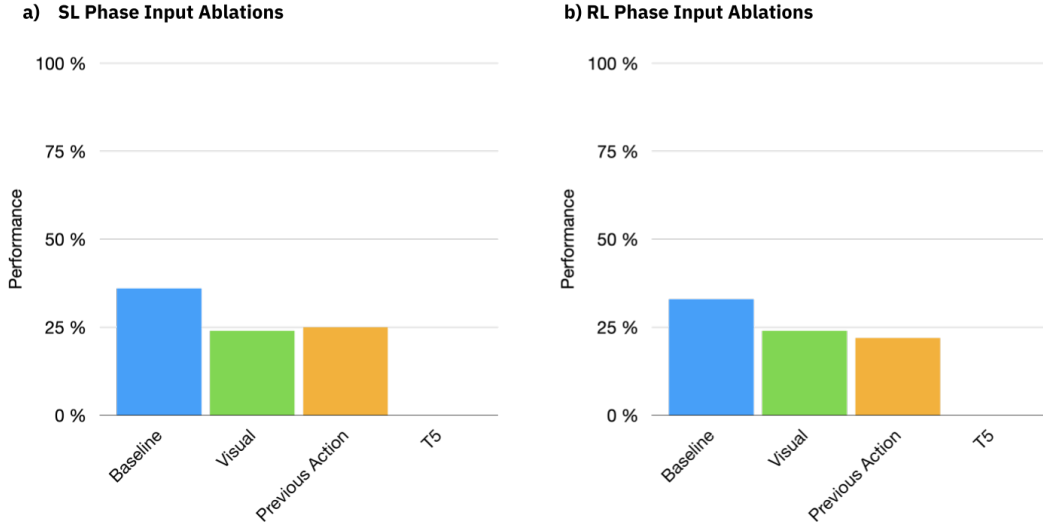


Figure 15: **Input Ablation of T5-Large Hierarchical and CC-Net inspired architecture.** a) ablations after the supervised-learning phase, b) ablations after the multiple offline and online phases using RL. We can observe that in both cases, the model is totally dependent on the outputs of the T5-large model finetuned over the hierarchical task. Nonetheless, the baseline model is also dependent on its visual and previous action concatenation inputs which shows that the model learned a multimodal representation of the environment in solving tasks over Miniwob.

as 'click-checkboxes-large' require a large number of actions, and the exploitation becomes an important hurdle to overcome in this episode, or in 'use-spinner' to cite a few.

Importantly, RL training introduces a feedback loop that can cause a shift in the data distribution, known as covariate shift. The shift in data distribution between the SL and RL phases can lead to discrepancies in performance. As we sample possible actions through a categorical distribution that has been defined during the SL phase, it is possible that the model is unable to derive an efficient distribution and performs worse actions. In addition, it is possible that the model did not converge enough to a stable policy due to insufficient experience.

These factors highlight the complexity of RL training and the various challenges involved. It's important to carefully analyze and address these factors to improve the performance of RL-trained models.

4.5 Benchmarking Results

The performance of the model has been measured in terms of accuracy over the success rate of the completion of a subset Miniwob tasks. A total of 40 tasks out of the available 80 have been benchmarked, whereas the discarded ones consisted of tasks requiring continuous actions or manipulation of mouse-cursor coordinates.

We have compared the amount of data used per task during our training according to our initial processing steps with previous works reported results. We observe close performance with the previous best SL methods while using much less training data per episode. Nonetheless, the result regarding the RL approach are using much less data as well, but shows worse results implying that the RL method requires more training as seen in figure 16.

A key part of this work is the focus on the distribution of reference numbers. As found in our results, the previous models tended to memorize the output distribution of these target elements in their own benchmarks. This does not translate well to their capacity as they memorize more than they ‘understand’ as previously claimed. Therefore, our benchmarking results, despite being in some cases lower than the ones in these previous papers, can be believed to be more robust over their performances and can be used as a more reliable benchmark.

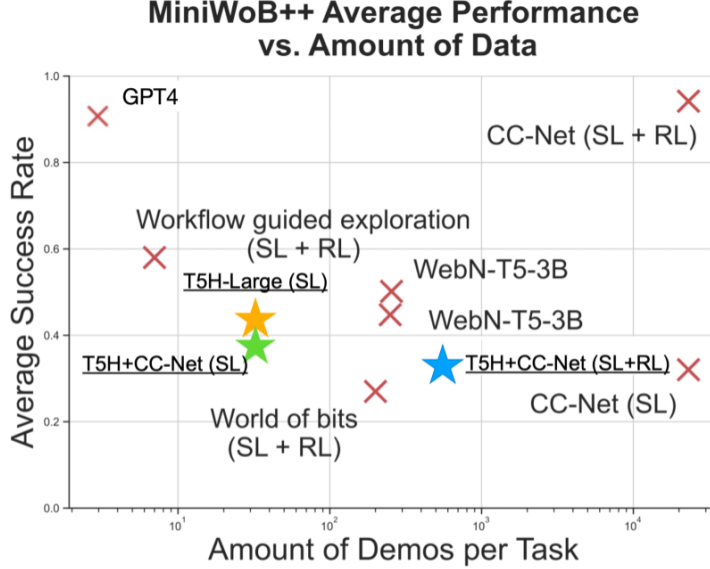


Figure 16: **Average performance of different previous models with ours.** The best T5-large model fine-tuned over the hierarchical task is represented by a yellow star, in the green star the same model coupled with a CC-Net-inspired architecture trained in an SL manner, while the blue star represents the same model trained by combining SL and RL. The original T5-large was fine-tuned using only an average of 10^{1.5} tasks which is much lower than other SL approaches for similar results. Comparison results are borrowed from Kim et al. [1].

Server setups

The gym environment computing the V-MPO algorithm is run on an M1 silicon computer, and the fine-tuned T5 model with its CC-Net variant is run on a server equipped with an RTX 4090 GPU. The V-MPO samples from the policy by performing requests to that server for the different parts of the model inferences, and also to back-propagate the loss and update the old policy with the new one. This dual setup allows to leverage of an efficient backend to sustain these models, and a classical ‘laptop’ environment to run a web browser with Miniwob.

5 Discussion

5.1 Pre-training and Fine-tuning

Training on task

Previous works trained their models exclusively over the Miniwob benchmark. This makes them extremely limited in transferring their capabilities over unseen sites. Furthermore, the Miniwob benchmark is composed of a small DOM tree, and its rendered dimensions do not exceed pixel dimensions of 210x160 pixels. It is therefore impossible for them to complete successfully any task from a real website, which could be more ambiguous than the one encountered in this environment. Another notable aspect of previous models is their tendency to memorize specific task patterns rather than truly understanding the underlying principles. Consequently, the transferability of CC-Net to different frameworks or contexts is limited. While ours and previous models demonstrate proficiency within the Miniwob environment, their abilities to adapt and generalize to novel situations outside of this training domain remain a challenge. This highlights the importance of exploring alternative approaches that prioritize a more comprehensive and transferable understanding of web-based tasks.

Pretrained Language Model

Pretrained language models possess good transferring skills as they have been trained over large amounts of text data, enabling them to capture some level of linguistic pattern and semantic relationships. This makes them beneficial for understanding and interpreting new environments like HTML. However, their out-of-the-box performance may not be sufficient, and fine-tuning is often necessary to achieve optimal results, but they become limited by several factors:

The first main limiting factor is also their input context window, which is suitable for small pages in the Miniwob benchmark, but is not sufficient over most real-world websites. Therefore, T5 models are not the right option in scaling these models over more complex domains, but their bidirectional attention encoders are to be inspired in choosing larger context models.

A second one is their tendency to memorize too much and therefore they tend to over-fit their training data too much. This is an important hurdle in considering language models as suitable architecture regarding a navigation task.

Thirdly, language models often hallucinate in their predictions. Our fine-tuned models sometimes predict action numbers that are not making sense and sometimes provide a reference number that does not exist on the page in the first place. Worst, our models fine-tuned over the hierarchical task are very sensitive to the format of the input instruction in the planning phase and can predict a string sequence composed of punctuation characters of tasks not bearing any meaning. Inferring actions for the model to perform from this leads to catastrophic outputs that are unusable for our selenium agent. Therefore we argue that it is important to consider solutions in constraining the outputs that these language models can produce, such as using a guided-workflow approach as used by Liu et al.[5], or exploring alignment solutions regarding large language models in the context of web navigation tasks.

Fourthly, it presents some limitations and bad zero-shot capabilities depending on the model's size. If fine-tuning is needed, the model will probably have bad shot capabilities over unseen formats if trained on a single task.

Finally, fine-tuning already pretrained model allows to retain semantic understanding and improve it over a specific task, here web navigation. Coupling language, HTML, and agent acting properties in one is quite powerful, but it is important to be aware of the presented limitations when using such architectures.

Boosting finetuning with RL

We used the `miniwob++` environment over the gymnasium’s module as it has been previously used in many works, the first of them Worlds of Bits by OpenAI in 2016. This work such as others have implemented their own presets for the gymnasium module, and we proposed an additional one made for our context. The environment supports 13 different action types that we reduce to 2: ‘`click_element`’ and ‘`focus_element_and_type_text`’, in the same fashion as Gur et al. Some tasks can not be completed in that manner, and also because we don’t support mouse coordinates in our model. These tasks are UI-based actions such as dragging a component or using built-in widgets such as a wheel.

This limits the number of tasks that can be performed but is still a good basis to test the capabilities of the model over ‘understanding’ computer tasks for the web. `Miniwob++` does not represent the entirety of the web, and training an agent using RL on the open web is a tedious task as there does not exist any robust reward procedure for unseen pages. Other methods such as RLHF [30][31] could offer this possibility with the necessary feedback.

New found baseline

Even if our results are in some cases lower than previous works, they are the first to identify several limitations and attempts at correcting them. The objective is to solve the Miniwob benchmark suite while keeping in mind the paradigm of automating web navigation consisting of good generalization over unseen environments. Previous works achieve better results but we showcased they are memorizing too much, especially the target distributions of reference numbers. Our current results are more conservative, but they better reflect the current capabilities of these models in performing web navigation tasks. We are confident that keeping these limitations in mind will lead to better results in the subsequent approaches.

5.2 How well can agents trained on user interactions navigate different web interfaces?

The models have limited capabilities in completing computer tasks, and even more in ‘understanding’ HTML code as previously claimed. They tend to memorize a lot and often fail when the environment changes such as shuffling the elements displayed. Their capabilities are constrained to the environment they have been trained on. RL with V-MPO helps mitigate the exploitation/exploration problem if not enough data is present for SL-only approaches, but is data intensive and suffers from similar memorization problems if the data is not sufficiently processed against biases. Combining pre-trained language components allows for faster results, alongside designing hierarchical planning abilities, but their integration remains tedious as they need additional RL optimization phases to deepen the fine-tuning which is computing intensive if these models are large. Best capabilities still lie in very large pre-trained models and prompting methods, but it leaves a gap for intermediate models to be investigated.

How well can be evaluated the performance of these agents?

The agent’s capabilities are sensitive to the environment they have been trained in. The accuracy in task completion for specific environments is a good metric, but the case for attacks such as randomizing elements in the DOM or visual page should also be monitored and offers additional insights. The paradigm of web exploration tasks lies in zero-shot abilities over unseen environments. Small models fail at these tasks while prompting techniques for very large models show promising results. Multi-modality properties of the models are important, and many

tasks and architectures are not yet being investigated, such as handling mouse cursor, dragging, long-horizon planning, etc. They should also be included as a metric. The robustness of these agents to change of their environment is an important next step towards more specialized models for computer tasks (from existing pre-tuned models) with generalization abilities instead of memorizing.

5.3 Chinchilla’s Problem over Navigation Task

An important question is what is the right size of models to use. Combining a pre-trained model with RL can save training time over language tasks, but still required extensive training to learn the latent representation of image input and self-attention mechanism in the multimodal layers. The covered models have limited capabilities and do not ‘understand’. Their sizes are relatively small compared to larger language models such as GPT4 which can have *some level* of understanding [32], and generalize much better. The question of optimal model sizes, with respect to the number of tokens and training time, is not solved for this area, and can be improved upon with the help of scaling laws practiced in the training of Chinchilla by Deepmind, Hoffman et al. in ‘Training Compute-Optimal Large Language Models’ [33]. They define scaling laws surrounding a computing budget with respect to a model’s total amount of parameters and the number of training tokens to model a final loss function. In the context of web navigation requiring a lot of inference, smaller models are to be favored but should be sufficiently large to capture the difficulties of navigation.

One of the main takeaways from this thesis is that current small models are sufficient for automating task-constrained environments with a training dataset, and can run on a single machine or GPU. They are not suited for more complex or unseen tasks which favor larger language models such as GPT4 or PaLM, but these last models are arguably too large for the tasks they solve, and the question of intermediate models remains open.

Our approach consisted in combined a T5-model with a multimodal approach, with a number of tokens arguably too large for the language task, whereas the CC-Net part only used a vocabulary of 1591 tokens. This discrepancy is not optimal and is important to move towards a unified tokenization allowing the same amount of tokens overall with a quantity suiting our tasks. Finding a general model architecture, multimodal with flexibly large input sizes, combining a more efficient balance between the total amount of parameters and training is a good basis for future work. Starting with smaller models and identifying their limitations was a good place to start, but several areas are now to be further improved before moving to larger models.

5.4 Proposition of Techniques to Improve the Training of such Models

From our results, we can propose a series of techniques to include in the training process in order to improve these models. The first one consists as we did in randomizing the reference elements of the DOM tree in order to force the model to not memorize the distribution of these numbers but instead focus on adjacent elements. This should force the model to base its decision on the actual content of the elements. Secondly, we propose augmenting the dataset by creating new episodes based on shuffling elements of the DOM tree such as inner tag fields, or changing the location of these elements in three. The action sequence should not be affected in most episodes, and therefore the model will be forced to learn a deeper representation of the data in its prediction. The addition of new examples could also help in balancing episodes that are under-represented in the dataset and this could significantly improve the models trained in a supervised manner. Thirdly, we argue that training the model directly on ablated inputs is an additional technique to make the model more accurate and robust in its predictions. The performed ablation studies

showed important discrepancies, even over tasks that the model does not require certain input types. The reason for this drop is that the model became too sensitive in its changes of inputs. Therefore, we think that the model could learn a better representation of the task if trained directly on ablated outputs, which at the same time would be an augmentation of the dataset alongside the previous points.

5.5 Ethical Considerations

Web environments offer the largest and most accessible quantities of data, but the rise of language models over this domain poses legal and ethical ramifications over their misuse. We will cover some aspects that should be carefully taken into consideration regarding how these models are used.

Privacy and Data Protection

Web navigation often involves accessing and interacting with user data, which may include personal information. It is essential to ensure that the automation process respects user privacy, complies with data protection laws, and obtains user consent when necessary. Measures should be in place to handle data securely, minimize data collection, and prevent unauthorized access or misuse.

Impersonating

Large language models are able to mimic human behavior from the dataset they have been trained on, and are already able to fool many humans. The Turing test [34] is not sufficient to differ a robot from a human in most online environments such as when using social media, and it is possible that in the future many of our interactions may be with sophisticated bots impersonating humans. This raises questions in regards to regulations, and data protection law, and adds a new layer to our senses of reality [35] that we must carefully account for.

Accountability and Liability

Determining liability can be complex when AI systems automate web navigation. It is important to establish clear lines of accountability to address potential harm or errors caused by the system's actions. They could for instance infringe copyright regulations or not abide by policies such as GDPR [36]. These rules are different across countries and it is important to define proper licensing agreements if these systems are distributed to define the liability of their users and distributors.

6 Conclusion

Automating web navigation tasks can be useful for many but it is important to understand the current limitations of these models tackling this domain. A new baseline of results and evaluation is presented in attempting to mitigate them, but many improvements are still required to create these specialized models, efficient at inference time and robust to unseen environments.

We answer our initial research questions by first confirming that agents trained in user interactions can solve web navigation tasks, but are largely limited by their modeling capabilities over many tasks. Language models are a powerful contribution, allowing them to be fine-tuned and leverage their pre-training capabilities over performing actions and designing navigation plans. We arrived at the conclusion that these models do not understand HTML as previously claimed, but their language abilities allow them to overcome a range of tasks that were previously solved to some level only through memorization.

The second research path focuses on evaluating web navigation models trained and benchmarked on the Miniwob environment. These models are specialized for specific tasks and pages within the framework and lack generalization abilities and sufficient context windows to capture real-world website information effectively. We contend that benchmark results from the Miniwob benchmark alone are inadequate for assessing model quality since previous works often achieved these results through memorization rather than generalization. Hence, it is crucial to incorporate additional metrics like model resilience to environmental changes and the ability to complete tasks on unfamiliar websites alongside an original navigation benchmark.

The best model architectures for web navigation remain an open question. However, this work emphasizes the effectiveness of using pre-trained language models due to their language understanding capabilities and their ability to be fine-tuned with relatively low computational requirements and data. These models should be combined with other architectures that support multimodality, such as processing visual inputs and handling various actions like mouse cursor movements. Currently, larger models outperform smaller ones but are computationally expensive, while smaller models are efficient for fast inference and can solve most tasks despite being significantly smaller in size. We propose that exploring intermediate sizes of multimodal models could offer better opportunities. In conclusion, large models excel in few-shot scenarios, while smaller models are more suitable for known environments and can accomplish tasks within a lower budget.

Understanding the limitations of these models is crucial. There have been claims that LLMs and neural network-based models can "understand" HTML or web navigation tasks, but we have found these claims to be false or exaggerated, especially for the largest existing models. While these models possess some level of understanding, they still struggle with memorization and hallucination issues when it comes to navigation tasks. We have demonstrated that these issues can be alleviated to some extent through pre-processing techniques, but further efforts are necessary to enhance and scale these approaches for more capable models.

We must also address the risks of misusing this technology for automating interference schemes on the web, impersonation, spreading false information, or engaging in nefarious activities. These pose significant threats to the safety and integrity of the web. This challenge extends beyond policymakers and organizations and also affects generic users, who can be harmed or negatively influenced as a result.

In conclusion, the field of automating web navigation through artificial intelligence holds great promise and has witnessed significant advancements in recent years. This work contributes by shedding light on crucial limitations and presenting techniques to mitigate them, as exemplified in our models. By addressing these challenges, we pave the way for more robust and effective approaches in the future.

7 Acknowledgements

Many thanks to Dr. Mirela Popa and Dr. Jerry Spanaskis (G.) for their precious supervision in this thesis during my final master's semester at Maastricht University, and for the compelling courses over the years.

Sincere gratitude is given to the cited works, which added me reasons to pursue a future in this field.

Bibliography

- [1] G. Kim, P. Baldi, and S. McAleer, “Language models can solve computer tasks,” 2023.
- [2] OpenAI, “Universe,” 2016. [Online]. Available: <https://github.com/openai/universe>
- [3] T. Shi, A. Karpathy, L. Fan, J. Hernandez, and P. Liang, “World of bits: An open-domain platform for web-based agents,” in *Proceedings of the 34th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, D. Precup and Y. W. Teh, Eds., vol. 70. PMLR, 06–11 Aug 2017, pp. 3135–3144. [Online]. Available: <https://proceedings.mlr.press/v70/shi17a.html>
- [4] P. C. Humphreys, D. Raposo, T. Pohlen, G. Thornton, R. Chhaparia, A. Muldal, J. Abramson, P. Georgiev, A. Goldin, A. Santoro, and T. Lillicrap, “A data-driven approach for learning to control computers,” 2022.
- [5] E. Z. Liu, K. Guu, P. Pasupat, T. Shi, and P. Liang, “Reinforcement learning on web interfaces using workflow-guided exploration,” in *International Conference on Learning Representations (ICLR)*, 2018. [Online]. Available: <https://arxiv.org/abs/1802.08802>
- [6] P. Pasupat, T.-S. Jiang, E. Liu, K. Guu, and P. Liang, “Mapping natural language commands to web elements,” in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Brussels, Belgium: Association for Computational Linguistics, Oct.-Nov. 2018, pp. 4970–4976. [Online]. Available: <https://aclanthology.org/D18-1540>
- [7] Z. He, S. Sunkara, X. Zang, Y. Xu, L. Liu, N. Wichers, G. Schubiner, R. Lee, J. Chen, and B. A. y Arcas, “Actionbert: Leveraging user actions for semantic understanding of user interfaces,” 2021.
- [8] R. Nakano, J. Hilton, S. Balaji, J. Wu, L. Ouyang, C. Kim, C. Hesse, S. Jain, V. Kosaraju, W. Saunders, X. Jiang, K. Cobbe, T. Eloundou, G. Krueger, K. Button, M. Knight, B. Chess, and J. Schulman, “Webgpt: Browser-assisted question-answering with human feedback,” 2022.
- [9] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, “Language models are few-shot learners,” 2020.
- [10] S. Yao, H. Chen, J. Yang, and K. Narasimhan, “Webshop: Towards scalable real-world web interaction with grounded language agents,” 2023.

- [11] A. Aghajanyan, D. Okhonko, M. Lewis, M. Joshi, H. Xu, G. Ghosh, and L. Zettlemoyer, “Htln: Hyper-text pre-training and prompting of language models,” 2021.
- [12] I. Gur, O. Nachum, Y. Miao, M. Safdari, A. Huang, A. Chowdhery, S. Narang, N. Fiedel, and A. Faust, “Understanding html with large language models,” 2023.
- [13] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, “Exploring the limits of transfer learning with a unified text-to-text transformer,” 2020.
- [14] OpenAI, “Gpt-4 technical report,” 2023.
- [15] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, “Openai gym,” 2016.
- [16] S. Branavan, H. Chen, L. Zettlemoyer, and R. Barzilay, “Reinforcement learning for mapping instructions to actions,” in *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*. Suntec, Singapore: Association for Computational Linguistics, Aug. 2009, pp. 82–90. [Online]. Available: <https://aclanthology.org/P09-1010>
- [17] A. Vogel and D. Jurafsky, “Learning to follow navigational directions,” in *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*. Uppsala, Sweden: Association for Computational Linguistics, Jul. 2010, pp. 806–814. [Online]. Available: <https://aclanthology.org/P10-1083>
- [18] I. Sutskever, O. Vinyals, and Q. V. Le, “Sequence to sequence learning with neural networks,” 2014.
- [19] K. Ganesan, “Rouge 2.0: Updated and improved measures for evaluation of summarization tasks,” 2018.
- [20] N. Carlini, D. Ippolito, M. Jagielski, K. Lee, F. Tramer, and C. Zhang, “Quantifying memorization across neural language models,” 2023.
- [21] D. Arpit, S. Jastrzebski, N. Ballas, D. Krueger, E. Bengio, M. S. Kanwal, T. Maharaj, A. Fischer, A. Courville, Y. Bengio, and S. Lacoste-Julien, “A closer look at memorization in deep networks,” 2017.
- [22] S. Gojare, R. Joshi, and D. Gaigaware, “Analysis and design of selenium web-driver automation testing framework,” *Procedia Computer Science*, vol. 50, pp. 341–346, 2015, big Data, Cloud and Computing Challenges. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1877050915005396>
- [23] R. Agarwal, D. Schuurmans, and M. Norouzi, “An optimistic perspective on offline reinforcement learning,” 2020.
- [24] J. Schrittwieser, T. Hubert, A. Mandhane, M. Barekatin, I. Antonoglou, and D. Silver, “Online and offline reinforcement learning by planning with a learned model,” 2021.
- [25] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” 2017.

- [26] H. F. Song, A. Abdolmaleki, J. T. Springenberg, A. Clark, H. Soyer, J. W. Rae, S. Noury, A. Ahuja, S. Liu, D. Tirumala, N. Heess, D. Belov, M. Riedmiller, and M. M. Botvinick, “V-mpo: On-policy maximum a posteriori policy optimization for discrete and continuous control,” 2019.
- [27] A. Abdolmaleki, J. T. Springenberg, Y. Tassa, R. Munos, N. Heess, and M. Riedmiller, “Maximum a posteriori policy optimisation,” 2018.
- [28] C.-Y. Lin, “ROUGE: A package for automatic evaluation of summaries,” in *Text Summarization Branches Out*. Barcelona, Spain: Association for Computational Linguistics, Jul. 2004, pp. 74–81. [Online]. Available: <https://www.aclweb.org/anthology/W04-1013>
- [29] N. Schluter, “The limits of automatic summarisation according to ROUGE,” in *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*. Valencia, Spain: Association for Computational Linguistics, Apr. 2017, pp. 41–45. [Online]. Available: <https://aclanthology.org/E17-2007>
- [30] D. M. Ziegler, N. Stiennon, J. Wu, T. B. Brown, A. Radford, D. Amodei, P. Christiano, and G. Irving, “Fine-tuning language models from human preferences,” 2020.
- [31] P. F. Christiano, J. Leike, T. Brown, M. Martic, S. Legg, and D. Amodei, “Deep reinforcement learning from human preferences,” in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/file/d5e2c0adad503c91f91df240d0cd4e49-Paper.pdf
- [32] I. LeCun, “Do large language models need sensory grounding for meaning and understanding?”
- [33] J. Hoffmann, S. Borgeaud, A. Mensch, E. Buchatskaya, T. Cai, E. Rutherford, D. de Las Casas, L. A. Hendricks, J. Welbl, A. Clark, T. Hennigan, E. Noland, K. Millican, G. van den Driessche, B. Damoc, A. Guy, S. Osindero, K. Simonyan, E. Elsen, J. W. Rae, O. Vinyals, and L. Sifre, “Training compute-optimal large language models,” 2022.
- [34] A. Ayesh, “Turing test revisited: A framework for an alternative,” 2019.
- [35] J. Baudrillard, *Simulacres et Simulations*. Galilée, 1981.
- [36] European Parliament and Council of the European Union. Regulation (EU) 2016/679 of the European Parliament and of the Council. [Online]. Available: <https://data.europa.eu/eli/reg/2016/679/oj>
- [37] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” 2017.

8 Appendix

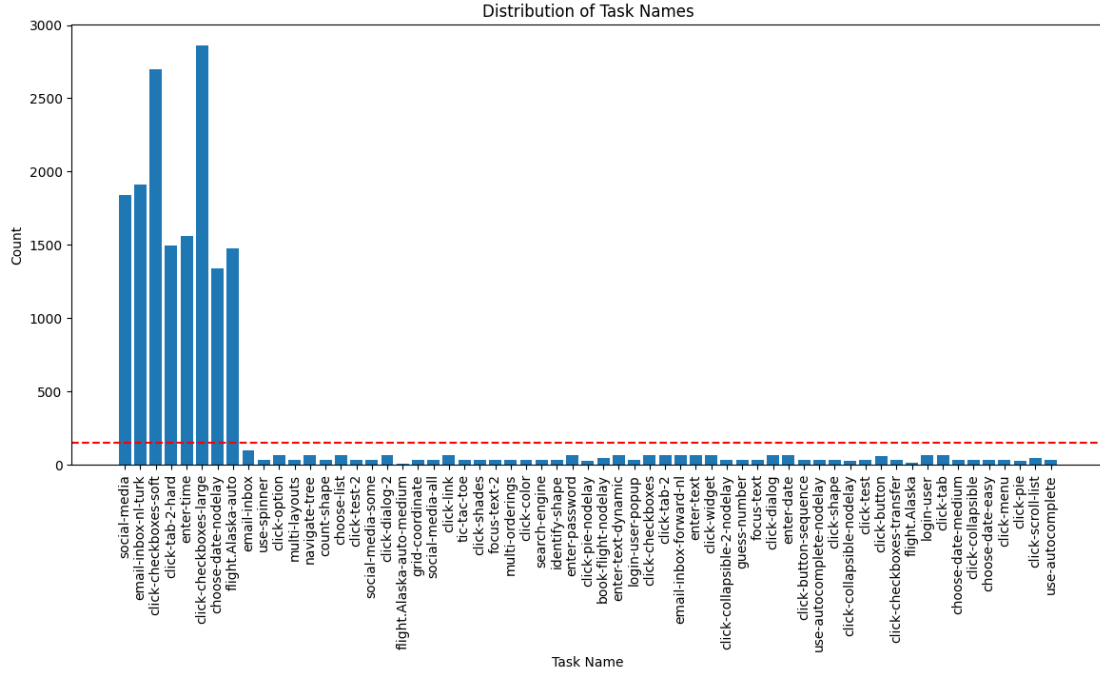


Figure 17: **Distribution of task names from the original Miniwob dataset.** We can observe that some tasks are over-represented, therefore we sample a maximum of 150 episodes for the examples exceeding that number. This leads to an average amount of 56.29 of episodes per task in the dataset.

Table 2: **Average Accuracy of the Different Models, Fine-Tuned T5 and Combined with CC-Net.**

Model Name	Average Accuracy
T5-Base Hierarchical	39.77%
T5-Base Hierarchical No-History	38.02%
T5-Large Hierarchical	43.49%
T5-Large Hierarchical No-History	43.58%
WebNT5-Base	37.94%
WebNT5-Base No-History	35.99%
WebNT5-Base Ordered Refs	35.25%
WebNT5-Base Randomized Refs Test	24.19%
CC-NeT5 Hierarchical (SL)	36.69%
CC-NeT5 Hierarchical (SL+RL)	33.86%

Table 3: **Hyper-parameters used in V-MPO during the RL 'online' phase of our training, inspired by CC-Net training parameters.**

Parameter	Value
Optimizer	Adam [37]
Learning rate	1e-4
Adam b1 parameter	0.9
Adam b2 parameter	0.999
Weight decay (biases excluded)	1e-1
VMPO α	0.1
VMPO η	0.2
Agent discount γ	0.9
Batch size SL	120
Trajectory unroll length	64
Target-network update period T	5
Maximum number of steps per episode	10

Experiment	H-Base	H-Base	H-Large	H-Large	Base	Base	Base	Base
Action History	Yes	No	Yes	No	Yes	Yes	Yes	Yes
Randomized Ref Training	Yes	Yes	Yes	Yes	Yes	Yes	No	No
Randomized Ref Test	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes
book-flight	0	0	0	0	0	0	0	0
chose-date-easy	0	0	0.05	0.06	0	0	0	0
choose-date-medium	0	0	0	0	0	0	0	0
choose-date	0	0	0	0	0	0	0	0
choose-list	0	0	0	0	0.19	0.17	0.32	0.21
click-button-sequence	1	1	1	1	1	1	1	0.08
click-button	1	0.95	1	1	1	1	0.96	0.62
click-checkboxes-large	0.02	0	0	0	0	0	0	0
click-checkboxes-soft	0.08	0.16	0.21	0.13	0	0	0.07	0
click-checkboxes-transfer	0.15	0.36	0.47	0.49	0	0	0.30	0.33
click-checkboxes	0	0.42	0.23	0.32	0.08	0	0.08	0.01
click-collapsible-2	0	0	0.08	0.08	0	0	0	0
click-collapsible	0	0	0	0	0	0	0	0
click-color	0.61	0.58	0.56	0.60	0.53	0.33	0.23	0.21
click-dialog-2	0.71	1	1	1	1	1	0	0
click-dialog	1	1	1	1	1	1	1	1
click-link	0.95	0.95	0.95	0.95	0.51	0.79	0.13	0.01
click-menu-2	0.30	0.29	0.03	0	0.31	0.24	0.53	0
click-menu	0.20	0.17	0.23	0.48	0.25	0.21	0.09	0
click-option	0.59	0.42	0.55	0.36	0.33	0.32	0	0
click-pie	0	0	0	0	0	0	0	0
click-scroll-list	1	0	0	0	0	0	0	0
click-shape	0.50	0.48	0.07	0.15	0.42	0.14	0.20	0.24
click-tab	1	1	1	1	1	1	1	0
click-test-2	1	1	1	1	1	1	1	1
click-test-transfer	1	1	1	1	1	1	1	1
click-test	1	1	1	1	1	1	1	1
click-widget	0.76	0.68	0.87	0.88	0.52	0.51	0.87	0.63
count-shape	0.10	0.07	0	0.02	0.10	0.09	0.08	0.08
enter-text-dynamic	0.12	0.01	0.23	0.25	0	0	0.02	0
enter-text	0.51	0.50	0.61	0.59	0.14		0.45	0.12
enter-time	0	0	0	0	0	0	0	0
focus-text-2	0.33	0.32	1	1	1	1	1	1
focus-text	0	0	1	1	1	1	1	1
identify-shape	0.12	0.12	0.16	0.28	0.13	0.10	0.19	0.17
navigate-tree	0.21	0.16	0.33	0.03	0.13	0.04	0.17	0
use-spinner	0.06	0.05	0.03	0.02	0.02	0.02	0	0

Table 4: Benchmark results over the miniwob environment for the different T5 fine-tuned models over the navigation task in a supervised manner.

Table 5: **CC-NeT + T5 Hierarchical (CC-NeT5)** after first SL 'offline' phase.

Experiment	Full	Ablation Previous Action	Ablation Visual Inputs	Ablation T5
book-flight	0	0	0	0
choose-date-easy	0.02	0	0	0
choose-date-medium	0	0	0	0
choose-date	0	0	0	0
choose-list	0	0	0	0
click-button-sequence	1	0.24	0.36	0
click-button	1	0.50	0.77	0
click-checkboxes-large	0	0	0	0
click-checkboxes-soft	0.20	0.04	0	0
click-checkboxes-transfer	0.41	0.02	0	0
click-checkboxes	0	0	0	0
click-collapsible-2	0	0	0	0
click-color	0.51	0.16	0.24	0
click-dialog-2	0.61	0.37	0.32	0
click-dialog	1	0.46	1	0
click-link	1	0.58	0	0
click-menu-2	0	0	0	0
click-menu	0.19	0.18	0	0
click-option	0.06	0.06	0	0
click-pie	0	0	0.08	0
click-scroll-list	0	0	0.34	0
click-shape	0.30	0.27	0.37	0
click-tab	0.33	0.28	0.17	0
click-tab-2	1	0.87	0.72	0
click-test-transfer	1	1	0.36	0
click-test	1	1	0.39	0
click-widget	0.96	0.62	0.21	0
count-shape	0.32	0.29	0.09	0
enter-text-dynamic	0	0	0	0
enter-text	0	0	0	0
enter-time	0	0	0	0
focus-text-2	1	0.72	0.66	0
focus-text	1	1	1	0
identify-shape	0.21	0.20	0.22	0
navigate-tree	0.20	0.23	0.16	0
use-spinner	0.02	0	0.04	0

Table 6: **CC-NeT + T5 Hierarchical (CC-NeT5) after the final RL phase.**

Experiment	Baseline	RL Visual	Ablation Previous Action	Ablation T5
book-flight	0.0	0	0	0
choose-date-easy	0	0	0	0
choose-date-medium	0	0	0	0
choose-date	0	0	0	0
choose-list	0.04	0.01	0	0
click-button-sequence	0.83	0.77	0.23	0
click-button	0.78	0.50	0.72	0
click-checkboxes-large	0.02	0	0	0
click-checkboxes-soft	0.22	0	0	0
click-checkboxes-transfer	0.34	0	0	0
click-checkboxes	0.12	0.08	0.05	0
click-collapsible-2	0	0	0	0
click-collapsible	0	0	0.06	0
click-color	0.37	0.16	0.31	0
click-dialog-2	0.64	0.32	0.57	0
click-dialog	0.96	0.16	0.27	0
click-link	0.95	0.36	0.34	0
click-menu-2	0	0	0	0
click-menu	0.19	0.05	0.16	0
click-option	0.07	0	0	0
click-pie	0.08	0.12	0.16	0.06
click-scroll-list	0	0	0	0
click-shape	0.31	0.23	0.29	0
click-tab	0.42	0.36	0.35	0
click-tab-2	0.80	0.43	0.62	0
click-test-transfer	1.00	1.00	1.00	0
click-test	1.00	1.00	1.00	0
click-widget	0.84	0.50	0.64	0
count-shape	0.27	0.12	0.09	0
enter-text-dynamic	0	0	0	0
enter-text	0	0	0	0
enter-time	0	0	0	0
focus-text-2	0.96	0.92	0.93	0
focus-text	1.00	1.00	1.00	0
identify-shape	0.18	0.12	0.20	0
navigate-tree	0.21	0.05	0	0
use-spinner	0.03	0	0	0