AutoTaskGen: Automatic Generation and Evaluation of Novel Web Agent Tasks

David Wang*, Ryan He*, Deepak Nathani, William Wang
University of California, Santa Barbara
Santa Barbara, CA, USA
{d_wang, ryanhe}@ucsb.edu

Abstract

While large language models have grown more powerful, they still struggle with realistic agent tasks, which are comparatively natural for humans. This is compounded by the fact that agentic behavior is out of distribution for base models, requiring precise prompting (Yao et al., 2023b) or specialized finetuning (AgentTuning, Agent-FLAN) to achieve. One large challenge to creating more powerful agents is the lack of challenging training data. Many commonly used datasets (MiniWob, Mind2Web) are oversimplified, while more realistic ones (WebArena, AppWorld) simply do not contain enough samples to tune an agent. In other domains, In this paper, we propose AutoTaskGen, a novel framework in which we give an agent context about the environment they are in and allow it to generate its own novel tasks to complete, along with evaluation functions using the environment's backing database. This is a difficult task because tasks must be both *achievable* and *correct*. After human annotation, we find that 33% of the tasks are both achievable and correctly evaluated. This framework constitutes a new scalable way to generate web agent tasks without manual human writing, just supervision.

1 Introduction

Online activity has become an essential part of our daily lives. From social media and online shopping to collaborative tools like GitHub and Google Docs, these utilities are deeply integrated into our routines. It would hardly be an exaggeration to say that modern life depends on them. However, many of these activities remain time-consuming, a challenge that could be addressed through autonomous agents.

These agents, powered by natural language commands, have the potential to significantly enhance human efficiency and quality of life. Yet, training such agents requires a substantial amount of task data, which is currently insufficient. This is partly because creating and verifying such tasks is both complex and resource-intensive. Additionally, many available tasks are overly simplistic, making them ineffective for training and evaluating advanced agents.

To address this, we investigate whether large language models (LLMs) can generate novel tasks for web agents. We observe that a naive approach of repeatedly prompting the same LLM is ineffective as they are not grounded in the site definition. Introducing randomness through a non-zero temperature in task generation mitigates some issues but often leads to duplicate or invalid tasks.

To overcome these challenges, we propose **AutoTaskGen**, a pipeline for automatically generating synthetic tasks. **AutoTaskGen** extracts the action space from each website, then employs a random sampler to select a set of actions as preconditions for each generated task. These preconditions are provided to an LLM, enabling it to generate task templates. Each task is synthetically validated, and a subset has been manually annotated to ensure correctness. This process results in a new dataset with verified accuracy.

To evaluate these tasks, we introduce a schema that leverages the website's database for automatic evaluation. While not all tasks are easily evaluable, many can be assessed deterministically using the website's database. Specifically, these tasks fall into two categories:

- Query tasks: These request information about the website's current state, such as the number of items in a shopping cart.
- OP tasks: These modify the website's state, such as adding n items to a shopping cart.

We also observe that LLMs are generally proficient

*Denotes equal contribution.

at generating valid SQL statements (Li et al., 2023a; Lei et al., 2024). Therefore, during task generation, we provide the database schema of each website to the model, enabling it to create appropriate SQL queries for evaluating each task.

In summary, we propose the following contributions:

- We propose a pipeline to demonstrate the feasibility of LLMs to automatically generate **correct** and **achievable** tasks to create a **challenging** dataset.
- We create a dataset consisting of **x** samples, **y** of which have been human-verified to be correct.
- We introduce a novel schema that uses SQL queries to automatically verify whether a task has been successfully completed.

2 Related Work

Language Models as Agents With the increase in capabilities of large language models, recent research has demonstrated capability for language models to act as agents. This is achieved via giving language model access to tools that allow them to interact with an external environment, interleaving thinking and actions (Yao et al., 2023b). Through this, language models have demonstrated a strong capacity for multistep reasoning.

However, challenges remain. Language agents often struggle to recover from Recent work has explored ways to improve language model agent performance via techniques such as self-reflection (Pan et al., 2024), finetuning on agent trajectories (Zeng et al., 2024), or memorizing previous workflows (Wang et al., 2024)

Benchmarks for Web Agents In order to evaluate how well different language models and frameworks perform in agentic settings, work has been done to produce concrete and reproducible benchmarks for web agent tasks. One specific form of such benchmarks are environment based, in which agents are given access to a simulation environment and allowed to execute actions. At the end, an evaluation function is used to determine whether the task was completed successfully or not. Such benchmarks span varying levels of difficulty. Earlier works such as MiniWob++ (Liu et al., 2018) or WebShop (Yao et al., 2023a) tend to greatly simplify the tasks and environment. More recent works such as WebArena (Zhou et al., 2024) or AppWorld (Trivedi et al., 2024) propose much more realistic tasks and generalizable evaluations. However, manually writing new tasks and evaluation functions for such environments is expensive and time-consuming.

Synthetic Data Data quality has shown to be critical for performance, as studies have shown that a small amount of high-quality data can be used to train a smaller model that significantly performs other models with the same amount of parameters (Li et al., 2023b). However, generating such data via manual annotation is time-consuming and expensive.

Recently, the usage of synthetic data to train and evaluate large language models has shown promise to address issues of data quantity and data quality (Wang et al., 2021). One common set-up is to use a large foundation model to generate synthetic data directly (Long et al., 2024). Schemes have been introduced combining synthetic data generation with human verification to ensure that generated data is verified (Pangakis et al., 2023).

Recent works have also shown potential to use language model to generate entirely new environments and tasks for language agents (Hu et al., 2024). In contrast to other works, this paper presents a method of data generation that is *grounded* in an existing environment.

3 Task Definition

A task consists of an **intent** and a **query**. An intent is a natural language task instruction that tells a web agent what to do. For example, `Create a new forum named "machinelearning" with the description "all things ML!"`. A query is a function that can be executed to evaluate whether or not the intent was completed successfully. Recent benchmarks have shown that large language models have competitive performance on realistic SQL tasks (Li et al., 2023a; Lei et al., 2024). Meanwhile, language agents do not yet meet human baselines on complex benchmarks. Thus, we leverage the strength of language models and have them generate evaluation queries as SQL queries that evaluates whether the web agent has successfully completed the intent or not. For example, `"SELECT EXISTS (SELECT 1 FROM forums WHERE name = 'name' AND description = 'description');"`.

Web agent tasks can be broadly divided into task categories - information-seeking, site navigation,

and content and configuration operation. (Zhou et al., 2024). In this paper, we focus on information-seeking (QA) and content and configuration operation (OP) tasks, as they can be directly checked via the backing database of an environment, while site navigation cannot.

We finally introduce the notion of task depth. Within a web environment, there are a number of distinct actions that an agent can perform. Task depth refers to how many such distinct actions are included in a given task. Tasks with higher depth are considered more complex, as they require the agent the execute more distinct workflows.

QA Task with Depth 1

Intent: What is the username of the user who made the top-1 submission in the books forum?

Query:

```
SELECT u.username
FROM submissions s
JOIN users u ON s.user_id = u.id
JOIN forums f ON s.forum_id = f.id
WHERE f.name = 'books'
ORDER BY s.net_score DESC
LIMIT 1;
```

OP task with depth 2

Intent: In the GitLab website, create a new group with the name `'developers'`, then create a new label within that group with the name `'maintenance'`.

Query:

```
SELECT EXISTS (
SELECT 1
FROM labels l
JOIN namespaces g ON l.group_id = g.id
WHERE g.name = 'developers'
AND l.title = 'maintenance'
);
```

3.1 Task Template

From a single action type, you can generate multiple task instances with different variables. For example, from the previous section, you could take the intent `Create a new forum named "machinelearning" with the description "all things ML!"` and replace the forum name and description with a different string to get a new intent, such as `Create a new forum named "cooking" with the description "fantastic flavors"`. The convenient thing here is that the query can similarly be updated based on the new variable values. In general, we can form intent or query templates by writing them with variable fields to be filled. You can then instantiate the task template by specifying variables values for each field. This allows a single valid task template to produce many valid task instances. Recent studies (Zhou et al., 2024; Trivedi et al., 2024; Mirzadeh et al., 2024) have shown that language model agents are not robust to variations of the same task template, so multiple instances from the same template are still useful for training and evaluating the agent.

Task Template

Intent Template: What is the best-selling product in month {{month}} of {{year}}?
Query Template:

```
SELECT product_name FROM
sales_bestsellers_aggregated_monthly
WHERE YEAR(period) = {{year}}
AND MONTH(period) = {{month}}
ORDER BY qty_ordered DESC
LIMIT 1;
```

Variables:
month: [1, 2, 3, ...]
year: [2022, 2023, 2024]

Intent: What is the best-selling product in month 1 of 2022?
Query:

```
SELECT product_name FROM
sales_bestsellers_aggregated_monthly
WHERE YEAR(period) = 2022
AND MONTH(period) = 1
ORDER BY qty_ordered DESC
LIMIT 1;
```

4 AutoTaskGen Pipeline

To start the task generation process, we must first ground our generation in the environment we are generating for. The first step is to extract a high level summary of the website description and available actions for each given environment. We then select up to k actions depending on the depth k, which is a deterministic variable that can be set

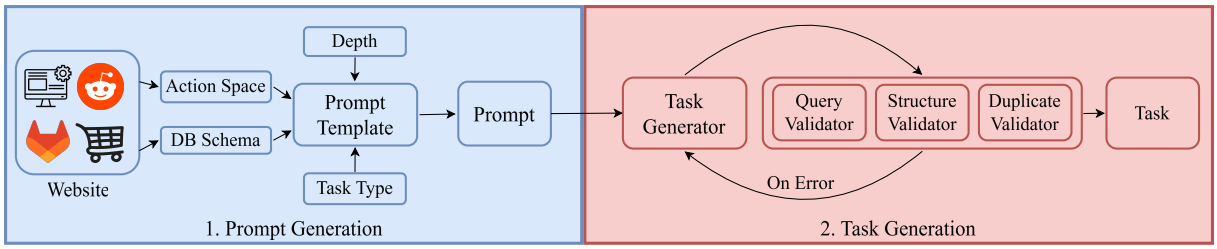

Figure 1: Overview of the pipeline to generate tasks. We first ..., then... .

for each generation. This determines how many distinct actions from the actions space must be explicitly included in the generated task. Based on the selected actions, we then extract the most relevant table from the environment database. The website description, available actions, relevant table schemas, specified actions, and few-shot examples are then input into the language model to generate a task template in natural language using Chain of Thought prompting (Wei et al., 2022). The result is then parsed into a structured task template.

4.1 Environment Extraction

4.2 Task Validation

One key feature of a high quality dataset is data diversity. To this end, we need to ensure that our data generation pipeline does not generate multiple version of the semantically same task. To this end, we use a sentence embedding model to embed all previously generated tasks. Whenever we generate a new task that is too similar to a previously generated one, we prompt the model to regenerate it.

Each task template has at least one SQL query used to evaluate task success. It may also have additional ones to select variable values. Large language models may generate syntactically or semantically invalid queries. As part of AutoTaskGen, we execute any generated queries to ensure they are grounded in the environment. If the query is invalid, this will result in execution feedback.

4.3 Naive Baseline

To evaluate how well our framework does in terms of reducing duplicates and generating valid tasks, we construct a naive baseline in which our Large Language Model is asked to generate new tasks directly based on few shot examples and website context, without the additional steps of extracting and selecting specific actions to include, grounding the generation in the database schema, and retrying generation based on execution feedback.

We plot below the number of errors that were caught via our pipeline on a set of naively generated task templates.

5 Evaluation

5.1 Correctness

For a task instance to be completely valid, it must be correct across both its intent and query.

Intent correctness measures if it meets the following criteria. First, it must be achievable. The agent must be able to perform the task with its given level of access. Some tasks may technically be achievable for some user in the environment but not the one the agent is operating. Second, the task must be objective. Given a trajectory of the agent interacting with the environment, it must be definitively possible to say whether the intent was successfully completed or not. For example, *delete the user named 'thoughtbot'* would be an invalid task in the gitlab environment because the agent does not have the power to

Query correctness measures if it correctly evaluates whether the intent was completed successfully or not. If the intent is incorrect, then the query is also incorrect by default.

A correct task instance is one with both a correct instance and correct query. Since all task instances with invalid intent also have invalid queries, this means that all task instances with correct queries are correct. A task template is correct if and only if the resulting task instance is correct for any combination of its variable values. A correct task template will always product correct task instances. An incorrect task template may also produce some valid task instances, so long as it is not correct for at least one combination of variables.

5.2 Experimental Setup

To evaluate the correctness of tasks, we annotated a total of 960 task instances. Before starting the annotation process, we spent time familiarizing

Method	Syntax (%)	SQL (%)	Duplicate (%)	Overall (%)
Naive Prompt	75	60	50	62
AutoTaskGen	90	85	80	85

Table 1: Success Rates for Syntax, SQL, Duplicate Tasks, and Total

Temperature	Task Part	Forum	Shopping	Admin	GitLab	Total
0.0	INTENT	90.00	73.33	61.67	34.17	**64.79**
	QUERY	72.22	40.91	16.22	85.37	**51.77**
	Overall	65.00	30.00	10.00	29.17	**33.54**
0.7	INTENT	79.17	61.67	58.33	30.00	**57.29**
	QUERY	58.95	54.05	24.29	86.11	**52.36**
	Overall	46.67	33.33	14.17	25.83	**30.00**

Table 2: Correctness: Evaluates the overall task accuracy, defined as the percentage of instances where the intent is correctly identified. Query: From the tasks with correctly identified intents, this measures the percentage with accurate SQL queries. Overall: Reflects the percentage of task instances where both the intent and SQL query are correct.

Temperature	Task Type	Intent	Query	Total
0.0	OP	55.83	49.25	27.50
	QA	73.75	53.67	39.58
0.7	OP	45.42	46.79	21.25
	QA	69.17	56.02	38.75

Table 3: Results by Temperature and Task Type

ourselves with the websites to ensure accuracy.

5.3 Analysis

We analyze our method across three main variables: temperature, task type, and environment.

Temperature We analyzed all generated task instances (Table 2) (Table 3) at both the intent and query levels. At the temperature level, `gpt-4o` with a temperature setting of 0.0 exhibited significantly better performance in intent correctness compared to a temperature of 0.7. This result aligns with expectations, as lower temperature values tend to produce more stable and deterministic outputs.

Environment Breaking down the results by environment, the forum environment consistently outperformed all others in intent correctness, followed by shopping, shopping admin, and GitLab. We hypothesize this pattern is driven by varying levels of complexity across environments.

Forums generally excel due to their straightforward structure, text-centric nature, and high representation in training datasets. Their layout and functionality are relatively uniform, with predictable patterns such as threads, posts, and comments. This simplicity reduces ambiguity in user intent and allows language models to focus on natural language understanding without needing to interpret complex workflows. Moreover, forums are well-represented in the training data of large language models, giving them a familiarity with common interaction patterns and user intents, further enhancing performance.

In contrast, shopping tasks involve greater variability in user actions and workflows, requiring models to navigate dynamic content such as product listings, filters, dropdowns, and cart management. These environments often include domain-specific terms (e.g., SKUs, discount codes) and multi-step processes like checkout flows, making them inherently more challenging for models to predict and execute accurately. The need to account for various user goals, from browsing to comparing products, introduces an additional layer of complexity.

Shopping admin tasks introduce even more complexity by requiring fine-grained control over platform settings and configurations. These tasks may involve managing inventories, updating product information, handling user accounts, or configuring backend operations. The variety and specificity of these actions demand a deep understanding of the

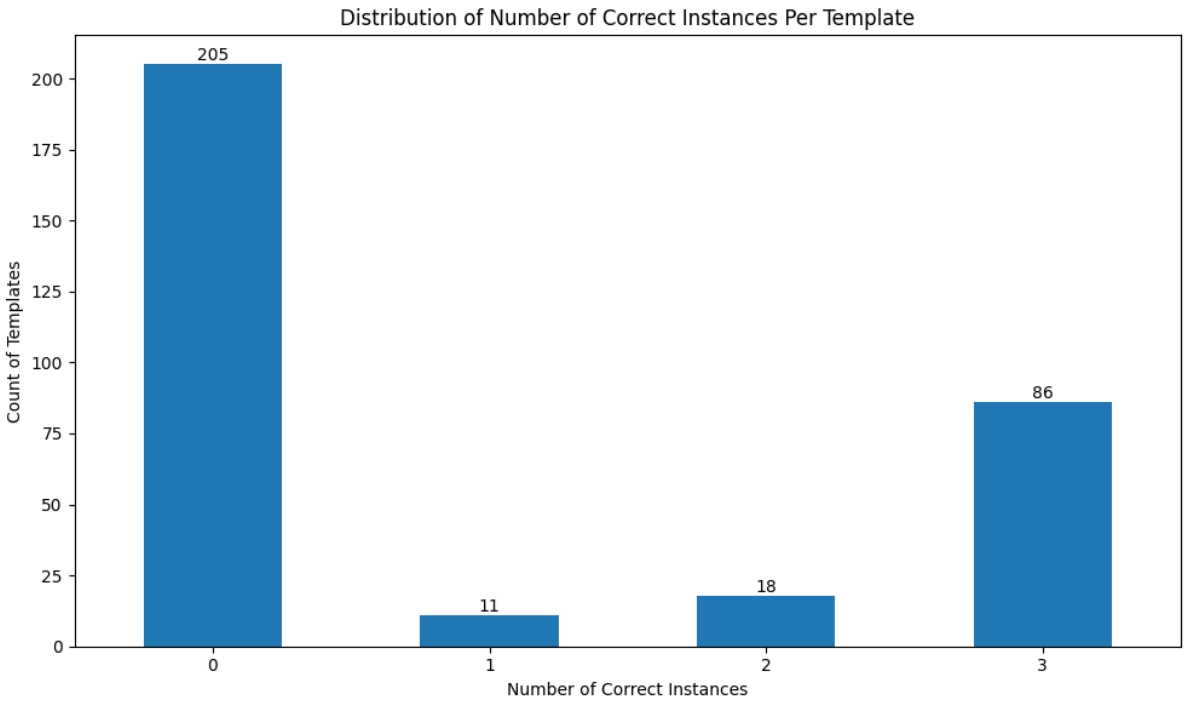

Figure 2: Distribution of number of correct instances per template

platform's infrastructure and workflows, increasing the likelihood of errors. Additionally, these tasks often require interpreting abstract or administrative concepts that may not align well with the model's training.

GitLab performed the worst due to its intricate permission structures and reliance on domain-specific knowledge. Tasks in GitLab frequently involve managing repositories, configuring CI/CD pipelines, or handling user roles, all of which require a precise understanding of technical workflows and permissions. The complexity is further compounded by features locked behind paywalls, which the model was unable to recognize or simulate. Unlike other environments, GitLab's specialized use case and technical nature demand expertise that is less represented in typical training datasets, making it more difficult for the model to navigate effectively.

Task Type In addition, we also analyze the results of OP and QA type tasks (Table 3). We find that QA tasks on average have better performance on both intent and query. We hypothesize this difference arises from the inherent nature of the tasks. QA tasks primarily involve querying the current state of the website, which tends to be straightforward and less error-prone. These tasks often require retrieving or matching information, aligning well with the capabilities of language models that excel at interpreting and generating text.

In contrast, OP tasks involve verifying whether a specific action successfully modified the state of the website. This process is inherently more complex, as it requires understanding both the intended action and its expected outcome. Additionally, OP tasks often involve multiple steps, dynamic interactions, or dependencies on the environment, increasing the likelihood of errors in intent identification or query formulation. These additional complexities contribute to the observed performance gap between QA and OP tasks.

6 Future Work

This paper aims to establish a foundation for exploring the complex challenge of automatic task generation for training web agents. This naturally paves the way for numerous exciting directions for future research.

By automatically generating tasks and evaluations, we open up the opportunity to fine-tuning agents on our tasks to assess and demonstrate measurable improvements in out-of-domain task performance. By iteratively training the agent on diverse tasks and incorporating feedback loops, we aim to refine its ability to generalize across domains. This process will help establish benchmarks for evaluat-

ing the agent's adaptability, robustness, and effectiveness in executing both predefined and emergent tasks.

In addition, from the poor quality of tasks `GPT-4o` has created, a reasonable next step would be to fine-tune the model itself to generate tasks more effectively using natural language inputs or even in scenarios with minimal or no input. This approach will focus on enabling the model to autonomously identify and articulate meaningful tasks, ensuring that the pipeline supports robust and contextually relevant task generation. By addressing potential shortcomings in task creation, we aim to improve the overall usability and versatility of the agent in real-world applications.

7 Conclusion

We present AutoGenTask, a comprehensive pipeline designed to automatically generate tasks for web agents. Through this pipeline, we have constructed a dataset of verified tasks, demonstrating its ability to produce tasks that are correct and challenging. To further evaluate the generated tasks, we leverage a novel SQL-based evaluation schema. By incorporating the website's database schema into the pipeline, we enable the automatic generation of SQL queries to evaluate the outcomes of each task. This approach ensures that tasks can be assessed deterministically. However, our analysis shows that the validity of the intent is highly dependent on the specific environment. Furthermore, automatic evaluation is a bottleneck even for valid intents. By introducing AutoGenTask, we aim to foster advancements in the development of robust and efficient web agents. The integration of the SQL schema and the pipeline's resource-efficient approach address critical challenges in creating large, high-quality datasets, paving the way for further research and innovation in this domain.

Acknowledgments

We would like to thank Jiachen Li and Antonis Antoniades for their valuable discussions and feedback during the development of this work. Additionally, we appreciate the support from UC Santa Barbara NLP Group for their API credits. (INSERT FUNDING HERE)

References

Mengkang Hu, Pu Zhao, Can Xu, Qingfeng Sun, Jianguang Lou, Qingwei Lin, Ping Luo, Saravan Rajmohan, and Dongmei Zhang. 2024. Agentgen: Enhancing planning abilities for large language model based agent via environment and task generation. *Preprint*, arXiv:2408.00764.

Fangyu Lei, Jixuan Chen, Yuxiao Ye, Ruisheng Cao, Dongchan Shin, Hongjin Su, Zhaoqing Suo, Hongcheng Gao, Wenjing Hu, Pengcheng Yin, Victor Zhong, Caiming Xiong, Ruoxi Sun, Qian Liu, Sida Wang, and Tao Yu. 2024. Spider 2.0: Evaluating language models on real-world enterprise text-to-sql workflows. *Preprint*, arXiv:2411.07763.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, Xuanhe Zhou, Ma Chenhao, Guoliang Li, Kevin Chang, Fei Huang, Reynold Cheng, and Yongbin Li. 2023a. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. In *Advances in Neural Information Processing Systems*, volume 36, pages 42330–42357. Curran Associates, Inc.

Yuanzhi Li, Sébastien Bubeck, Ronen Eldan, Allie Del Giorno, Suriya Gunasekar, and Yin Tat Lee. 2023b. Textbooks are all you need ii: phi-1.5 technical report. *Preprint*, arXiv:2309.05463.

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

Lin Long, Rui Wang, Ruixuan Xiao, Junbo Zhao, Xiao Ding, Gang Chen, and Haobo Wang. 2024. On LLMs-driven synthetic data generation, curation, and evaluation: A survey. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 11065–11082, Bangkok, Thailand. Association for Computational Linguistics.

Iman Mirzadeh, Keivan Alizadeh, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. 2024. Gsm-symbolic: Understanding the limitations of mathematical reasoning in large language models.

Jiayi Pan, Yichi Zhang, Nicholas Tomlin, Yifei Zhou, Sergey Levine, and Alane Suhr. 2024. Autonomous evaluation and refinement of digital agents. *Preprint*, arXiv:2404.06474.

Nicholas Pangakis, Samuel Wolken, and Neil Fasching. 2023. Automated annotation with generative ai requires validation. *Preprint*, arXiv:2306.00176.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. 2024. AppWorld: A controllable world of apps and people for benchmarking interactive coding agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 16022–16076, Bangkok, Thailand. Association for Computational Linguistics.

Shuohang Wang, Yang Liu, Yichong Xu, Chenguang Zhu, and Michael Zeng. 2021. Want to reduce labeling cost? GPT-3 can help. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 4195–4205, Punta Cana, Dominican Republic. Association for Computational Linguistics.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. 2024. Agent workflow memory. *Preprint*, arXiv:2409.07429.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed H. Chi, Quoc Le, and Denny Zhou. 2022. Chain of thought prompting elicits reasoning in large language models. *CoRR*, abs/2201.11903.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. 2023a. Webshop: Towards scalable real-world web interaction with grounded language agents. *Preprint*, arXiv:2207.01206.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023b. React: Synergizing reasoning and acting in language models. *Preprint*, arXiv:2210.03629.

Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. 2024. AgentTuning: Enabling generalized agent abilities for LLMs. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 3053–3077, Bangkok, Thailand. Association for Computational Linguistics.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. 2024. Webarena: A realistic web environment for building autonomous agents. *Preprint*, arXiv:2307.13854.

A Example Appendix

This is an appendix.