Turbocharging Web Automation: The Impact of Compressed History States

Xiyue Zhu[†*] **Peng Tang**[‡**] **Haofu Liao**[‡] **Srikar Appalaraju**[‡**]
[†]University of Illinois at Urbana-Champaign [‡]AWS AI Labs
xiyuez2@illinois.edu, tangpeng723@gmail.com, {liahaofu, srikara}@amazon.com

Abstract

Language models have led to a leap forward in web automation. The current web automation approaches take the current web state, history actions, and language instruction as inputs to predict the next action, overlooking the importance of history states. However, the highly verbose nature of web page states can result in long input sequences and sparse information, hampering the effective utilization of history states. In this paper, we propose a novel web history compressor approach to turbocharge web automation using history states. Our approach employs a history compressor module that distills the most task-relevant information from each history state into a fixed-length short representation, mitigating the challenges posed by the highly verbose history states. Experiments are conducted on the Mind2Web and WebLINX datasets to evaluate the effectiveness of our approach. Results show that our approach obtains 1.2-5.4% absolute accuracy improvements compared to the baseline approach without history inputs.

1 Introduction

The task of web automation involves performing a sequence of actions to accomplish given tasks on any website, guided by language instructions (Deng et al., 2024; Lù et al., 2024; Liu et al., 2018; Yao et al., 2022; Zhou et al., 2023; Park et al., 2025). Driven by advances in language models, web automation has attracted a lot of attention in recent years (Gur et al., 2023; Furuta et al., 2023; Cheng et al., 2024; Zheng et al., 2024; Park et al., 2025; Gao et al., 2024). These approaches leverage the current web state (i.e., web HTML and/or screenshot), history actions, and language instruction to predict the next action, obtaining promising web automation accuracy.

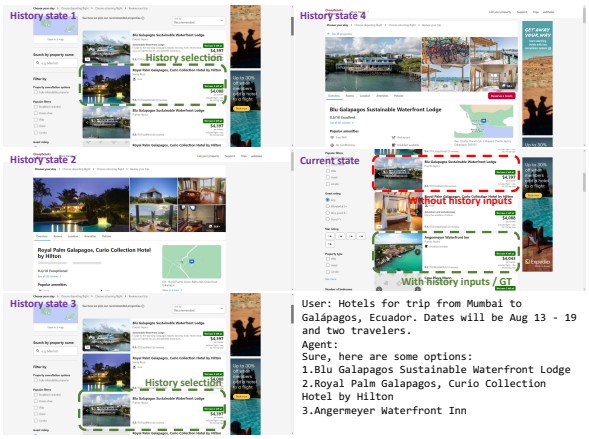

Figure 1: Example results of w/ and w/o history inputs. Without seeing the histories, the model picks the same hotel in different steps. Adding history states in model inputs correctly picks different hotels in different steps.

However, the existing approaches do not consume history states, ignoring the fact that the history states are crucial to accomplish some web automation tasks, which leads to sub-optimal accuracy, see Figure 1. This fact motivates us to explore techniques that leverage history states to improve the accuracy of web automation.

The most straightforward way to leverage history states is to concatenate history states with other inputs (i.e., the current state, history actions, and language instruction) and feed the concatenated inputs into models. But the state of real-world web pages could be very verbose (Deng et al., 2024; Lù et al., 2024), posing several challenges to benefiting web automation models from the straightforward approach: 1) **Long input sequence.** Concatenating verbose history and current states results in a long input sequence, leading to high GPU memory cost and inference latency. 2) **Sparse information in history states.** Compared to the current state, the information that is relevant to the next action is much sparser in history states. Failing to effectively distilling the sparse information from history states could adversely impact the accuracy.

[*]The work was done when Xiyue Zhu was an intern at Amazon. [**]Peng Tang and Srikar Appalaraju are the corresponding authors.

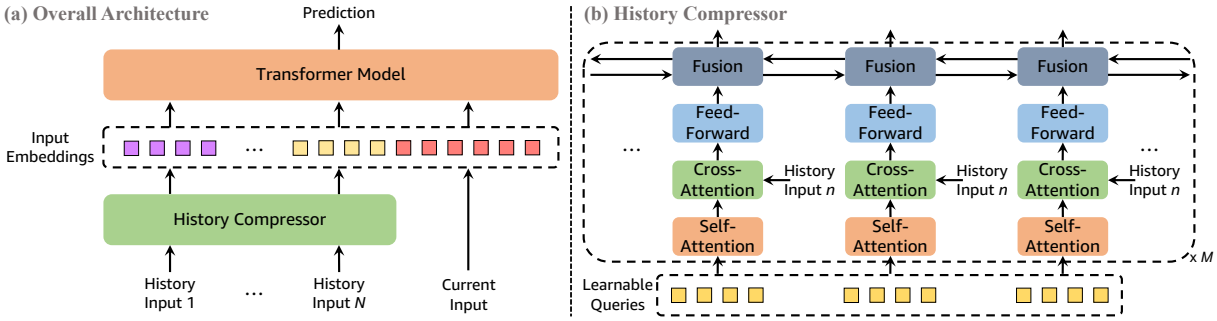

Figure 2: (a) Overall architecture of our model. Our model takes the current input and N history inputs as input, where the history inputs are fed into a history compressor before being fed into the transformer model. The next action is predicted based on the inputs. (b) The architecture of the history compressor. For each history, the history compressor takes a fixed-length sequence of learnable queries and one history input as inputs with a history fusion module that fuses information among different history inputs, and outputs the representations of the learnable queries.

To address these challenges, we propose Web History Compressor, a novel approach to turbocharge web automation using history states. Instead of feeding verbose history states into models directly, our approach trains a history compressor to compress each history state into a fixed-length short representation and extract the most relevant information. Inspired by Perceiver (Jaegle et al., 2021), for each history state, the history compressor takes a fixed number of learnable queries, the history state, history actions, and language instructions as inputs, and outputs the representations of the learnable queries, effectively reducing the sequence length of the history state to the fixed number. In addition, in the history compressor, the learnable queries cross-attend to the history state, history actions, and language instructions, with information fusion among different history inputs, allowing the history compressor to distill the most task-relevant information into the representations of the learnable queries, guided by the language instructions. The resulting compressed representations of each history state are concatenated with other inputs (i.e., the current state, history actions, and language instruction). The concatenated inputs are fed into the model to predict the next action.

Experiments are carried out on the challenging Mind2Web (Deng et al., 2024), and WebLINX (Lù et al., 2024) datasets. Our approach shows 1.2-5.4% absolute accuracy improvements on the Mind2Web and WebLINX datasets across different evaluation metrics compared to the state-of-the-art MindAct approach (Deng et al., 2024), confirming the effectiveness of our approach.

2 Approach

Figure 2 (a) shows the architecture of our model. Our model takes the current input and N history inputs as input, where each input consists of state (i.e., web HTML and/or screenshot), history actions, and language instruction describing the task. A history compressor module is employed to compress each history input into a fixed-length representation, see Section 2.1. Here, the history compressors for different history inputs share the same model weights. The compressed representations of history inputs are concatenated with the current inputs, forming the input to a transformer model. The transformer model then predicts the next action based on the inputs. We apply a history compressor to history inputs only, because the current input is highly relevant to the task so we want to keep as much information in the current input as we can.

2.1 History Compressor

The states of the real-world web pages could be highly verbose (Deng et al., 2024; Lù et al., 2024), leading to long input sequences and sparse information in history states if we feed the history states into models directly. Having an approach that can effectively distill the sparse information into compact representations can not only reduce the input sequence length but also improve model accuracy. To address this, we propose a history compressor module to handle the verbose history state.

Inspired by Perceiver (Jaegle et al., 2021), each history compressor layer consists of a self-attention module, a cross-attention module, a feed-forward module, and a history fusion module. Specifically, for each history input, a fixed number of learnable queries is first fed into the self-attention module, and then input to a cross-attention module that

cross-attends to one history input, followed by a feed-forward module to enhance the learned representations. A history fusion module is next applied to fuse the information from the current history inputs and the neighboring history inputs. The history fusion module operates by concatenating features from various history inputs along the channel dimension, followed by a fully-connected layer for dimension reduction. The weights of learnable queries and different modules are shared across different history inputs. See Figure 2 (b).

With the history fusion module, different history inputs can communicate with each other to reduce the redundant information that are shared among different history inputs and learn the most useful information. With the guidance from the language instructions in the history input, the history compressor is able to distill the most task-relevant information into the learned representations. By stacking M history compressor layers, the learned representations of the fixed number of learnable queries are further enhanced for the task described in the language instruction.

2.2 Implementation Details

Base Model We use MindAct (Deng et al., 2024) as our base model. Specifically, we use flan-T5-base (Raffel et al., 2020; Chung et al., 2024) as our transformer model, which consists of encoder and decoder layers, and we use pruned HTML as the web state, following MindAct. Here the weights of History Compressor are randomly initialized.

Training The representations of history inputs are derived from the outputs of the history compressor. However, due to the absence of a history compressor for the current input, there is a misalignment between the different representations in the feature space. This misalignment causes unstable training if we train all modules of our model jointly. To mitigate this issue, we adopt a two-stage training strategy. Specifically, in the first-stage training, we freeze the transformer model and train only the history compressor. In addition, zero-initialized attention (Zhang et al., 2024), which assigns zero attention scores to history input representations at the beginning of training and gradually learns a factor that controls the attention weight assigned to these representations, is applied. This approach allows the model to train the history compressor and align the representations of history inputs with those of the current input progressively, fostering a more stable and manageable training process. Subsequently, in the second stage of training, we train all the modules together.

Table 1: Results of different approaches on the Mind2Web dataset. MindAct is the baseline approach without history inputs (Deng et al., 2024). Pruning and LLM correspond to the alternative pruning-based compressor and zero-shot LLM compressor, respectively. We also use LLMLingua (Jiang et al., 2023), an existing work for LLM prompt compression. Ours indicates our history compressor approach. More details are in Section 2.3.

	Element Acc (↑)	Marco Element Acc (↑)	Step Acc (↑)	Macro Step Acc (↑)
Cross-Task Split				
MindAct	40.78	42.50	37.54	39.35
Pruning	38.54	41.39	36.15	38.81
LLM	34.53	39.07	32.04	36.37
LLMLingua	39.72	42.37	38.27	39.23
Ours	**45.80**	**47.15**	**41.83**	**43.47**
Cross-Website Split				
MindAct	29.57	31.96	26.37	28.54
Pruning	29.57	33.13	26.88	30.38
LLM	24.91	29.05	22.14	25.73
LLMLingua	30.27	33.79	23.38	26.73
Ours	**32.17**	**35.71**	**28.73**	**31.83**
Cross-Domain Split				
MindAct	31.40	32.47	28.54	29.78
Pruning	31.79	33.54	29.00	30.94
LLM	29.27	29.28	26.70	**31.80**
LLMLingua	31.56	32. 97	28.79	30.52
Ours	**32.65**	**33.90**	**29.70**	30.99

2.3 Alternative Approaches

There are alternative approaches for history compressors, including the pruning-based and zero-shot LLM compressors. For the pruning-based compressor, we follow the prior work (Deng et al., 2024; Lù et al., 2024) by employing an off-the-shelf language model to rank and select top-50 HTML elements. For the zero-shot LLM compressor, we carefully prompt strong LLMs (e.g., Claude-3 Haiku used here), with HTML, history actions, and language instruction as inputs, to summarize each history HTML. In addition, we use previous work for LLM prompt compression, LLMLingua (Jiang et al., 2023), which uses pretrained LLMs to identify and remove non-essential tokens. Similar to our approach, N history inputs are used for the two approaches. However, without task-specific training, these alternative approaches fail to distill the most task-relevant information into the compressed representations, resulting in unsatisfactory accuracy, see Section 3.2.

Table 2: Results of different approaches on the WebLINX test-iid dataset. MindAct is the baseline approach without history inputs (Deng et al., 2024). PPruning and LLM correspond to the alternative pruning-based compressor and zero-shot LLM compressor, respectively, see Section 2.3. Ours indicates our history compressor approach.

	Overall Micro Avg (↑)	Overall Intent-Match (↑)	Element-group IoU (↑)	Text-Group F1 (↑)
MindAct	32.03	83.91	31.93	30.52
Pruning	31.37	83.32	30.65	28.61
LLM	31.24	82.90	31.29	29.01
Ours	**34.72**	**88.35**	**37.33**	**32.57**

Table 3: Inference GPU memory cost and latency comparison among different approaches on WebLINX. MindAct is the baseline approach of without history inputs (Deng et al., 2024). Pruning corresponds to the alternative pruning-based compressor described in Section 2.3. Ours indicates our history compressor approach. Experiments are performed on a single NVIDIA A100-SXM4-80GB GPU.

	Inference GPU Memory (GB/sample ↓)	Inference Latency (s/demonstration ↓)	Average # tokens / history	Maximum # tokens / history
No compressor	-	-	4065 ± 318	4096
MindAct	**25.60**	**3.47**	$\mathbf{0 \pm 0}$	**0**
Pruning	64.20	8.42	1290 ± 595	2048
LLM	<u>32.12</u>	7.25	<u>178 ± 32</u>	279
Ours	38.30	<u>6.21</u>	256 ± 0	<u>256</u>

3 Experiments

3.1 Experimental Setup

Datasets and Evaluation Metrics We benchmark our model on the challenging Mind2Web (Deng et al., 2024) and WebLINX (Lù et al., 2024) datasets. Following the official setups, we use element accuracy, macro element accuracy, step accuracy, and macro step accuracy as the evaluation metrics for Mind2Web, and overall micro average accuracy, overall intent-match accuracy, element-group IoU accuracy, and text-group F1 accuracy as the evaluation metrics for WebLINX. We follow the official train/test splits for these two datasets. Please see Appendix B for more details of datasets and evaluation metrics.

Hyper-Parameter Setups For each history compressor, 256 learnable queries with feature dimension 768 are used, and there are 2 history compressor layers. At most 5 history inputs are consumed by the model (i.e., $N = 5$), see Appendix C.1 for the impact of the number of history inputs.

3.2 Main Results

The results presented in Table 1 demonstrate the effectiveness of our proposed history compressor approach on the Mind2Web (Deng et al., 2024) dataset. Across various evaluation metrics, our approach shows 1.2-5.0% accuracy improvement compared to the baseline approach of without history inputs. This improvement confirms the ability of our approach to learn task-relevant representations for history inputs, thereby enhancing web automation performance. In contrast, the alternative approaches, namely the pruning-based compressor and the zero-shot LLM compressor, yield mixed results. While these approaches outperform the baseline on certain metrics, they also exhibit lower accuracy on others. This is because, without task-specific training, these approaches cannot learn the most task-relevant information as the compressed representations, confirming the effectiveness of our approach. Consequently, the superior performance of our approach highlights its effectiveness in leveraging history states for web automation tasks. In addition, different splits in Mind2Web evaluate the zero-shot generalization capabilities of the model (see Appendix B). Our approach consistently show improvements across different splits, confirming the generalization capabilities of our approach.

The results on WebLINX (Lù et al., 2024) shown in Tab. 2 show the same conclusion as on Mind2Web. In particular, our approach shows consistent performance gains over the baseline approach of without history inputs, with 2.0-5.4% accuracy improvements. Furthermore, our approach outperforms the pruning-based and zero-shot LLM compressor approaches by a substantial margin of 3.4-6.0%. These results confirm the effectiveness of our approach. The consistent improvements observed across multiple datasets and evaluation metrics provide compelling evidence of the robustness of our approach, further solidifying its effectiveness in web automation tasks.

3.3 Inference Cost Analysis

We study the inference GPU memory cost, latency, and the number of input tokens here. As shown in Tab. 3, our approach obtained 40.3% lower GPU memory cost, 26.2% lower latency, and 5+ times fewer input tokens compared to the pruning-based compressor. This is because our approach effectively compresses the verbose history inputs into fixed-length learned representations, contributing to minimal inference overhead from including history inputs. Our approach has comparable GPU memory cost and latency compared to the zero-shot LLM compressor, with much higher web automation accuracy as shown in Tab. 1 and Tab. 2. We are not able to run training / inference for using

history inputs without any compression due to the GPU memory limitation.

4 Conclusion

In this paper, we propose a novel web history compressor approach to improve web automation using history states. Our approach trains a history compressor module to compress each highly verbose history state to a fixed-length short representation, maintaining the most task-relevant information meanwhile. Experimental results show that our approach surpasses the baseline approach of without history inputs by 1.2-5.4% on the Mind2Web and WebLINX datasets. In the future, we will apply our approach to stronger transformer models and models that can take different modalities (e.g., image, image + text) as inputs, and apply our approach to more web automation datasets.

5 Limitations

Although our approach shows consistent improvements over the MindAct approach, theoretically, it can be applied to any base web automation models with any modality inputs, e.g., the stronger transformer model Llama 3 (Dubey et al., 2024) and the model with image modality inputs (Cheng et al., 2024) or multi-modality inputs (Kil et al., 2024). Applying our approaches to these models could further confirm the effectiveness of our approach. In addition, although History Compressor gives significant accuracy boost for web automation, it also leads to 79% higher latency compared to no history inputs. In the future, we will explore ways to reduce inference latency overhead, e.g., reducing the size of the history compressor model and reducing the number of learnable queries, etc.

References

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*.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. 2024. Scaling instruction-finetuned language models. *Journal of Machine Learning Research*, 25(70):1–53.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su. 2024. Mind2web: Towards a generalist agent for the web. *Advances in Neural Information Processing Systems*, 36.

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. *Preprint*, arXiv:2306.06070.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Hiroki Furuta, Kuang-Huei Lee, Ofir Nachum, Yutaka Matsuo, Aleksandra Faust, Shixiang Shane Gu, and Izzeddin Gur. 2023. Multimodal web navigation with instruction-finetuned foundation models. *arXiv preprint arXiv:2305.11854*.

Yuan Gao, Kunyu Shi, Pengkai Zhu, Edouard Belval, Oren Nuriel, Srikar Appalaraju, Shabnam Ghadar, Zhuowen Tu, Vijay Mahadevan, and Stefano Soatto. 2024. Enhancing vision-language pre-training with rich supervisions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 13480–13491.

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*.

Andrew Jaegle, Felix Gimeno, Andrew Brock, Andrew Zisserman, Oriol Vinyals, and Joao Carreira. 2021. Perceiver: General perception with iterative attention. *Preprint*, arXiv:2103.03206.

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. 2023. LLMLingua: Compressing prompts for accelerated inference of large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 13358–13376, Singapore. Association for Computational Linguistics.

Jihyung Kil, Chan Hee Song, Boyuan Zheng, Xiang Deng, Yu Su, and Wei-Lun Chao. 2024. Dual-view visual contextualization for web navigation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 14445–14454.

Evan Zheran Liu, Kelvin Guu, Panupong Pasupat, Tianlin Shi, and Percy Liang. 2018. Reinforcement learning on web interfaces using workflow-guided exploration. *arXiv preprint arXiv:1802.08802*.

Xing Han Lù, Zdeněk Kasner, and Siva Reddy. 2024. Weblinx: Real-world website navigation with multi-turn dialogue. *arXiv preprint arXiv:2402.05930*.

Xing Han Lù, Zdeněk Kasner, and Siva Reddy. 2024. Weblinx: Real-world website navigation with multi-turn dialogue. *Preprint*, arXiv:2402.05930.

Joonhyung Park, Peng Tang, Sagnik Das, Srikar Appalaraju, Kunwar Yashraj Singh, R. Manmatha, and Shabnam Ghadar. 2025. R-vlm: Region-aware vision language model for precise gui grounding. In *Findings of ACL*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. 2022. Webshop: Towards scalable real-world web interaction with grounded language agents. *Advances in Neural Information Processing Systems*, 35:20744–20757.

Renrui Zhang, Jiaming Han, Chris Liu, Peng Gao, Aojun Zhou, Xiangfei Hu, Shilin Yan, Pan Lu, Hongsheng Li, and Yu Qiao. 2024. Llama-adapter: Efficient fine-tuning of language models with zero-init attention. *Preprint*, arXiv:2303.16199.

Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. 2024. Gpt-4v (ision) is a generalist web agent, if grounded. *arXiv preprint arXiv:2401.01614*.

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. *arXiv preprint arXiv:2307.13854*.

A Inputs and Outputs of the Transformer Model

Here we provide details of the inputs and outputs of the transformer model in Figure 2. The transformer model used here is a Flan-T5 (Raffel et al., 2020; Chung et al., 2024) based encoder-decoder model. The inputs to the encoder layers are the concatenation of the representations of history inputs outputted from History Compressor, HTML representations of the current web state which are structured text inputs, and natural language instructions that describe the tasks to be accomplished. These inputs are processed through a series of encoder layers to produce encoded representations, which are then fed into the cross-attention layers of the decoder. The decoder performs autoregressive generation to produce the required output for task completion.

B Datasets and Evaluation Metrics

Mind2Web (Deng et al., 2023) has over 2,000 open-ended tasks from 137 websites and 31 domains. There are 1,009 tasks from 73 websites for the training set. We evaluate our model on all 3 test splits: Cross-Task, Cross-Website, and Cross-Domain splits. The Cross-Task split contains 252 tasks from 69 websites, and the training and testing sets contain websites from similar domains but different tasks to evaluate zero-shot task generalization. The Cross-Website split contains 10 websites and 177 tasks, and the training and testing sets contain websites from different domains but similar tasks to evaluate zero-shot domain generalization. The Cross-Domain split contains 912 tasks from 73 websites, and the training and testing sets contain websites from different domains and different tasks to evaluate zero-shot generalization across both website domains and task. We evaluate our model using Element Accuracy and Step Success Rate. In the Mind2Web benchmark, the model is asked to select an element on the current HTML webpage and predict the action to perform on that element. Element Accuracy (element acc) compares the selected element with all acceptable elements, and the Step Success Rate (step acc) considers a step to be successful only if both the selected element and the predicted operation are correct. The raw metrics are calculated by averaging over each step. In contrast, the macro metrics average across tasks, each with a sequence of steps. Therefore, the macro metrics will weigh more on tasks consisting of short

Table 4: Results of different numbers of history inputs of our approach on the Mind2Web dataset. 0 history means the MindAct baseline.

	Element Acc (↑)	Marco Element Acc (↑)	Step Acc (↑)	Macro Step Acc (↑)
Cross-Task Split				
0 history	40.78	42.50	37.54	39.35
1 history	42.26	42.90	38.12	40.89
2 histories	43.84	45.74	39.52	42.64
3 histories	43.36	43.64	38.92	41.78
4 histories	44.69	46.68	40.21	43.12
5 histories	**45.80**	**47.15**	**41.83**	**43.47**
Cross-Website Split				
0 history	29.57	31.96	26.37	28.54
1 history	29.08	31.43	25.96	28.18
2 histories	30.32	32.79	27.81	28.82
3 histories	31.21	33.89	28.13	31.43
4 histories	32.01	35.60	28.36	31.52
5 histories	**32.17**	**35.71**	**28.73**	**31.83**
Cross-Domain Split				
0 history	31.40	32.48	28.54	29.78
1 history	31.78	32.98	28.78	30.01
2 histories	31.98	33.13	29.45	30.33
3 histories	32.36	33.32	29.36	30.12
4 histories	32.45	33.68	29.62	30.51
5 histories	**32.65**	**33.90**	**29.70**	**30.99**

Table 5: Results of different numbers of tokens used to represent each history input after compression on the Mind2Web dataset.

	Element Acc (↑)	Marco Element Acc (↑)	Step Acc (↑)	Macro Step Acc (↑)
Cross-Task Split				
64 tokens	44.29	45.67	40.03	42.39
128 tokens	45.08	46.36	41.07	42.97
512 tokens	45.49	46.90	41.36	43.02
256 tokens	**45.80**	**47.15**	**41.83**	**43.47**
Cross-Website Split				
64 tokens	31.14	34.56	27.45	30.52
128 tokens	31.70	35.01	28.34	31.42
512 tokens	31.56	34.84	28.02	31.13
256 tokens	**32.17**	**35.71**	**28.73**	**31.83**
Cross-Domain Split				
64 tokens	31.98	33.14	29.39	30.48
128 tokens	32.47	33.76	29.58	30.74
512 tokens	32.01	33.24	29.23	30.32
256 tokens	**32.65**	**33.90**	**29.70**	**30.99**

sequences of actions.

WebLINX (Lù et al., 2024) contains 2,337 demonstrations from 155 real-world websites. There are 969 demonstrations with 43,538 turns in the training set. We evaluate our model on the Test_iid split, containing 100 demos and 4,318 turns with similar tasks and websites in the training set to test the in-domain generalization of the model. The WebLINX benchmark requires text input from the model in **Text-Group** actions, such as load and say. It also contains **Element-Group** actions, such as click and submit, which require the model to select elements to perform the action. The metrics element group IoU and text group F1 calculate whether the text is correctly input and the elements are correctly selected, respectively. Intent match estimates whether the model can correctly predict which action to perform using accuracy. If an action is not correctly predicted in a certain step, the element group iou and text group F1 is 0 for this step. The overall micro average equals element group iou and text group F1, depending on the ground truth actions.

Table 6: Results of without and with the history fusion module of our approach on the Mind2Web dataset. 'With fusion' means our full approach.

	Element Acc (↑)	Marco Element Acc (↑)	Step Acc (↑)	Macro Step Acc (↑)
Cross-Task Split				
Without fusion	44.08	45.72	40.78	42.36
With fusion	**45.80**	**47.15**	**41.83**	**43.47**
Cross-Website Split				
Without fusion	**32.56**	**36.15**	**28.84**	**31.95**
With fusion	32.17	35.71	28.73	31.83
Cross-Domain Split				
Without fusion	32.47	33.58	29.59	30.84
With fusion	**32.65**	**33.90**	**29.70**	**30.99**

Table 7: Results of without and with the history fusion module of our approach on the WebLINX test-iid dataset. 'With fusion' means our full approach.

	Overall Micro Avg (↑)	Overall Intent-Match (↑)	Element-group IoU (↑)	Text-Group F1 (↑)
Without fusion	34.12	87.65	36.03	32.02
With fusion	**34.72**	**88.35**	**37.33**	**32.57**

C Ablation Studies

Here we conduct experiments to analyze the impact of the number of history inputs, the length of each compressed history state, and the history fusion module.

C.1 The Impact of the Number of History Inputs

We study the impact of the maximum number of histories on the Mind2Web dataset here. The pruning-based compressor is used as the compression approach. As shown in Tab. 4, there is an overall trend that the more history inputs we incorporate, the better accuracy we can get. The experiment results show that using history states can turbocharge web automation, which confirms our motivation that history states are important for web automation.

C.2 The Impact of the Length of Each Compressed History State

We investigate how the length of each compressed history state affects performance on the Mind2Web dataset. Our q-former-based compressor enables flexible control over the compression length by adjusting the number of learnable queries. As shown in Tab. 5, the optimal length of history states is 256. Using shorter compressed histories can lead to excessive information loss, resulting in sub-optimal performance. Conversely, longer compressed histories may dilute relevant information, making it harder for the web agent to utilize the past context effectively.

C.3 The Impact of the History Fusion Module

Tab. 6 and Tab. 7 show the impact of the history fusion module on the Mind2Web and WebLINX datasets. The integration of the history fusion module yields consistent improvements in web automation accuracy across the majority of test cases compared to without fusion. These results validate our hypothesis that the history fusion module effectively facilitates communication between different history inputs, enabling the model to extract and utilize the most relevant historical information for web automation.