AutoWebGLM: A Large Language Model-based Web Navigating Agent

Hanyu Lai*†‡
laihy23@mails.tsinghua.edu.cn
Tsinghua University
Beijing, China

Xiao Liu*
shawliu9@gmail.com
Tsinghua University & Zhipu AI
Beijing, China

Iat Long Iong*†
rongyl20@mails.tsinghua.edu.cn
Tsinghua University
Beijing, China

Shuntian Yao†
yaoshuntian@bupt.edu.cn
Beijing U. of Posts and Telecoms
Beijing, China

Yuxuan Chen†
chenyuxu21@mails.tsinghua.edu.cn
Tsinghua University
Beijing, China

Pengbo Shen†
pengbo.shen@outlook.com
U. of Chinese Academy of Sciences
Beijing, China

Hao Yu†
longinyh@gmail.com
Tsinghua University
Beijing, China

Hanchen Zhang†
hc-zhang22@mails.tsinghua.edu.cn
Tsinghua University
Beijing, China

Xiaohan Zhang
xiaohan.zhang@zhipuai.cn
Zhipu AI
Beijing, China

Yuxiao Dong‡
yuxiaod@tsinghua.edu.cn
Tsinghua University
Beijing, China

Jie Tang‡
jietang@tsinghua.edu.cn
Tsinghua University
Beijing, China

ABSTRACT

Large language models (LLMs) have fueled many intelligent web agents, but most existing ones perform far from satisfying in real-world web navigation tasks due to three factors: (1) the complexity of HTML text data (2) versatility of actions on webpages, and (3) task difficulty due to the open-domain nature of the web. In light of these challenges, we develop the open AutoWebGLM based on ChatGLM3-6B. AutoWebGLM can serve as a powerful automated web navigation agent that outperform GPT-4. Inspired by human browsing patterns, we first design an HTML simplification algorithm to represent webpages with vital information preserved succinctly. We then employ a hybrid human-AI method to build web browsing data for curriculum training. Finally, we bootstrap the model by reinforcement learning and rejection sampling to further facilitate webpage comprehension, browser operations, and efficient task decomposition by itself. For comprehensive evaluation, we establish a bilingual benchmark—AutoWebBench—for real-world web navigation tasks. We evaluate AutoWebGLM across diverse web navigation benchmarks, demonstrating its potential to tackle challenging tasks in real environments. Related code, model, and data are released at https://github.com/THUDM/AutoWebGLM.

*HL, XL, and ILI contributed equally to this research.
†Work done while these authors interned at Zhipu AI.
‡Corresponding Authors: YD and JT.

This work is licensed under a Creative Commons Attribution International 4.0 License.

KDD '24, August 25–29, 2024, Barcelona, Spain

© 2024 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-0490-1/24/08
https://doi.org/10.1145/3637528.3671620

CCS CONCEPTS

• **Computing methodologies** → **Intelligent agents**.

KEYWORDS

ChatGLM, Large Language Model, LLM Agent, Web Agent, Reinforcement Learning, Rejection Sampling Finetuning

ACM Reference Format:
Hanyu Lai, Xiao Liu, Iat Long Iong, Shuntian Yao, Yuxuan Chen, Pengbo Shen, Hao Yu, Hanchen Zhang, Xiaohan Zhang, Yuxiao Dong, and Jie Tang. 2024. AutoWebGLM: A Large Language Model-based Web Navigating Agent. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '24), August 25–29, 2024, Barcelona, Spain.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3637528.3671620

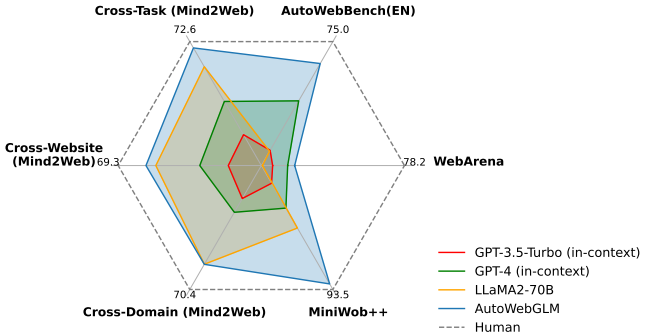
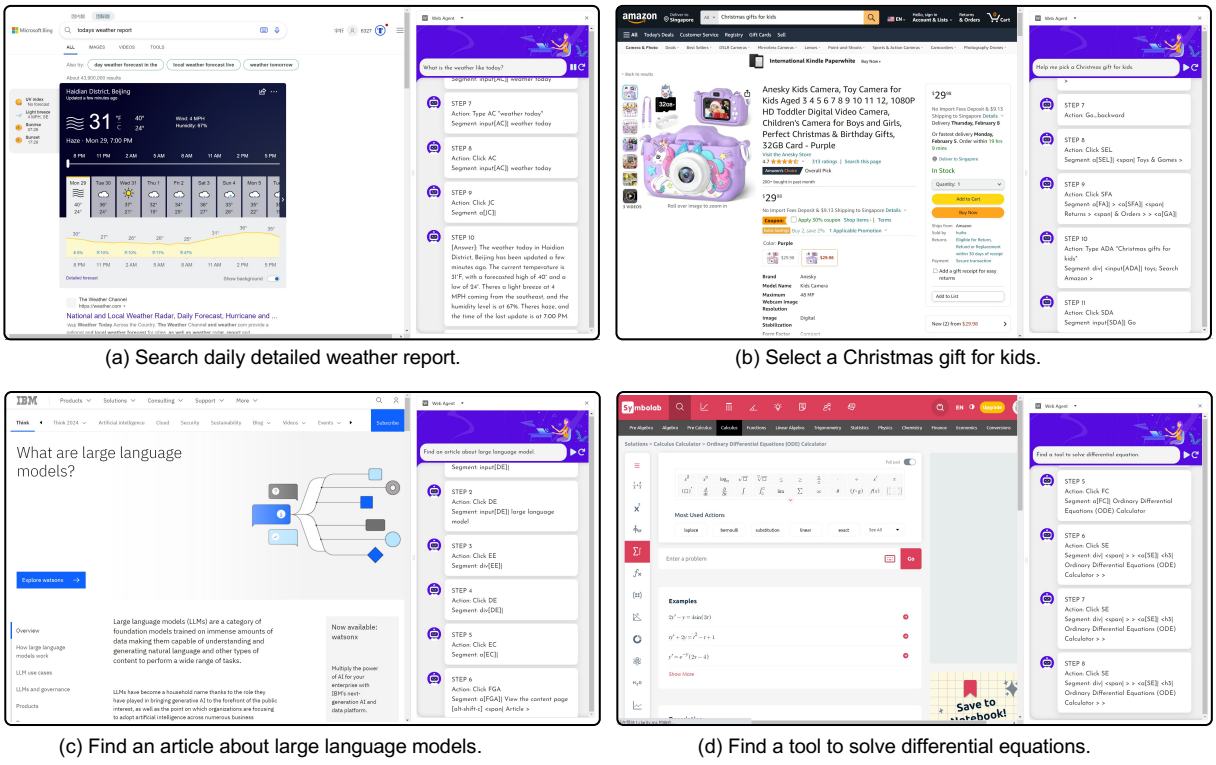


Figure 1: The performance of AutoWebGLM on various web browsing tasks in comparison with GPT-4 and open LLMs.

1 INTRODUCTION

The concept of autonomous digital agents as helpful assistants is an enticing prospect. Enhanced by LLMs' formidable comprehension



(a) Search daily detailed weather report.

(b) Select a Christmas gift for kids.

(c) Find an article about large language models.

(d) Find a tool to solve differential equations.

Figure 2: Examples of AutoWebGLM's execution on four user tasks.

and response capabilities [1, 29–31, 42, 43], we can envision various scenarios previously unimaginable. For instance, an LLM-based agent could support a daily routine that summarizes the online news from the open web for us. This integration of LLMs into everyday tasks heralds a significant shift in how we interact with machines, optimizing our efficiency and redefining the boundaries of machine-assisted productivity [32, 37].

Tremendous efforts have been underway to construct auto web agents. One is AutoGPT, a popular open-source project that utilizes ChatGPT [23] to integrate LLMs with predetermined tools such as web and local file browsing. Meanwhile, the development of agent-centric LLMs has gained significant momentum [10, 26, 33, 40]. Nevertheless, the majority of existing web agents are to date severely restricted in terms of practical applications, predominantly due to the following challenges:

- A universal action space covering all necessary task executions across various websites is absent.
- The diversity and complexity of webpages and their tendentious verbosity pose a significant challenge for LLMs to comprehend the content and carry out correct operations accurately.
- Existing agents notably lack the capability for correct inference and self-checking on web tasks. Once caught in an erroneous loop, they struggle to rectify the issue promptly.

In this work, we introduce AutoWebGLM for building webpage navigation agents. It is built upon the open ChatGLM3-6B model [42]. First, we propose various efficient data strategies to support the swift construction of a sizeable, reliable training dataset while state-of-the-art models cannot reliably complete data annotation tasks [45]. Furthermore, by leveraging supervised [24] and reinforcement learning methods [27], we train AutoWebGLM on top of the collected web agent dataset to achieve performance superiority on general webpage browsing tasks. A step further, we employ rejection sampling finetuning (RFT) [31] for lifelong learning in specific web environments, enabling the agent to excel in a particular domain.

We develop and deploy a Chrome extension based on AutoWebGLM (See Figure 2 for examples). Throughout our experiments, it can reason and perform operations on various websites to complete user tasks accurately, making it practically applicable to real-world services. In addition, we construct the first bilingual (English and Chinese) webpage browsing evaluation dataset to build AutoWebBench, given that websites from different regions have substantial stylistic variations.

In conclusion, we make the following contributions in this paper:

- We design and develop the AutoWebGLM agent for effectively completing web browsing tasks through curriculum learning, bootstrapped by self-sampling reinforcement learning, and RFT in the web browsing environment.
- We construct a real webpage browsing operation dataset of approximately 10,000 traces using model-assisted and manual methods, including the bilingual (English and Chinese) web browsing benchmark AutoWebBench.
- We perform experiments to demonstrate that AutoWebGLM with six billion parameters achieves performance comparable to

the most advanced LLM-based agents, and more importantly, it reaches a practically usable level for real-world web tasks.

2 RELATED WORK

Constructing a comprehensive web browsing agent is a complex task that involves various modules, such as a language model for decision-making and an HTML parser for environment observation. Furthermore, it is essential to have appropriate web browsing evaluation criteria when creating an effective web browsing agent. In this section, we will discuss the works related to these aspects.

Language Models (LMs). Large language models (LLMs) [44], such as GPT-4 [1], Claude-2 [2], LLaMA-2 [30], ChatGLM [8, 42], OPT [43], and BLOOM [29], have accumulated extensive knowledge in various natural language processing tasks. However, due to the high cost of deploying such large language models, smaller models with lower costs and comparable capabilities are usually preferred. Many open-source projects, such as LLaMA-2-7B [30] and ChatGLM3-6B [42], have demonstrated strong performance to large language models in some domains.

Benchmarks for Web Navigation. The primary web browsing evaluation datasets provide a variety of evaluation metrics. MiniWoB++ [12] provides several simulated web environments, with tasks primarily to evaluate the model's ability to interact with webpage components. However, with the increasing demand for complex web operation capabilities, Mind2Web [7] and WebArena [45] have been created. Mind2Web is an offline evaluation set for complex web browsing that provides several metrics. The evaluation method is straightforward and commonly used for model evaluations. In contrast, the WebArena benchmark, based on real websites, creates multiple virtual environments and uses various evaluation methods to assess the task completion rate, making it more suitable for real-world task completion evaluation.

Agents for Web Automation. Previous work such as WebGPT [21] and WebGLM [16] combined LLMs with web environments. However, their primary application was question-answering (QA) tasks [4, 15, 22, 28], utilizing internet resources to answer user queries. Recent works [6, 10, 19, 39] focus more on executing complex operations or interactive tasks. A fundamental aspect of web browsing tasks involves a comprehensive understanding of HTML. StructGPT [13] explores methodologies to enhance the zero-shot reasoning ability of LLMs in handling structured data. Specifically, MindAct [7] works by filtering webpage elements and selecting the element through multiple rounds of multiple-choice questions. It often requires more than ten model calls to complete a single web operation, which could be more efficient. On the other hand, WebAgent [9] uses HTML-T5 to process the observation space's content, including HTML, previous operations, and user instructions. It uses the Flan-U-Plam model to generate code to control webpages, exhibiting excellent web browsing performance. However, it faces deployment challenges due to the size of the Flan-U-Plam model, which is 540B scale. AutoWebGLM, based solely on a single ChatGLM3-6B, has a robust web browsing capability comparable to WebAgent, demonstrating high value for practical deployment.

Prompt-based Data Construction Methods. Constructing data through prompts has recently gained significant traction [5, 11, 20, 25, 35]. This approach leverages language models to generate synthetic data for training. A notable example is Evol-Instruct [17, 38], inspired by the theory of evolution, demonstrating the effectiveness of using LLMs to generate diverse and complex instructions for various tasks. Additionally, some researchers explore the potential of generating data in a zero-shot setting, where the model produces data for tasks it has yet to be explicitly trained on [18], highlighting the versatility of prompt-based data construction. These methodologies rapidly evolve, offering a promising avenue for data generation in various domains, especially where traditional data collection methods could be more practical and sufficient..

Rejection Sampling Finetuning. The methodology of Rejection Sampling Finetuning (RFT) [41] employs a supervised learning model to generate and collect accurate reasoning paths, subsequently utilized as an augmented finetuning dataset. Using RFT to expand the dataset with diverse reasoning paths can boost the mathematical performance of LLMs. Our experiments show that RFT can also be effectively implemented in web page browsing tasks, significantly increasing professional capabilities within specific environments.

3 AUTOWEBGLM AS A WEB AGENT

3.1 Problem Setup

We consider web browsing tasks as a sequential decision-making process. The state, denoted as S, includes the current page status, such as HTML, URL, and Window Position. The action set A contains all potential browsing operations, including click, type, scroll, etc (See complete operations in Table 1).

$$S = \{\text{HTML, URL, Window Position}\}, A = \{\text{click, scroll, type, } \ldots\}$$

The state's transition is determined by the webpage's current state and the agent's output action. During the decision-making process, the function ϕ updates the historical information based on the previous history H_{t-1}, the most recent action A_{t-1}, and the current state S_t.

$$H_t = \phi(H_{t-1}, A_{t-1}, S_t)$$

The policy π is the process for the agent to choose actions based on the current state and the history. A complete decision process starts from the initial state S_0 and history H_0, iterating through the policy π and transition function T. This iteration ceases when the action A_t is *finish* or reaches the maximum length.

$$(S_{t+1}, H_{t+1}) = (T(S_t, A_t), \phi(H_t, A_t, S_{t+1}))$$
$$A_t = \pi(S_t \mid H_t)$$
$$S_{t+1} = T(S_t, A_t)$$

3.2 The AutoWebGLM Framework

As depicted in Figure 3, we process information through HTML simplification and OCR (Optical Character Recognition) modules, yielding a simplified HTML representation after acquiring HTML and webpage screenshots. With attributes facilitating operability judgment, we mark operable elements for agent interaction. The OCR module is for notating text elements during image parsing.

Agents initiate action prediction by combining this representation with other observational data. Upon outputting action, the automated web program is employed for action execution; this

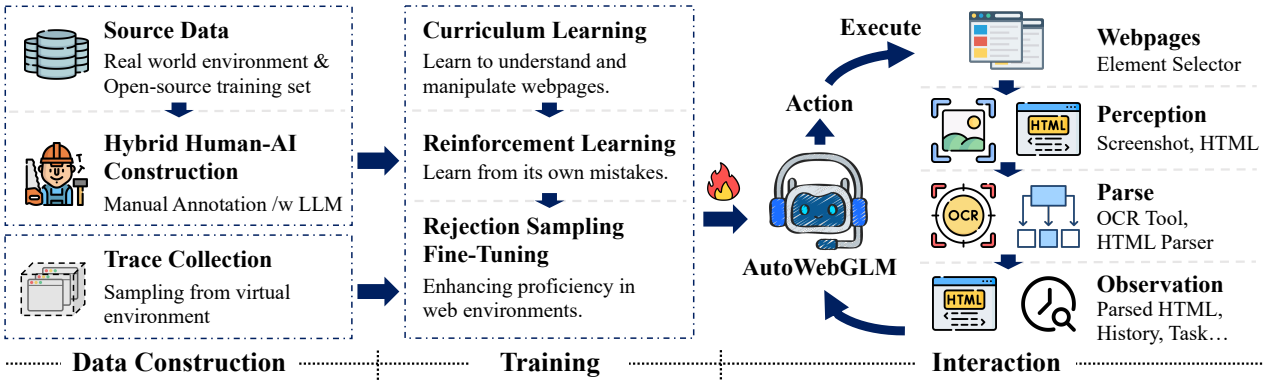


Figure 3: The System Architecture of AutoWebGLM. Our system comprises two key components: interaction framework and LM agent. The LM agent learns from data procured from diverse sources. It further employs RL and RFT to bootstrap itself, thus enhancing web browsing capabilities. The interaction framework uses various web processing modules to organize concise HTML and other information for the LM agent to make decisions that are then executed by an automated browsing program.

Table 1: Action space for AutoWebGLM to interact through.

Instruction	Description
`click(id)`	Click at an element
`hover(id)`	Hover on an element
`select(id, option)`	Select option in an element
`type_string(id, text, enter)`	Type to an element
`scroll_page(direction)`	Scroll up or down of the page
`go(direction)`	Go forward or backward of the page
`jump_to(url, newtab)`	Jump to URL
`switch_tab(id)`	Switch to i-th tab
`user_input(message)`	Notify user to interact
`finish(answer)`	Stop with answer

iterative cycle persists until task termination. AutoWebGLM enhances interactive capacity and webpage navigation precision by amalgamating these components into a singular framework.

A comprehensive, precise observation and action space is vital for constructing a robust web browsing framework. These spaces standardize the conversion of varied data sources into a uniform format. We discuss our designs in the following:

3.2.1 Observation space. We suggest using a unified observation space to enhance the model's webpage comprehension and operation level. The observation space should provide information as close as possible to what the browser's graphical interface can provide, thus maximizing the upper bound of the agent's capabilities. We identify four critical indicators for web browsing tasks: task description, simplified HTML, current location, and past operation records. The HTML provides the model with structural and content information about the page, while the current location information helps the model understand its position within the webpage. The record of past operations provides the model with historical context, which helps to generate more consistent subsequent operations. By incorporating these elements into the observation space, we strive to construct a more resilient and practical model that can handle the intricacy and variability inherent in web browsing tasks. The following are detailed illustrations of the observation space components.

Algorithm 1: HTML Pruner

Data: tree *tree*, kept elements *kept*, recursion count *rcc*, max depth *d*, max children *mc*, max sibling *ms*
Result: pruned tree *tree*

```
nodes ← []
for t ← 0 to rcc do
    for id in kept do
        node ← tree.element with id
        nodes.append(node)
        nodes.append(getAnscendants(node, d))
        nodes.append(getDescendants(node, d, mc))
        nodes.append(getSiblings(node, ms))
    end for
    d, mc, ms ← update(d,mc,ms)// make them smaller
end for
for node in reversed(tree) do
    if not node in nodes or not (node has text or attrib or
     len(node.children) > 1 or node is root) then
        tree.remove(element)
    end if
end for
```

HTML. The HTML webpages are vast and complex, so it is necessary to simplify them before inputting them into the model. The simplification process aims to extract essential information while eliminating redundant or disruptive elements that could hinder the model's understanding. Throughout this process, the HTML's basic structure and significant content information must be retained to enable the model to comprehend and utilize this information for effective web browsing. HTML Pruner can efficiently convert a tree of elements into a concise representation. We can use the processing techniques to streamline the original HTML format into a more understandable structure for the model to interpret and manage, improving model effectiveness in web browsing tasks.

Current Position. Based on our observation of the model's interaction with the environment, agents could perform better when provided with window position and page size. The agent uses the page scroll position to understand the content of the currently visible area and the page height information to comprehend the scale of the entire page, providing a spatial context for the model.

Previous actions. The best solution to inform the agent of past operations is explicitly providing it. This approach helps the agent understand its past behaviors. It prevents the agent from getting stuck in an ineffective loop of repeating the same actions due to operational failures, improving its ability to adapt to the complexities and dynamics of web browsing tasks by preventing the recurrence of unsuccessful operations.

3.2.2 Action space. As the approach of this work is to build a language model-based web browsing agent, we focus on operational possibilities when constructing the action space. On an extensive summary of experiences in the real task execution process, we define a complete and self-consistent action space (in Table 1) for the language model to act in the web browsing world. We design our prompt input in Section B.

4 BUILDING AUTOWEBGLM

In this section, we detail the construction of a web browsing agent. Given the high costs associated with manual data construction and the inadequacy of current LLMs for automated data generation, we employed a Human-AI hybrid data construction method to efficiently produce large volumes of training data at a reduced cost. Additionally, we implemented a multi-stage training approach, rather than relying solely on imitation learning, to enhance our model's general and specialized web browsing capabilities.

4.1 Data Construction

Considering the scarcity of high-quality, complex web browsing data produced by actual users, we aim to create a training dataset. However, the dataset construction presents several challenges:

- Task Collection: A significant hurdle is acquiring diverse, real-user task queries across various websites.
- Privacy and Security: Privacy and security limitations hinder the direct acquisition of user browser operation sequences. It is also challenging to rule out redundant or incorrect operations not pertinent to task completion and to confirm user task completion.
- Objective Annotation: The labor-intensive nature of collecting user objectives for each operational step makes it impractical in real-world data-gathering scenarios.
- Model Limitations: Current models cannot process complex user queries across different websites, thus eliminating the chance of using purely automated methods for accurate browsing trajectory collection in real and complex application contexts.

As illustrated in Figure 4, we suggest a hybrid human-AI Data Construction method to create our training data in response to these challenges. After careful consideration, we categorize our data into two types for construction:

4.1.1 Web Recognition & Simple Task Operation Construction. For web browsing tasks, efficient and accurate understanding and manipulation of webpages become vital challenges in model development due to the diversity of user behaviors and the complexity of web content. This section illustrates our construction method for web recognition and simple task operation to train models to recognize webpage structures and perform basic operations accurately.

Web Recognition. The main objective of Web Recognition includes understanding particular HTML formats, identifying different types of web elements (such as text boxes, buttons, images, etc.), and understanding the role of these elements in user interaction. We propose the following construction approach based on the above practical challenges.

We initiate our process by collecting URLs from Chinese and English mainstream websites listed on Similarweb[1]. In the data processing stage, we use our HTML parser to identify operable components in each webpage and record essential information such as component position and size. We then generate a simplified HTML by rearranging and simplifying the component tree (see details in Section 3.2).

We design tasks such as website and component function descriptions to aid model recognition of webpage structures and interactive components' functions. For each task, we develop a series of natural language questions to serve as the source field in our data. GPT-3.5-Turbo is utilized to generate multiple formulations for each question, thereby diversifying the question formation.

For the target, we leverage GPT-3.5-Turbo to generate the response. We supply a simplified HTML with the pertinent question in the prompt and impose a limit on the response length, thereby obtaining our target.

Simple Task Operation. The main objective of the Simple Task Operation dataset is to train models to perform single-step web operations. This involves executing basic functionalities on web pages, such as clicking links, filling out forms, or navigating to specific sections. To build our data, we collect various websites in the same way as Web Recognition. Then, we construct a split for each operation type to ensure that our dataset covers all the requirements for simple task operations. We adjust the data size for each split based on the frequency of each operation in practice.

Our key to constructing the dataset is by rules instead of model generation. We try GPT-3.5-Turbo for tasks, intent, and operation generation and Selenium [2] to validate the executability of the generated results. However, it has obvious drawbacks: The model cannot reach an acceptable accuracy in the operation to fulfill the task, and the correctness of the model-generated operations is hard to judge. To address the above issues, we endeavor to approach from a novel perspective. We identify various actionable elements within the webpage, assembling them into web operations. Then, we use GPT-3.5-Turbo to produce the corresponding tasks and operational intents for these actions. For operation types with relatively fixed behaviors, such as Scroll and Jump_to, we directly generate their corresponding tasks with templates; for flexible and feature-rich operations, such as Click and Type, we use GPT-3.5-Turbo to help

[1] https://www.similarweb.com/top-websites
[2] https://www.selenium.dev

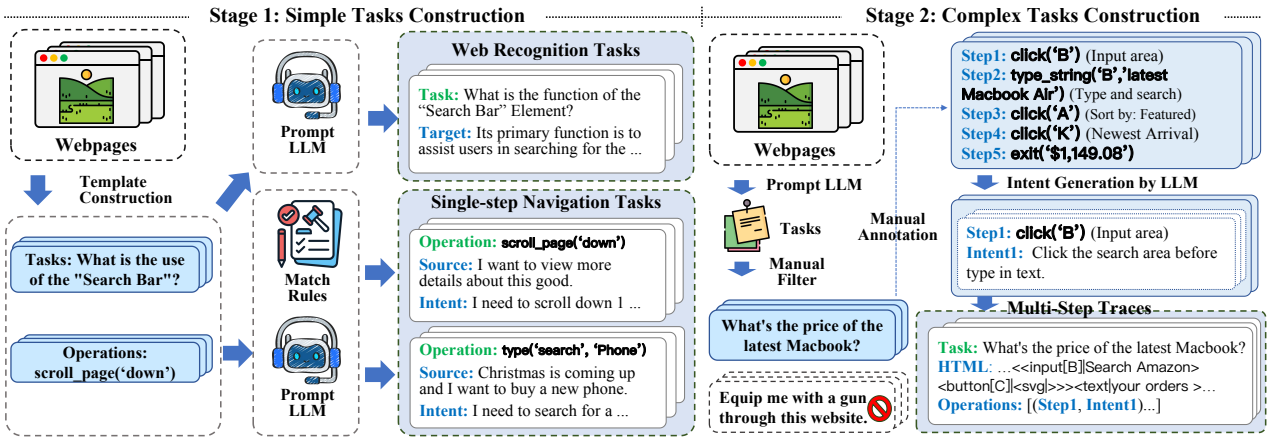


Figure 4: Data Construction. Data construction is divided into two main stages; the first stage is webpage recognition tasks and simple tasks operation construction, and the second stage is complex tasks construction.

complete the construction. This approach ensures the instructions' executability and provides the operation tasks' richness.

4.1.2 Complex Task Operation Construction. We developed a dataset for complex web tasks to enable the model to make plans and reason in the web browsing scenario. Each sample in the dataset consists of a real-world complex web browsing task, the sequence of operations to complete the task, and the intent of each step.

We first designed 50 complex tasks for each website using the prompting technique referring to Evol-Instruct [38], from which about 20 feasible tasks were manually selected and labeled. For operation sequence, due to the high complexity of the tasks, even the most advanced LLMs cannot complete the task with satisfactory accuracy. Therefore, we leveraged manual annotations to capture web task executions via a browser plugin that records actions during website tasks. Chain-of-thought [36] reasoning has been proven to improve task comprehension and model performance [14, 35] significantly. However, leveraging human annotators to document their intent and reasoning during web browsing is inefficient. To improve the CoT construction process, we used GPT-4 as the operational intent predictor. Our first approach of iterative step-by-step creation proved to generate weak operational links and incurred high API costs due to data construction. To address this, we employed a global thought chain prompting method, where all operations and critical HTML segments are inputted into a trace. Then, we prompted GPT-4 to output intentions for each step. This method improves the accuracy and cohesion of each step, thus forming highly relevant, consistent thought chains.

After construction, we merge our data with the training set from Mind2Web and MiniWob++ to form our final training dataset. The proportion of each split is in Figure 5.

4.2 Training

We train the model through three steps illustrated in Figure 6.

4.2.1 Step 1: Curriculum Learning. The first one is Supervised Fine-Tuning (SFT). We utilize data in Section 4.1 for training

$$\mathcal{L}_{\mathcal{SFT}}(\pi_\theta) = -\mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\log \pi_\theta(y \mid x)\right] \quad (1)$$

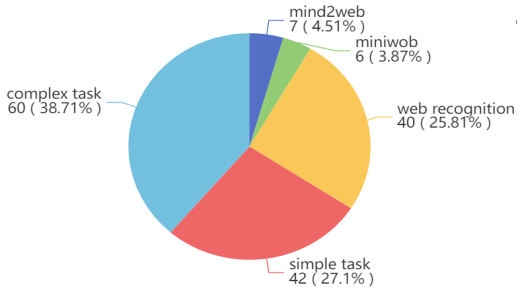


Figure 5: Dataset Proportion. Piechart of the distribution of splits within our training data.

This approach enhances the model's comprehension of webpages and its capability as an agent to perform operations within the environments. Significantly, we use curriculum learning (CL), which mimics the human learning process, advocating for models to start learning from easy samples and gradually advance to complex ones. It has been demonstrated in prior works[3, 34] to improve model capabilities substantially.

Enabling LM to Read and Operate on the Web. In the initial stage, we mix the data constructed in Section 4.1.1 to equip the model with the ability to (1) comprehend the structure of web pages and the functions of various web components, and to (2) execute predefined operations on the current webpage, thus implementing simple user instructions.

To Make LM Learn to Plan & Reason on the Web. During this stage, we continue to employ the constructed data in Section 4.1.2 for training. We enable our model to decompose tasks into subtasks and execute subsequent steps based on the current webpage and the sequence of prior operations.

After the above training, our model M_{SFT} acquired essential capability in completing web browsing tasks and could independently execute operations based on user instructions.

4.2.2 Step 2: Reinforcement Learning. Following previous training, M_{SFT} has demonstrated some ability to operate the browser and infer the task. However, due to the distinctive nature of SFT training,

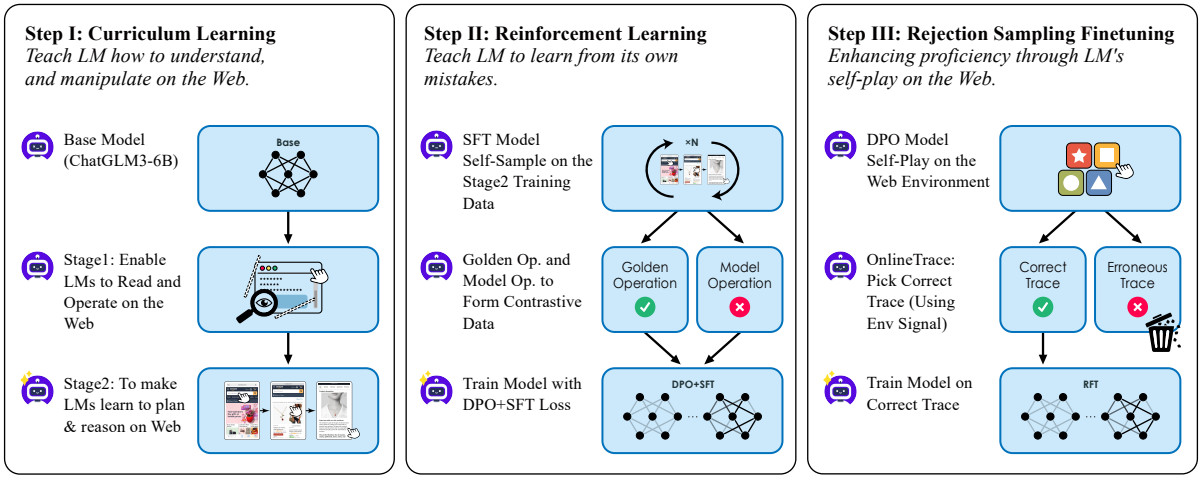


Figure 6: The Training Procedure. First, the model learns webpage interpretation and operation via curriculum learning. Next, it self-samples training data, learning from its mistakes. Finally, it self-plays in the environment, becoming a domain expert.

M_{SFT} attempts to mimic the inference process and operations but sometimes overlooks the webpage's state and preceding operation sequences, leading to hallucination. Consequently, we propose a self-sampling reinforcement learning to mitigate these operative illusions.

First, we use M_{SFT} for n-fold sampling (n=20) on complex task operation samples in the training set. We combine the sampled output and golden answer to construct contrastive data with positive and negative pairs. Subsequently, we retain samples based on the following criteria:

- From all n iterations of sampling, we select data where the model completed the tasks from 1 to n-1 times. If M_{SFT} answered all iterations correctly, we consider it devoid of training value and incapable of providing practical negative examples. Conversely, If M_{SFT} answered incorrectly across all iterations, we suspect issues with the data and exclude them, as the model cannot adequately fit these outliers during optimization.
- We retain different erroneous operations and remove duplicates to preserve distinct negative examples.

After constructing contrastive data $D_{\text{Const.}}$, we employ the DPO[27] training approach to make M_{SFT} learn from its mistakes and further enhance its capabilities. During the training, we found that the direct use of DPO loss led to instability. To mitigate this issue, we propose including SFT loss to stabilize the reinforcement learning process and increase the number of training steps while ensuring no loss of the original model's natural language and agent abilities, achieving a more robust model M_{DPO}:

$$\mathcal{L}_{DPO}(\pi_\theta;\pi_{\text{ref}}) = -\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}}\left[\log\sigma\left(\beta\log\frac{\pi_\theta(y_w|x)}{\pi_{\text{ref}}(y_w|x)} - \beta\log\frac{\pi_\theta(y_l|x)}{\pi_{\text{ref}}(y_l|x)}\right)\right] \tag{2}$$

$$\mathcal{L}_{SFT}(\pi_\theta;\pi_{\text{ref}}) = -\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}}\left[\log\pi_\theta(y_w \mid x)\right] \tag{3}$$

$$\mathcal{L}_{Total} = \lambda\cdot\mathcal{L}_{DPO} + \mathcal{L}_{SFT} \tag{4}$$

4.2.3 Step 3: Rejection Sampling Finetuning. In the RFT (Rejection Sampling Finetuning) step, we aim to optimize for webpage environments in specific domains. RFT enables us to perform targeted training through substantial sampling from an existing model, selecting the accurate trajectories in instances lacking ones via reward signals. Our reward signals can be furnished either by the environment itself or through pre-designed reward models. Due to the network policy constraints inherent in real webpage environments, we conduct our experiments within sandbox environments furnished by MiniWob++ and WebArena.

For MiniWob++, we leverage the query generator in MiniWob++ to auto-generate multiple user queries for each task. We determine the number of generated queries for each task based on its difficulty. Then, we employ M_{DPO} to try to solve the queries. If a trace completes the task (as adjudged by the MiniWob++ environment), we consider this trace as a positive trace.

In the case of WebArena, to prevent overlap with the test set, we manually construct multiple distinctive user queries based on WebArena's templates. For each sample, we apply M_{DPO} to perform 64 times of sampling. Similarly, if our model completes the task at least once (adjudged by manually written rules), we deem the successful trace as a positive trace.

By utilizing the methods above, we constructed two distinct successful datasets, one from MiniWob++ and the other from WebArena. These comprise approximately 15k traces (66k steps) and 240 traces (2k steps), respectively, which are used for AutoWebGLM's individual finetuning on these two tasks.

4.3 Benchmark: AutoWebBench

We segment the complex task operation dataset collected in Section 4.1.2 for evaluation. AutoWebBench is divided into two splits: in- and out-of-domain, which serve as bases for our performance assessment. The in-domain dataset represents training data collected from the same website, measuring the model's performance

Table 2: The performance on AutoWebBench.

Model	Size	English		Chinese	
		Cross-Task	Cross-Domain	Cross-Task	Cross-Domain
GPT-3.5-Turbo	N/A	12.1	6.4	13.5	10.8
GPT-4	N/A	38.6	39.7	36.7	36.3
Claude2	N/A	13.2	8.1	13.0	7.9
LLaMA2	7B	3.3	2.5	-	-
LLaMA2	70B	8.3	8.9	-	-
Qwen	7B	9.0	7.6	9.1	7.5
AutoWebGLM	6B	**64.8**	**58.6**	**65.4**	**61.8**

under familiar conditions. In contrast, the out-of-domain dataset encompasses data collected from websites entirely excluded from our training set. It offers a unique opportunity to measure the model's generalizability and ability to adapt to unfamiliar environments. We select 50 browsing traces for each split as our test data. These traces are scrutinized and filtered via human verification, ensuring a more reliable evaluation benchmark.

Drawing on the methodology presented in Mind2Web, we comprehensively evaluate each step involved in the operation. This allows us to assess the step and overall accuracy of the model's operations. Detailed results of this evaluation are available in Table 2.

5 EXPERIMENTS

We establish a bilingual (Chinese-English) benchmark AutoWebBench and evaluate the abilities of publicly available agents. We also conduct extensive experiments on numerous benchmarks to evaluate the performance of AutoWebGLM in comparison to several baselines across various tasks involving navigating both English and Chinese websites.

5.1 Main Results

AutoWebBench. As discussed in Section 4.3, We divide the test set into four splits: Chinese, English, in-domain, and out-of-domain, for evaluation purposes. We use the Step Success Rate (SSR) as our evaluation metric. The results are in Table 2.

Mind2Web. We use the settings from Mind2Web with SSR as our primary evaluation metric. To compare the model fairly, we utilize the MindAct framework provided by Mind2Web to evaluate the model's performance. The results are in Table 3.

MiniWoB++ & WebArena. For MiniWob++, following the experimental setup from WebAgent [9], we test MiniWoB++ with 56 tasks by running 100 evaluation episodes per task to evaluate model capabilities. For WebArena, we integrate our HTML parser module and action execution module into the WebArena environment to make it compatible with our system. The results are in Table 4.

5.2 System Execution Efficiency

Furthermore, since execution speed is critical to user experience, we conduct a series of performance experiments to evaluate the execution efficiency of each system component and identify areas that could be further optimized. The results of these experiments are presented in Table 5.

Table 3: The performance on Mind2Web. † indicates that only top-10 candidates were used for this test, otherwise top-50 was used. * indicates model's finetuning on train set.

Model	Size	Cross-Task	Cross-Website	Cross-Domain	Average
GPT-3.5-Turbo	N/A	17.4	16.2	18.6	17.4
GPT-4†	N/A	36.2	30.1	26.4	30.9
Flan-T5-XL*	3B	52.0	38.9	39.6	43.5
Html-T5-XL*	543B	**71.5**	**62.2**	**67.1**	**66.9**
LLaMA2*	7B	52.7	47.1	50.3	50.1
LLaMA2*	70B	55.8	51.6	55.7	54.4
Qwen-VL*	9.6B	12.6	10.1	8.0	10.2
SeeClick*	9.6B	23.7	18.8	20.2	20.9
AutoWebGLM	6B	66.4	56.4	55.8	59.5

Table 4: The performance on MiniWoB++ and WebArena. * indicates model's finetuning on train set.

Model	Size	MiniWoB++	WebArena
GPT-3.5-Turbo	N/A	13.4	6.2
GPT-4	N/A	32.1	14.4
Text-Bison-001	N/A	-	5.1
LLaMA2	7B	42.8*	1.2
LLaMA2	70B	47.1*	0.6
Html-T5-XL	543B	85.6*	-
WebN-T5-XL	3B	48.4*	-
Lemur	70B	-	5.3
AutoWebGLM	6B	**89.3**	**18.2**

Table 5: System Execution Efficiency

Action	count/tr	Fetch	Parse	Predict	Execute	Loading
`type_string`	1.00	362.00	28.88	2282.99	6.12	2082.61
`click`	3.62	438.62	72.07	2252.10	28.10	2094.60
`finish`	0.38	419.33	66.00	3054.16	6.00	2119.06
`scroll_page`	0.38	475.33	88.67	3396.37	5.33	2033.35
Others	0.25	680.50	152.50	2666.08	10.00	2142.92
Average	-	436.93	68.68	2407.34	20.36	2092.13
Percentage(%)	-	8.70	1.37	47.89	0.41	41.64

5.3 Ablation Study

To evaluate the impact of different stages of data and training strategies on model performance enhancement, we conduct a comprehensive ablation study in Table 6.

Training Data Ablation. We train and test only models that contain the original training set and incorporate simple and complex task data (see Section 4.1) for training. This approach helps to qualitatively measure the impact of different datasets on the model.

The Complex Task dataset significantly improves model performance. We hypothesize that this is due to the complex data more closely aligning with real-world scenarios, thereby fundamentally transforming model performance.

The simple task dataset shows only a slight improvement when training alone. However, when training jointly with the complex

Table 6: Ablation study. AutoWebBench and WebArena do not have a training set, while the RFT stage is only suitable for sampling in the environment, so we represent them by "-".

Method	AutoWebBench	Mind2Web	MiniWob++	WebArena
Training Data Ablation				
Only Train Set	-	48.1	44.3	-
+) Stage1	23.5	48.4	48.3	2.5
+) Stage2	60.2	55.2	78.9	7.6
+) Stage1+2	61.8	56.7	81.7	8.3
Training Strategy Ablation				
SFT	61.8	56.7	81.7	8.3
+) DPO	62.7	59.5	80.8	8.5
+) RFT	-	-	89.3	18.2
AutoWebGLM	62.7	59.5	89.3	18.2

Table 7: Error Distribution in Web Task Automation

Error Type	Proportion
Hallucinations	44%
Poor Graphical Recognition	28%
Misinterpretation of Task Context	20%
Pop-Up Interruption	8%

task dataset, there is a significant improvement. We find that training exclusively with complex task datasets leads to basic operational errors, suggesting that training with simple task datasets can effectively mitigate this problem.

Training Strategy Ablation. We compare the results of SFT, DPO, and RFT-enhanced models and find that: (1) Compared to SFT, the DPO training facilitates model learning from its mistakes, further enhancing model performance. (2) RFT enables our model to perform bootstrap enhancement in different domains. With practice comes proficiency, resulting in improvements within each domain.

5.4 Case Study and Error Analysis

To assess the effectiveness of our model, we conduct a series of case studies covering a range of web-based tasks, including everyday use, leisure and relaxation, and academic research, covering the typical range of web requirements. Our system achieves satisfactory results in most scenarios.

While our system performs commendably well on a variety of web-based tasks, it has limitations. We identify errors that occasionally occur during task execution, which can be broadly categorized into four types: hallucinations, poor graphical recognition, misinterpretation of task context, and pop-up interruptions. Table 7 outlines the proportion of these errors observed during error analysis. Although relatively infrequent, these errors are crucial in our ongoing efforts to refine and enhance the system's capabilities.

6 FUTURE DIRECTION

6.1 Multimodal Input

While HTML input has produced satisfactory results in many scenarios, our system falters when confronted with advanced web applications such as maps, animations, and video browsing. In our analysis, the strength of image input lies in its indispensable role in interpreting images, icons, and special effects. However, compared to text input, image input presents additional challenges in understanding numerals and extensive web text. Consequently, we consider that a multimodal system, integrating HTML and webpage screenshots, combines the advantages of both modalities, substantially enhancing the model's capability in web browsing tasks.

6.2 Reasoning and Self-check Techniques

The system's efficiency and success rate in web browsing may decrease when dealing with unfamiliar websites or those with unique operating logic. To mitigate this issue, an exciting avenue for exploration is the development of novel reasoning strategies distinct from the Chain-of-Thought approach, enabling the model to make better-informed decisions based on previous browsing experiences, thereby improving the success rate and efficiency of web browsing. Moreover, due to unstable internet connections and other factors, the stability of a real web environment is not guaranteed. Thus, self-check mechanisms within the web browsing agent system, including confirming the current state and verifying the intended operation's effect, could significantly improve the system's robustness and effectiveness.

6.3 Mobile Application

The mobile platform is another promising application scenario with massive potential. Compared to the web platform, it presents its challenges and opportunities. For example, due to their screen size, mobile devices display fewer elements within the viewport, simplifying the page XML. Furthermore, the operation logic on mobile platforms is generally more straightforward than on web platforms. However, mobile operation space includes more complex actions such as gestures, and mobile platforms face more system security restrictions, imposing additional constraints on software development.

7 CONCLUSION

In this work, we present AutoWebGLM, an advanced language model-based agent exhibiting robust performance in various autonomous web navigation benchmarks. Our model addresses extant LLM limitations and simplifies webpages by effectively controlling HTML text length and handling the web's open-domain nature. We strategically employ curriculum learning, reinforcement learning, and rejection sampling finetuning to enhance webpage comprehension and browser operation learning. We also introduce a unique bilingual web browsing benchmark— that lays a solid foundation for future research. Our findings represent significant progress in utilizing LLMs for intelligent agent tasks.

ACKNOWLEDGMENTS

This work is supported by Natural Science Foundation of China (NSFC) 62276148 and 62425601, the New Cornerstone Science Foundation through the XPLORER PRIZE and Tsinghua University (Department of Computer Science and Technology) -Siemens Ltd., China Joint Research Center for Industrial Intelligence and Internet of Things (JCIIOT).

REFERENCES

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774* (2023).

[2] Anthropic. 2023. Model Card and Evaluations for Claude Models. (2023).

[3] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. 2009. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*. 41–48.

[4] Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on freebase from question-answer pairs. In *Proceedings of the 2013 conference on empirical methods in natural language processing*. 1533–1544.

[5] Jiale Cheng, Xiao Liu, Kehan Zheng, Pei Ke, Hongning Wang, Yuxiao Dong, Jie Tang, and Minlie Huang. 2023. Black-box prompt optimization: Aligning large language models without model training. *arXiv preprint arXiv:2311.04155* (2023).

[6] Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Yantao Li, Jianbing Zhang, and Zhiyong Wu. 2024. SeeClick: Harnessing GUI Grounding for Advanced Visual GUI Agents. *arXiv preprint arXiv:2401.10935* (2024).

[7] Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. 2023. Mind2Web: Towards a Generalist Agent for the Web. *arXiv preprint arXiv:2306.06070* (2023).

[8] Zhengxiao Du, Yujie Qian, Xiao Liu, Ming Ding, Jiezhong Qiu, Zhilin Yang, and Jie Tang. 2022. GLM: General Language Model Pretraining with Autoregressive Blank Infilling. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 320–335.

[9] Izzeddin Gur, Hiroki Furuta, Austin Huang, Mustafa Safdari, Yutaka Matsuo, Douglas Eck, and Aleksandra Faust. 2023. A real-world webagent with planning, long context understanding, and program synthesis. *arXiv preprint arXiv:2307.12856* (2023).

[10] Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, et al. 2023. CogAgent: A Visual Language Model for GUI Agents. *arXiv preprint arXiv:2312.08914* (2023).

[11] Or Honovich, Thomas Scialom, Omer Levy, and Timo Schick. 2022. Unnatural instructions: Tuning language models with (almost) no human labor. *arXiv preprint arXiv:2212.09689* (2022).

[12] Peter C Humphreys, David Raposo, Tobias Pohlen, Gregory Thornton, Rachita Chhaparia, Alistair Muldal, Josh Abramson, Petko Georgiev, Adam Santoro, and Timothy Lillicrap. 2022. A data-driven approach for learning to control computers. In *International Conference on Machine Learning*. PMLR, 9466–9482.

[13] Jinhao Jiang, Kun Zhou, Zican Dong, Keming Ye, Wayne Xin Zhao, and Ji-Rong Wen. 2023. Structgpt: A general framework for large language model to reason over structured data. *arXiv preprint arXiv:2305.09645* (2023).

[14] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *Advances in neural information processing systems* 35 (2022), 22199–22213.

[15] Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, et al. 2019. Natural questions: a benchmark for question answering research. *Transactions of the Association for Computational Linguistics* 7 (2019), 453–466.

[16] Xiao Liu, Hanyu Lai, Hao Yu, Yifan Xu, Aohan Zeng, Zhengxiao Du, Peng Zhang, Yuxiao Dong, and Jie Tang. 2023. WebGLM: Towards An Efficient Web-Enhanced Question Answering System with Human Preferences. *arXiv preprint arXiv:2306.07906* (2023).

[17] Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xiubo Geng, Wenxiang Hu, Chongyang Tao, Jing Ma, Qingwei Lin, and Daxin Jiang. 2023. WizardCoder: Empowering Code Large Language Models with Evol-Instruct. *arXiv preprint arXiv:2306.08568* (2023).

[18] Yu Meng, Jiaxin Huang, Yu Zhang, and Jiawei Han. 2022. Generating training data with language models: Towards zero-shot language understanding. *Advances in Neural Information Processing Systems* 35 (2022), 462–477.

[19] Suraj Mishra, Peixian Liang, Adam Czajka, Danny Z Chen, and X Sharon Hu. 2019. CC-NET: Image complexity guided network compression for biomedical image segmentation. In *2019 IEEE 16th International Symposium on Biomedical Imaging (ISBI 2019)*. IEEE, 57–60.

[20] Subhabrata Mukherjee, Arindam Mitra, Ganesh Jawahar, Sahaj Agarwal, Hamid Palangi, and Ahmed Awadallah. 2023. Orca: Progressive learning from complex explanation traces of gpt-4. *arXiv preprint arXiv:2306.02707* (2023).

[21] Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, et al. 2021. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332* (2021).

[22] Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, and Li Deng. 2016. MS MARCO: A human generated machine reading comprehension dataset. *choice* 2640 (2016), 660.

[23] OpenAI. 2022. Introducing chatgpt. (2022).

[24] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems* 35 (2022), 27730–27744.

[25] Baolin Peng, Chunyuan Li, Pengcheng He, Michel Galley, and Jianfeng Gao. 2023. Instruction tuning with gpt-4. *arXiv preprint arXiv:2304.03277* (2023).

[26] Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A Smith, and Mike Lewis. 2022. Measuring and Narrowing the Compositionality Gap in Language Models. (2022).

[27] Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D Manning, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. *arXiv preprint arXiv:2305.18290* (2023).

[28] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250* (2016).

[29] Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilić, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, Matthias Gallé, et al. 2022. Bloom: A 176b-parameter open-access multilingual language model. *arXiv preprint arXiv:2211.05100* (2022).

[30] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971* (2023).

[31] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288* (2023).

[32] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Ji-Rong Wen. 2023. A Survey on Large Language Model based Autonomous Agents. *arXiv preprint arXiv:2308.11432* (2023).

[33] Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. 2023. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. *arXiv preprint arXiv:2305.04091* (2023).

[34] Xin Wang, Yudong Chen, and Wenwu Zhu. 2021. A survey on curriculum learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44, 9 (2021), 4555–4576.

[35] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022. Self-Consistency Improves Chain of Thought Reasoning in Language Models. In *The Eleventh International Conference on Learning Representations*.

[36] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems* 35 (2022), 24824–24837.

[37] Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, Rui Zheng, Xiaoran Fan, Xiao Wang, Limao Xiong, Yuhao Zhou, Weiran Wang, Changhao Jiang, Yicheng Zou, Xiangyang Liu, Zhangyue Yin, Shihan Dou, Rongxiang Weng, Wensen Cheng, Qi Zhang, Wenjuan Qin, Yongyan Zheng, Xipeng Qiu, Xuanjing Huang, and Tao Gui. 2023. The Rise and Potential of Large Language Model Based Agents: A Survey. *arXiv preprint arXiv:2309.07864* (2023).

[38] Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, and Daxin Jiang. 2023. Wizardlm: Empowering large language models to follow complex instructions. *arXiv preprint arXiv:2304.12244* (2023).

[39] Yiheng Xu, Hongjin Su, Chen Xing, Boyu Mi, Qian Liu, Weijia Shi, Binyuan Hui, Fan Zhou, Yitao Liu, Tianbao Xie, et al. 2023. Lemur: Harmonizing natural language and code for language agents. *arXiv preprint arXiv:2310.06830* (2023).

[40] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2022. ReAct: Synergizing Reasoning and Acting in Language Models. In *The Eleventh International Conference on Learning Representations*.

[41] Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Chuanqi Tan, and Chang Zhou. 2023. Scaling relationship on learning mathematical reasoning with large language models. *arXiv preprint arXiv:2308.01825* (2023).

[42] Aohan Zeng, Xiao Liu, Zhengxiao Du, Zihan Wang, Hanyu Lai, Ming Ding, Zhuoyi Yang, Yifan Xu, Wendi Zheng, Xiao Xia, et al. 2022. GLM-130B: An Open Bilingual Pre-trained Model. In *The Eleventh International Conference on Learning Representations*.

[43] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. 2022. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068* (2022).

[44] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. 2023. A survey of large language models. *arXiv preprint arXiv:2303.18223* (2023).

[45] Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. 2023. WebArena: A Realistic Web Environment for Building Autonomous Agents. In *Second Agent Learning in Open-Endedness Workshop*.

A IMPLEMENTATION DETAILS OF AUTOWEBGLM

During the SFT phase, we set the learning rate to 1e-5 with a batch size of 32. In the DPO stage, we sample the complex task dataset 20 times. After the filtering process, we build a contractional dataset of approximately 13k. We set the learning rate for the DPO to 1e-6, the batch size to 64, and the β parameter to 0.15. We add the SFT loss, weighted by a factor of 0.8. During the RFT stage, we collect samples from two diverse environments, MiniWoB++ and WebArena, resulting in successful datasets of approximately 66k and 2k, respectively, which underwent finetuning. The learning rate set for this stage was 1e-5, and the batch size was 32.

B INPUT PROMPT

Below is our input prompt for AutoWebGLM:

```
<html> {html_content} </html>

You are a helpful assistant that can assist with
    web navigation tasks.
You are given a simplified html webpage and a task
     description.
Your goal is to complete the task. You can use the
     provided functions below to interact with the
     current webpage.

#Provided functions:
def click(element_id: str) -> None:
    """
    Click on the element with the specified id.

    Args:
       element_id: The id of the element.
    """

... (Other function definitions)

#Previous commands: {previous_commands}

#Window tabs: {
    exist_window_tabs_with_pointer_to_current_tab}

#Current viewport (pages): {current_position} / {
    max_size}

#Task: {task_description}

You should output one command to interact to the
    currrent webpage.
You should add a brief comment to your command to
    explain your reasoning and thinking process.
```

C DATA CONSTRUCTION PROMPT

Data construction prompt for task and trace intent:

```
HTML:
{html_content}

I want you to act as a task generator that can
    help generate Task-Operation pairs.
Based on the above HTML webpage, I will give you a
     specified operation. Your goal is to come up
    with a ONE-STEP task that the specified
    operation can solve.
Your answer SHOULD be in the following format:

Task: {Generated one-step task}

Operation: {The right operation to solve the task}

Intention: {The intention and thinking in your
    operation}

NOTICE:
1. Your generated task should not be too SIMPLE,
    NAIVE
2. You can only do \#type\# on <input> and <
    textarea>

User's overall task: {task_description}

User's actions: {annotated_action_trace}

Based on this information, deduce the intent
    behind each of the user's actions. Your
    response should be structured as follows:
Intent of Step 1: [Describe the intent of the user
    's first action from the user's first-person
    perspective]
Intent of Step 2: [Describe the intent of the user
    's second action from the user's first-person
    perspective]
... and so on.
Note: Your response should have the same number of
     lines as the number of user actions. The
    number of user actions in this task is {
    number_of_steps_in_action}.
```

D ANNOTATION DETAILS

The annotation process was performed by 20 annotators for one month using the Google Chrome browser with our plugin installed to record their actions on assigned websites. The annotators first visited the target websites and checked whether the website descriptions matched the actual tasks. They then evaluated the tasks for clarity, relevance, achievability, complexity, and subjectivity, skipping those that didn't meet the criteria. They carefully recorded each step during a task, including any login or captcha steps. For tasks that required an answer, the annotators manually edited the responses. If a task was not doable, they could modify its description or abandon it.

E FULL RESULTS OF MINIWOB++

Table 8 is the per-task average success rate on 56 tasks from MiniWoB++.

Table 8: PER-TASK PERFORMANCE ON MINIWOB++

Task	AutoWebGLM	HTML-T5-XL	WebN-T5-XL	GPT-4	GPT-3.5-Turbo
book-flight	0.50	0.99	0.48	0.00	0.00
choose-date	1.00	0.16	0.08	0.00	0.00
choose-date-easy	1.00	1.00	1.00	0.00	0.00
choose-date-medium	1.00	0.56	0.07	0.00	0.00
choose-list	0.15	0.22	0.16	0.00	0.00
click-button	1.00	1.00	1.00	0.67	1.00
click-button-sequence	1.00	1.00	1.00	0.33	0.00
click-checkboxes	1.00	1.00	0.22	0.33	0.00
click-checkboxes-large	0.83	0.90	0.54	0.00	0.00
click-checkboxes-soft	0.37	0.99	0.08	0.00	0.00
click-checkboxes-transfer	1.00	1.00	0.63	1.00	0.00
click-collapsible	1.00	1.00	0.26	0.00	0.00
click-collapsible-2	0.76	0.93	0.27	0.00	0.00
click-color	0.74	1.00	0.34	0.67	0.00
click-dialog	1.00	1.00	1.00	0.33	0.00
click-dialog-2	1.00	0.74	1.00	0.67	0.67
click-link	1.00	1.00	0.99	0.33	0.33
click-menu	1.00	0.37	0.41	0.00	0.50
click-option	1.00	1.00	0.87	0.67	0.00
click-pie	1.00	0.96	0.51	0.67	1.00
click-scroll-list	0.57	0.99	0.98	0.00	0.00
click-shades	1.00	0.00	0.00	0.00	0.00
click-shape	0.64	0.79	0.24	0.00	0.00
click-tab	1.00	1.00	0.57	0.00	0.67
click-tab-2	1.00	0.94	0.57	0.00	0.00
click-tab-2-hard	1.00	0.88	0.12	0.33	0.00
click-test	1.00	1.00	1.00	1.00	0.00
click-test-2	0.93	1.00	1.00	1.00	1.00
click-widget	1.00	1.00	1.00	1.00	0.00
count-shape	0.65	0.67	0.64	0.00	0.00
email-inbox	1.00	1.00	0.38	0.00	0.33
email-inbox-forward-nl	1.00	1.00	0.33	0.00	0.00
email-inbox-forward-nl-turk	1.00	1.00	0.23	0.00	0.00
email-inbox-nl-turk	1.00	0.99	0.20	0.67	0.00
enter-date	1.00	1.00	0.89	0.66	0.00
enter-password	1.00	1.00	0.72	0.67	0.00
enter-text	1.00	1.00	0.89	0.67	0.00
enter-text-dynamic	1.00	1.00	1.00	0.00	0.00
enter-time	0.00	1.00	0.00	0.00	0.00
focus-text	1.00	1.00	1.00	0.00	0.00
focus-text-2	1.00	1.00	1.00	0.00	1.00
grid-coordinate	1.00	1.00	1.00	1.00	0.33
guess-number	1.00	0.13	0.00	0.00	0.00
identify-shape	1.00	1.00	0.88	0.67	0.00
login-user	1.00	1.00	0.82	0.33	0.00
login-user-popup	0.63	1.00	0.72	0.33	0.00
multi-layouts	1.00	1.00	0.83	0.33	0.00
multi-orderings	1.00	1.00	0.88	0.67	0.00
navigate-tree	1.00	0.99	0.91	0.33	0.00
search-engine	1.00	0.93	0.34	0.67	0.00
social-media	1.00	0.99	0.20	0.33	0.00
social-media-all	0.90	0.31	0.21	0.00	0.00
social-media-some	0.76	0.89	0.42	0.00	0.00
tic-tac-toe	0.74	0.57	0.48	0.00	0.00
use-autocomplete	0.85	0.97	0.02	1.00	0.67
use-spinner	1.00	0.07	0.07	0.00	0.00
Average	**0.893**	0.856	0.484	0.321	0.134