



GRAPH2EVAL: AUTOMATIC MULTIMODAL TASK GENERATION FOR AGENTS VIA KNOWLEDGE GRAPHS

Yurun Chen¹, Xavier Hu¹, Yuhan Liu², Ziqi Wang¹, Zeyi Liao³, Lin Chen⁴,

Feng Wei⁴, Yuxi Qian⁴, Bo Zheng⁴, Keting Yin¹, Shengyu Zhang¹

¹Zhejiang University

²Xiamen University

³The Ohio State University

⁴Ant Group

yurunchen.research@gmail.com

ABSTRACT

As multimodal LLM-driven agents continue to advance in autonomy and generalization, evaluation based on static datasets can no longer adequately assess their true capabilities in dynamic environments and diverse tasks. Existing LLM-based synthetic data methods are largely designed for LLM training and evaluation, and thus cannot be directly applied to agent tasks that require tool use and interactive capabilities. While recent studies have explored automatic agent task generation with LLMs, most efforts remain limited to text or image analysis, without systematically modeling multi-step interactions in web environments. To address these challenges, we propose GRAPH2EVAL, a knowledge graph-based framework that automatically generates both multimodal document comprehension tasks and web interaction tasks, enabling comprehensive evaluation of agents’ reasoning, collaboration, and interactive capabilities. In our approach, knowledge graphs constructed from multi-source external data serve as the task space, where we translate semantic relations into structured multimodal tasks using subgraph sampling, task templates, and meta-paths. A multi-stage filtering pipeline based on node reachability, LLM scoring, and similarity analysis is applied to guarantee the quality and executability of the generated tasks. Furthermore, GRAPH2EVAL supports end-to-end evaluation of multiple agent types (Single Agent, Multi-Agent, Web Agent) and measures reasoning, collaboration, and interaction capabilities. We instantiate the framework with GRAPH2EVAL-BENCH, a curated dataset of 1,319 tasks spanning document comprehension and web interaction scenarios. Experiments show that GRAPH2EVAL efficiently generates tasks that differentiate agent and model performance, revealing gaps in reasoning, collaboration, and web interaction across different settings and offering a new perspective for agent evaluation. Our code is available here: <https://github.com/YurunChen/Graph2Eval>.

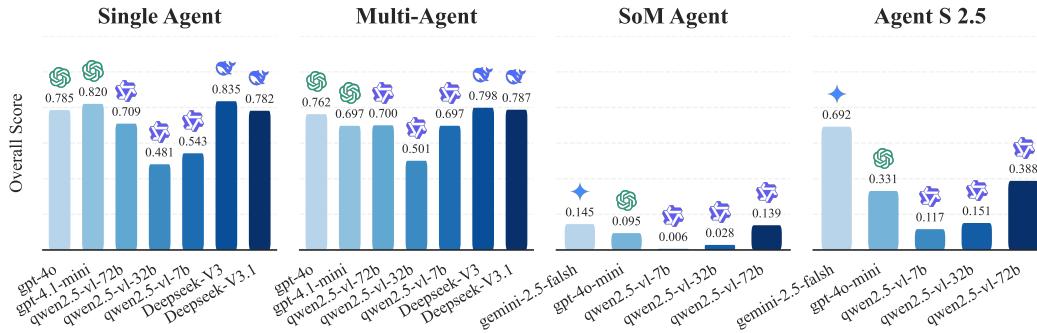


Figure 1: Performance of Agent–Model combinations evaluated on GRAPH2EVAL-BENCH.

1 INTRODUCTION

A static and overly narrow evaluation framework risks conflating superficial familiarity with genuine cognitive ability. Imagine evaluating a person’s problem-solving ability by repeatedly presenting the same fixed set of exercises, eventually leading them to memorize the answers and achieve seemingly perfect performance, while their true adaptability remains untested. The same concern arises for (M)LLM-based agents: While large-scale pretraining on foundation models and domain-specific finetuning have substantially advanced agent performance (Liu et al., 2025a;b; Choudhury & Sodhi, 2024; Chen et al., 2025a; Hu et al., 2025), existing static datasets fail to disentangle whether an agent’s task success reflects authentic capability or mere retrieval of memorized knowledge, thereby undermining the assessment of true competence. Therefore, success on static datasets does not imply that an agent can generalize or remain reliable in real-world scenarios.

Rapidly expanding and updating datasets can reduce the risk that agent evaluations are biased by reliance on internal training knowledge, yet current agent evaluation frameworks offer limited support for such capabilities. Current methods for constructing static datasets remain heavily dependent on manual annotation or the reuse of prior resources (Mialon et al., 2023; Deng et al., 2023; Liu et al., 2023; Yan et al., 2025; Chen et al., 2025b; Xiang et al., 2025), and even online environment datasets (Zhou et al., 2023; Xie et al., 2024a; Liao et al., 2025; Evtimov et al., 2025; Tur et al., 2025; Levy et al., 2025; Boisvert et al., 2025) require substantial human effort for task expansion, such as constructing controlled online environments and designing tasks from website content. These constraints limit task diversity and alignment with dynamic environments, motivating the development of automated agent task generation methods to alleviate such labor-intensive bottlenecks (Shi et al., 2025; Li et al., 2025; Zhang et al., 2025). However, these methods face two major limitations: **(I) Absence of explicit entity-relation modeling:** without structured representations of entities and their relations, generated tasks fail to capture scenarios requiring complex reasoning over interdependent concepts, restricting their utility for evaluating higher-order cognitive capabilities. **(II) Limited adaptation to dynamic environments:** most generated tasks are limited to tool calls or trivial reasoning, and rarely address multi-step, cross-modal, or dynamic interactions, neglecting scenarios that require interactive workflows and multi-hop dependencies in web environments.

To address these gaps, we propose **GRAPH2EVAL**, a knowledge graph-based framework for automatic task generation and evaluation. In this framework, the knowledge graph serves as both the *data repository* and the *latent space* for task generation, enabling the creation of document comprehension and web interaction tasks. GRAPH2EVAL consists of two main components: dataset generation and dataset evaluation. (1) During dataset generation, GRAPH2EVAL employs a unified graph abstraction to encode entities, relations, and interactions from textual and web data as nodes and edges representing both semantic and interactive elements. Building on this graph, GRAPH2EVAL defines task structures using task templates and meta-paths, which specify the types and order of nodes in a task and guide structured task generation. Subgraph sampling strategies are then applied to extract the required nodes and edges. Subsequently, LLMs integrate the sampled subgraph structures with contextual information via context engineering, generating diverse and well-formed task instances. (2) For evaluation, GRAPH2EVAL constructs multiple agent configurations, including single agent, multi-agent, and web-based agents, and combines rule-based scoring with LLM-as-a-judge to perform comprehensive evaluation.

We implemented a prototype of GRAPH2EVAL and constructed GRAPH2EVAL-BENCH, a dataset containing 1,319 diverse tasks, including 1,002 document comprehension tasks and 317 web interaction tasks. Based on the GRAPH2EVAL-integrated agents and multi-dimensional evaluation metrics, we conduct comprehensive experiments on GRAPH2EVAL-BENCH. The results show that Graph2Eval is highly efficient in task generation, with an average generation time of 34.87 seconds for document understanding tasks and 95.51 seconds for web interaction tasks. Furthermore, the generated dataset effectively distinguishes performance differences across various combinations of agents and LLMs of different scales (as shown in Figure 1).

Our contributions are summarized as follows:

- We propose a new perspective on task generation for agents, treating knowledge graphs constructed from multi-source data as a latent task space, and instantiate tasks through sampling.

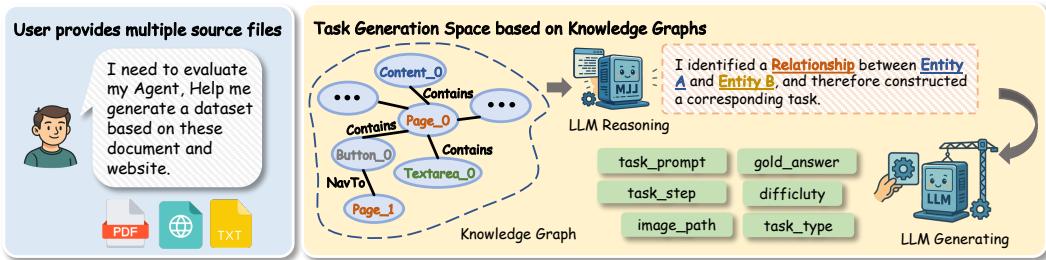


Figure 2: Overview of the dataset generated by GRAPH2EVAL.

- We introduce GRAPH2EVAL, a knowledge graph-based framework that exploits semantic relations for automatic task generation, providing a unified pipeline for rapid dataset creation and systematic evaluation of agent capabilities in a scalable and reproducible manner.
- To the best of our knowledge, GRAPH2EVAL is the first framework capable of automatically generating interactive tasks in web environment.
- We implement a full prototype of GRAPH2EVAL and construct GRAPH2EVAL-BENCH, a dataset comprising 1,319 agent tasks, on which we conduct extensive experiments demonstrating that the framework can efficiently generate diverse task datasets and effectively distinguish performance differences across different agent–model combinations.

2 BACKGROUND

In this section, we demonstrate the existing dataset’s deficiencies in customization and scalability.

Human-annotated data. A variety of agent datasets have been proposed, such as GAIA (Mialon et al., 2023), MiniWob (Shi et al., 2017), MiniWoB++ (Liu et al., 2018), Mind2Web (Deng et al., 2023), most of which are constructed through manual annotation. Beyond purely annotated corpora, several benchmarks are built upon realistic web or application environments—including OS World (Xie et al., 2024b), AndroidWorld (Rawles et al., 2025), RedTeamCuA (Liao et al., 2025), WASP Evtimov et al. (2025), SafeArena (Tur et al., 2025), STWebAgentBench (Levy et al., 2025), and DoomArena (Boisvert et al., 2025)—providing more interactive and dynamic settings. However, despite the richer environments, the task specifications in these datasets are still predominantly defined through human annotation, which limits scalability and diversity.

Synthetic Data. Existing synthetic data generation methods have primarily focused on training LLMs, while data generation for agent tasks remains relatively underexplored. Some approaches leverage LLMs to synthesize inputs in various ways, including seed-task-based generation (Wang et al., 2023; Xu et al., 2023; Li et al., 2024; Toshniwal et al., 2024), question rewriting (Yu et al., 2024), and self-iterative methods (Zelikman et al., 2022; Qiao et al., 2024). Additionally, other methods enhance instruction complexity through rule-based modifications (Xu et al., 2025) or extract question-answer pairs from web-pretrained corpora to construct training data (Yue et al., 2024). TaskCraft (Shi et al., 2025) automates the construction of tool-using agent tasks by first generating atomic tasks through LLM-based document traversal and then expanding the task space via task composition. Zhang et al. (2025) propose counterfactual replay and programmatic error injection to automatically build error-attribution datasets, which also indirectly results in the synthesis of agent trajectory data. However, although these methods have advanced the development of task synthesis to some extent, they remain limited in their ability to support multi-step, continuous task generation in web scenarios.

3 GRAPH2EVAL

The goal of GRAPH2EVAL is to establish an efficient task generation and evaluation framework for multimodal and multi-scenario settings, addressing the limitations of existing benchmarks that

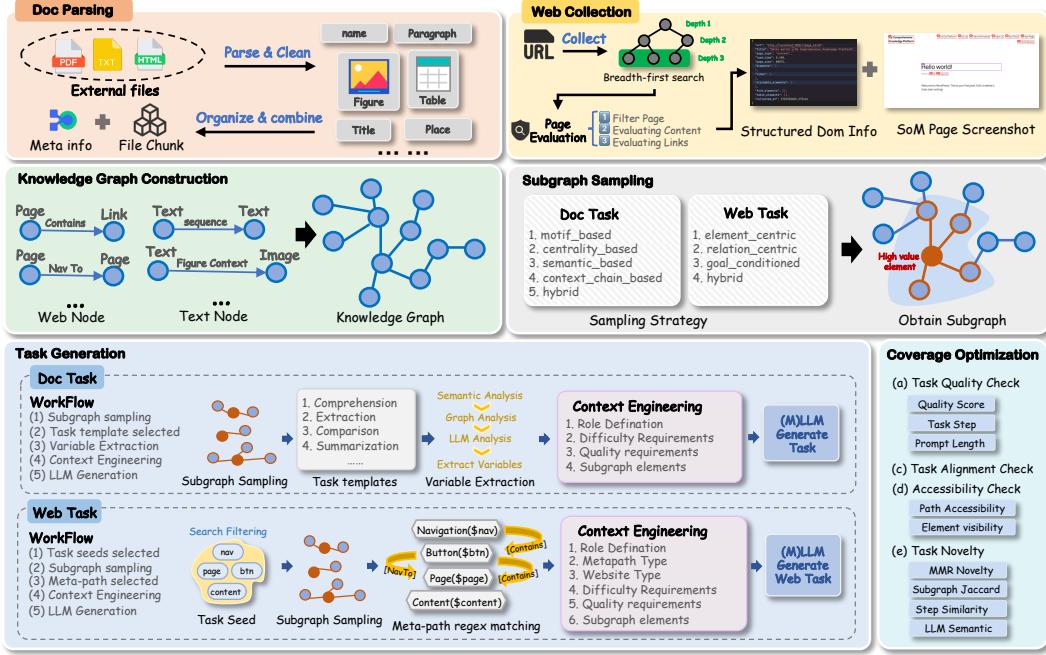


Figure 3: Workflow for dataset generation in GRAPH2EVAL: (1) Data Ingestion (**Top Left / Right**): parsing documents and crawling web pages to extract structured content. (2) Knowledge Graph Construction (**Middle Left**): building the graph by identifying nodes and edges that encode semantic, structural, and interactive relations. (3) Subgraph Sampling (**Middle Right**): applying scenario-specific sampling strategies for document and web tasks based on execution modes. (4) Task Generation (**Bottom Right**): instantiating and composing tasks from sampled subgraphs, producing diverse, executable task units. (5) Coverage Optimization (**Bottom Left**): evaluating and selecting generated tasks to ensure quality, diversity, and representativeness.

remain overly constrained to single-modality and static tasks. GRAPH2EVAL is characterized by three key strengths: **(1) Scalability & Customizability**, enabling low-cost, automatic task generation with controllable complexity and difficulty across diverse domains; **(2) Cross-Modal and Dynamic Task Support**, allowing both text-based reasoning tasks and web interaction tasks to be generated and executed in a semantically coherent, flexible, and progressive manner; and **(3) Multi-Dimensional Evaluation**, providing comprehensive agent evaluation across single and multi-agent settings. The workflow of dataset generation consists of five stages: *data parsing* → *knowledge graph construction* → *subgraph sampling* → *task generation* → *coverage optimization*, as illustrated in Figure 3.

3.1 DATA PARSING

During preprocessing, GRAPH2EVAL structures document content beyond plain text by preserving hierarchical semantics and layout elements such as paragraphs, tables, headings, and figure captions. This involves three key steps: **(1) Semantic Chunking**, segmenting the document into minimal semantic units mapped to knowledge graph nodes; **(2) Embedding Computation**, encoding each node with deep semantic embeddings to capture contextual dependencies; and **(3) Metadata Annotation**, enriching nodes with source and positional metadata (*e.g.*, file path, title, author). This *content* → *node* → *embedding + metadata* representation ensures semantic fidelity and cross-document consistency, enabling knowledge graph construction and task generation. For web data, pages are collected via automated URL crawling, extracting DOM structures and screenshots. To handle complex modern web designs, we integrate simulated human-like interactions to navigate pages. Collection quality is further improved through filtering strategies combining rule-based heuristics and LLM-based evaluation, effectively pruning low-quality links while increasing information density.

3.2 KNOWLEDGE GRAPH CONSTRUCTION

We construct a knowledge graph to transform unstructured and semi-structured content into a computable, reasoning-friendly semantic space. Formally, we define the graph as

$$G = (V, E, R),$$

where V is the set of nodes, E is the set of edges, and R is the set of relation types.

Node Extraction. Nodes are extracted by parsing a document or webpage D to identify elements such as paragraphs, headings, hyperlinks, forms, buttons, and table cells. Each element is mapped to a node:

$$V = \{v_i \mid v_i \in \text{Elements}(D), \text{type}(v_i) \in \text{NodeTypeSet}\},$$

where NodeTypeSet includes Paragraph, Heading, Hyperlink, Form, Button, Table, and other domain-specific elements. The contextual path $\text{Path}(v_i)$ is preserved to maintain the DOM hierarchy or document structure.

Node Representation. Each node v_i contains textual content c_i^T (e.g., text, captions, alt text) and visual content c_i^V (e.g., images, screenshots). Visual content is first converted to textual descriptions using a function ϕ_{visual} :

$$c_i^{T+V} = c_i^T \parallel \phi_{\text{visual}}(c_i^V),$$

where \parallel denotes text concatenation. The combined textual representation is then embedded into a vector space:

$$h_i = f_{\text{embed}}(c_i^{T+V}), \quad f_{\text{embed}} : \text{Text} \rightarrow \mathbb{R}^d,$$

where d is the embedding dimension (e.g., $d = 384$ for all-MiniLM-L6-v2). The resulting vector h_i is stored in a vector database D_{vec} for efficient semantic search and similarity matching.

Edge Construction. Relations between nodes are captured as a heterogeneous edge set:

$$E = E_{\text{text}} \cup E_{\text{web}}, \quad E \subseteq V \times V \times R,$$

where E_{text} encodes text-based relationships, including structural relations (e.g., sequence, contains), semantic associations (e.g., entity relations, semantic similarity), contextual relations (e.g., figure or table context), and reference relations (e.g., co-reference, cross-document links). E_{web} models web-specific interactions, such as navigation relations (e.g., page navigation, form submission), interaction relations (e.g., click triggers), and layout relations (e.g., visual layout or data flow).

3.3 SUBGRAPH SAMPLING

In the subgraph sampling stage, given a task objective g , GRAPH2EVAL extracts a local subgraph $G_g = (V_g, E_g) \subseteq G$ by selecting relevant nodes and their interconnections.

Notation. Let the knowledge graph be $G = (V, E)$, where V is the set of nodes and E the set of edges. The task objective is g . Each node v_i is evaluated for relevance using a scoring function $\text{Relevance}(\cdot)$ (in document mode, $\text{Relevance}(v_i, g) = \cos(h_i, h_g)$). The structural alignment is captured by $\text{StructMatch}(v_i, g)$. In the web mode, $S_{\text{seed}}(g)$ denotes task-specific seed nodes, and $\text{Neighbor}(v_i, k)$ returns the k -hop neighborhood of node v_i . Node sets are restricted by NodeTypeSet (document) or WebNodeSet (web).

Scenario-specific Sampling. The subgraph sampling follows different strategies depending on the scenario: **(1) Document comprehension:** Nodes include *DocumentElementNode* (paragraph, heading, table, image), *EntityNode* (person, location, organization), and *SemanticChunkNode*, capturing both semantic content and structural roles. Sampling prioritizes semantic relevance (via embeddings) and structural coherence (via *StructMatch*). Only nodes of the relevant types are included. **(2) Web interaction:** Sampling follows a seed-driven strategy. First, task-specific seed nodes $S_{\text{seed}}(g)$ (buttons, forms, navigation links) are identified. Second, the k -hop neighbors of each seed node are collected, including valid *WebPageNode* and *WebElementNode* entities. This ensures that the local interaction context is captured around the seeds.

Algorithm 1: Workflow of Subgraph Sampling

Input: Knowledge graph $G = (V, E)$, task objective g , mode $m \in \{\text{document, web}\}$, threshold τ (document), neighborhood k (web), seed nodes $S_{\text{seed}}(g)$ (web)

Output: Sampled subgraph $G_g = (V_g, E_g)$

```

1  $V_g \leftarrow \emptyset, E_g \leftarrow \emptyset$ ▷ Initialize node and edge sets
2 foreach  $v_i \in V$  do
3   /* Evaluate node  $v_i$  based on current mode (document or web) */
4   if  $m = \text{document}$  then
5      $h_i \leftarrow \text{EMBEDDING}(v_i)$ ▷ Compute node embedding
6     if  $\cos(h_i, h_g) > \tau$  or  $\text{StructMatch}(v_i, g) = 1$  then
7       if  $v_i \in \text{NodeTypeSet}$  then
8         |  $V_g \leftarrow V_g \cup \{v_i\}$ ▷ Include relevant node in subgraph
9   if  $m = \text{web}$  then
10    if  $v_i \in S_{\text{seed}}(g)$  then
11       $V_g \leftarrow V_g \cup \{v_i\}$ ▷ Add task-specific seed node
12       $N_i \leftarrow \text{NEIGHBOR}(v_i, k)$ ▷ Retrieve k-hop neighbors
13      foreach  $v_j \in N_i$  do
14        if  $v_j \in \text{WebNodeSet}$  then
15          |  $V_g \leftarrow V_g \cup \{v_j\}$ ▷ Add valid neighbor node
16  $E_g \leftarrow \{(v_i, v_j) \in E \mid v_i, v_j \in V_g\}$ ▷ Add edges connecting selected nodes
  /* Final subgraph  $G_g$  contains selected nodes and their interconnections */
17 return  $G_g = (V_g, E_g)$ 

```

3.4 TASK GENERATION

In GRAPH2EVAL, we use subgraphs from knowledge graphs to generate tasks. Each subgraph is transformed into an executable and evaluable task, yielding two task types: document comprehension and web interaction.

Document Comprehension. For document comprehension tasks, the generation pipeline of GRAPH2EVAL consists of four stages: **(1) Task Templates:** We maintain a library of *task templates* that cover fundamental categories such as question answering, comparison, analysis, and reasoning (see Appendix C for details). **(2) Subgraph Sampling:** The system performs *subgraph sampling* from the knowledge graph to select subgraphs that satisfy the constraints imposed by the task template. The sampled subgraph must meet template-specified requirements, including mandatory node and edge types, node count ranges, and maximum hop distance. By adjusting subgraph size, edge types, and sampling strategies, the framework can flexibly control task complexity and reasoning depth. **(3) Variable Extraction:** From the sampled subgraph, the system extracts template variables—such as node contents, edge relations, contextual information, and metadata—that serve as necessary inputs for task instantiation. **(4) Task Generation:** Given the subgraph, GRAPH2EVAL combines the structural content with the template-defined context, and further leverages LLMs to generate concrete task instances.

Web Interaction. For web interaction tasks, we propose a *Seed-Driven Subgraph Sampling Strategy*, which consists of four stages: **(1) Task Seed Identification:** GRAPH2EVAL first parses the page to identify key operational nodes (*e.g.*, buttons, input boxes, forms, navigation links) as “task seeds,” thereby anchoring task execution to actual page functionality. **(2) Subgraph Sampling:** Based on these seeds, relevant contextual nodes and interaction edges are sampled from the knowledge graph to construct a subgraph. **(3) Meta-path Matching:** Meta-path patterns are then applied to match and extend the subgraph, producing concrete task chains. Details of the meta-path design and implementation are provided in the Appendix D. **(4) Dynamic Task Generation:** Once the subgraph and task chain are obtained, LLMs generate concrete task instances by combining the subgraph structure, meta-paths, and page context information (*e.g.*, screenshots and element lists processed by the Set of Mark). For example, if only a search box, submit button, and result items are detected, the task chain becomes *Search + Detail*; if a filter is also present, the chain becomes *Search + Filter + Detail*. The *seed → subgraph sampling → meta-path matching* pipeline drives task generation with contextual relevance, enables multi-hop and branching flexibility through di-

verse sampling strategies, and offers controllability via seed selection and meta-path design. This compositional mechanism avoids rigid *all-or-nothing* constraints.

Highlights

The divergence between doc tasks and web tasks stems from their execution paradigms: doc tasks typically require an Agent to perform a limited number of API calls within a few dialogue turns, whereas web tasks inherently involve sequential, multi-step interactions within dynamic web environments, thereby necessitating distinct generation strategies.

3.5 COVERAGE OPTIMIZATION

We propose a multi-stage optimization framework to ensure the quality, diversity, and representativeness of generated tasks. For web interaction tasks, candidate tasks are first filtered using LLM- or rule-based quality scores, and coverage is quantified across node type, edge type, pattern, page-level, website type, and difficulty dimensions. Novelty is measured via multi-level similarity, and tasks are iteratively selected using an MMR-based strategy. For document comprehension tasks, coverage emphasizes semantic diversity across task type, difficulty, template, variable, and content length, with novelty assessed via LLM-based semantic similarity and selection also guided by MMR.

The final task sets balance quality, coverage, and diversity, as illustrated in Figure 4.

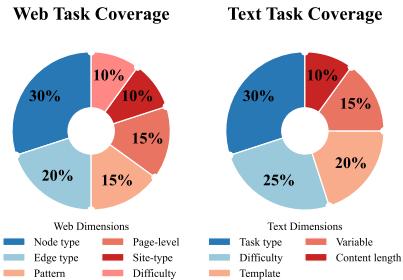


Figure 4: Coverage proportions of Web and Doc task dimensions used in the task optimization.

4 EXPERIMENT VERIFICATIONS

In this section, we present the dataset **GRAPH2EVAL-BENCH**, demonstrating that **GRAPH2EVAL** efficiently scales task generation while effectively differentiating agent and model capabilities.

4.1 IMPLEMENTATION DETAILS

Agents. We evaluate **GRAPH2EVAL** using multiple agent types, including Single Agent, Multi-Agent, and Web Agents, specifically the Set-of-Marks (SoM) Agent (Yang et al., 2023) and Agent S 2.5 (Agashe et al., 2025). Both Single- and Multi-Agent systems employ Retrieval-Augmented Generation (RAG) and primarily focus on document comprehension tasks, whereas the Web-Agent is designed for multi-step web interactions. The Multi-Agent architecture consists of a planner, retriever, reasoner, verifier, and summarizer. Notably, the SoM Agent leverages *SoM-annotated images* as input, providing richer multimodal context. In contrast, Agent S 2.5 integrates *task-aligned reflection* and *multidimensional memory management* to enhance reasoning performance. Detailed structure for all agents are provided in the Appendix F.

Baselines & Params. The models used for task generation and optimization are primarily based on GPT-4o (Hurst et al., 2024). We evaluate multiple model families, including: the GPT series, such as GPT-4o and GPT-4.1-mini (Hurst et al., 2024); the Deepseek series, including Deepseek-V3 and Deepseek-V3.1 (Guo et al., 2025); the Qwen series, including Qwen2.5-VL-7B, Qwen2.5-VL-32B, and Qwen2.5-VL-72B (Bai et al., 2023); and Gemini-2.5-flash (DeepMind, 2025). For all models, the temperature is set to 0.1, and the vector representation used for graph construction is based on all-MiniLM-L6-v2.

Metrics. We evaluate document comprehension tasks using three metrics: (1) *F1*, (2) *ROUGE-L*, and (3) *LLM-as-a-Judge* (abbreviated as *LLM Judge*), capturing performance at both the rule-based (exact match) and semantic understanding levels. For web-based tasks, we measure success using

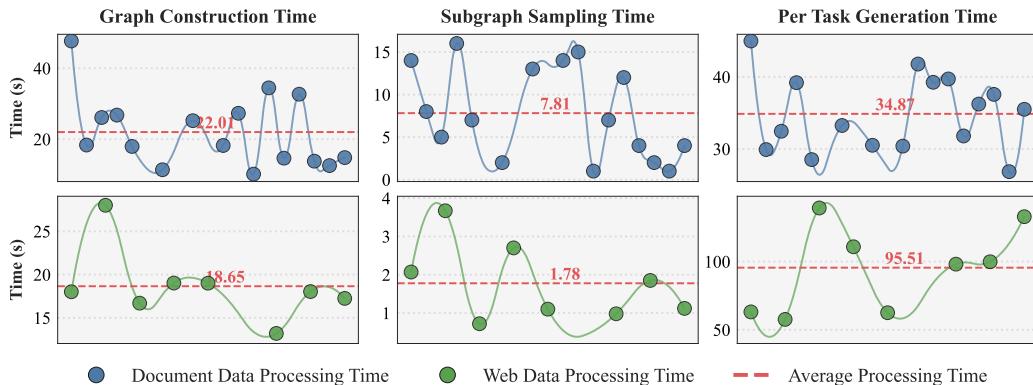


Figure 5: Comparison of Processing Times Across Documents and Websites.

(1) *Success Rate (SR)*, computed as the proportion of tasks determined to be successfully completed by an LLM evaluator. Detailed formulas are provided in the Appendix A.

4.2 ANALYSIS OF DATASET CONSTRUCTION

For document-based tasks, we collected existing high-quality papers as the data source. For web-based tasks, we gathered data from various websites, including screenshots and DOM structures. Using GRAPH2EVAL, these multi-source data were integrated to construct the GRAPH2EVAL-BENCH dataset. As shown in Table 1, the dataset contains 1,319 automatically generated tasks, covering both document understanding and multi-step web interaction scenarios. Additionally, we measured the time cost of GRAPH2EVAL for graph construction, subgraph sampling, and per-task generation. As illustrated in Figure 5, the total time spent on these stages is significantly lower than that required for manually constructed tasks. The detailed dataset composition is provided in the Appendix E.

4.3 ANALYSIS OF DOCUMENT COMPREHENSION TASKS

Overall Performance. We evaluated the GRAPH2EVAL-BENCH under both single agent and multi-agent collaboration settings, with results in Table 2. We tested both multimodal models and text-only models (in text mode, Graph2Eval converts figure interpretation tasks into text-based reasoning tasks by extracting image metadata, including titles, captions, alt text, and OCR-extracted text). Overall, GPT-4○ achieves the highest *F1* and *ROUGE-L* scores, while its performance under LLM-as-a-Judge ranks second. Deepseek-V3, on the other hand, performs best in the LLM-based assessment. Notably, multi-agent collaboration does not lead to significant improvements, suggesting that the multi-agent design provides limited enhancement for RAG-based understanding.

Task Type Performance. Figures 6 and 7 present single agent evaluation results across different task categories. Deepseek-V3 and GPT-4○ consistently achieve top or near-top performance across all typies, and clear differences are observed among models of varying parameter scales. These findings indicate that the tasks generated by GRAPH2EVAL are sufficiently discriminative and challenging, effectively capturing differences in language understanding and reasoning capabilities across models.

Models	Single Agent				Multi-Agent			
	F1	Rouge-L	LLM Judge	Avg. Token	F1	Rouge-L	LLM Judge	Avg. Token
Multimodal Models								
gpt-4o	0.5766	0.4874	0.7854	1631.82	0.5916	0.4873	0.7623	3560.92
gpt-4.1-mini	0.4202	0.4202	<u>0.8202</u>	3593.13	0.3496	0.3068	0.6972	4679.13
qwen2.5-vl-72b	<u>0.5730</u>	<u>0.4837</u>	0.7094	2394.19	<u>0.5673</u>	<u>0.4711</u>	0.6999	3855.65
qwen2.5-vl-32b	0.3677	0.3300	0.4811	2275.01	0.4341	0.3813	0.5008	3779.32
qwen2.5-vl-7b	0.2093	0.1939	0.5427	2455.17	0.3496	0.2548	0.6973	3732.27
Text Models								
Deepseek-v3	0.5376	0.4518	0.8351	<u>1710.65</u>	0.5497	0.4635	0.7984	<u>3462.88</u>
Deepseek-v3.1	0.5276	0.4329	0.7816	1777.98	0.5141	0.4253	<u>0.7875</u>	3435.12

Table 2: Performance comparison of models under single agent and multi-agent evaluation settings. Best and second-best values are **bolded** and underlined, respectively.

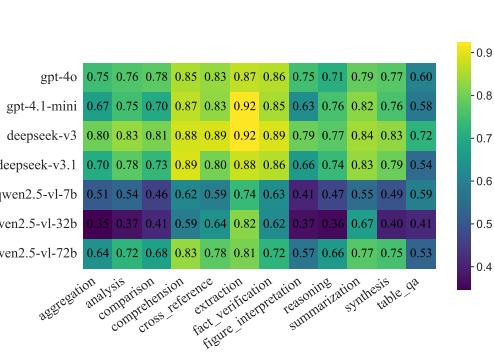


Figure 6: The heatmap shows the LLM evaluation quality scores of model responses across different task types (X-axis) and models (Y-axis).

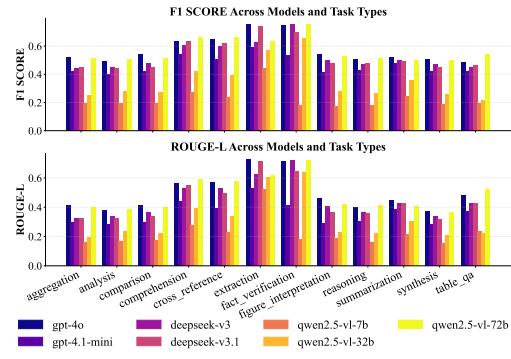


Figure 7: The grouped bar chart shows *F1* and *ROUGE-L* Score across different task types (X-axis) and models. Each model uses gradient-filled bars to reflect the score magnitude.

Findings

Multi-agent collaboration increases token usage without improving performance on document comprehension tasks, and can even slightly degrade results.

4.4 ANALYSIS OF WEB INTERACTION TASKS

We evaluated the web interaction tasks in the GRAPH2EVAL-BENCH using SoM Agent and Agent S 2.5, with the results presented in Table 3. Overall, Agent S 2.5 consistently outperforms SoM Agent across nearly all task types. On *gemini-2.5-flash*, both agents achieved their best overall performance, with SoM Agent reaching 14.51% and Agent S 2.5 achieving 69.20%, demonstrating a clear performance gap. The open-source model *qwen2.5-vl-72b* also achieved strong results, ranking as the second-best overall. By contrast, *gpt-4o-mini* showed competitive performance on specific tasks (*e.g.*, business navigation and modal interaction), but its overall effectiveness remained limited. The relatively low scores of *qwen2.5-vl-72b* and *qwen2.5-vl-32b* further indicate that the benchmark effectively distinguishes performance differences across models of varying scales when paired with agents. We provide a detailed case study in the Appendix G.

Models	Basic Nav.	Toast	Cont.	Search	Modal	Buss. Navi.	Button	Overall
SoM Agent								
gemini-2.5-flash	0.1778	0.0000	0.2000	0.1134	0.0000	0.2500	0.1875	0.1451
gpt-4o-mini	0.0667	0.0000	0.0000	0.1237	0.0000	0.0750	0.0000	0.0946
qwen2.5-v1-7b	0.0000	0.0000	0.0000	0.0103	0.0000	0.0000	0.0000	0.0063
qwen2.5-v1-32b	0.0222	1.0000	0.0667	0.0155	0.0000	0.0750	0.0000	0.0283
qwen2.5-v1-72b	0.1333	1.0000	0.2000	0.1031	0.1666	0.2750	0.1250	0.1388
Agent S 2.5								
gemini-2.5-flash	0.4889	1.0000	0.6667	0.6340	1.0000	0.5500	0.6250	0.6920
gpt-4o-mini	<u>0.3334</u>	0.0000	0.2000	0.3763	0.5000	0.1750	0.2500	0.3312
qwen2.5-v1-7b	0.1112	0.0000	0.0666	0.1392	0.1667	0.0750	0.0000	0.1167
qwen2.5-v1-32b	0.1778	0.0000	0.2000	0.1546	0.3333	0.0750	0.1250	0.1514
qwen2.5-v1-72b	<u>0.3334</u>	1.0000	<u>0.2667</u>	<u>0.4278</u>	<u>0.8333</u>	0.1500	<u>0.5625</u>	<u>0.3880</u>

Table 3: Performance comparison of models under *SoM Agent* and *Agent S 2.5* evaluation settings. Best and second-best values are **bolded** and underlined, respectively.

Findings

Task-aligned reflection (as a Test-Time Scaling strategy) and multidimensional memory management can improve the reasoning performance of (M)LLMs in web environments.

5 CONCLUSION AND FUTURE WORK

In this work, we presented GRAPH2EVAL, an automatic task generation framework that leverages knowledge graphs as an intermediate representation. By systematically modeling entities and their relationships within documents and web data, this framework integrates multi-source data into a unified task space, enabling scalable task creation to evaluate agent capabilities across diverse scenarios. Based on this framework, we constructed the GRAPH2EVAL-BENCH dataset. Experimental results demonstrate that GRAPH2EVAL effectively generates tasks spanning a wide range of scenarios, and comparative studies on models of varying scales and agent types confirm that these tasks reliably assess document comprehension and web interaction abilities under different settings. Future work will pursue two directions: (1) incorporating formalized safety policies to generate testable safety tasks for evaluating agent robustness in complex, dynamic environments; and (2) exploiting the structural properties of the knowledge graph to implement error attribution, allowing detailed analysis of agents’ weaknesses in language understanding, reasoning, and task execution.

REFERENCES

- Saket Agashe, Kyle Wong, Vincent Tu, Jiachen Yang, Ang Li, and Xin Eric Wang. Agent s2: A compositional generalist-specialist framework for computer use agents, 2025. URL <https://arxiv.org/abs/2504.00906>.
- Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. Qwen technical report. *arXiv preprint arXiv:2309.16609*, 2023.
- Leo Boisvert, Mihir Bansal, Chandra Kiran Reddy Evuru, Gabriel Huang, Abhay Puri, Avinandan Bose, Maryam Fazel, Quentin Cappart, Jason Stanley, Alexandre Lacoste, Alexandre Drouin, and Krishnamurthy Dvijotham. Doomarena: A framework for testing ai agents against evolving security threats, 2025. URL <https://arxiv.org/abs/2504.14064>.
- Yurun Chen, Xavier Hu, Yuhan Liu, Keting Yin, Juncheng Li, Zhuosheng Zhang, and Shengyu Zhang. Harmonyguard: Toward safety and utility in web agents via adaptive policy enhancement and dual-objective optimization, 2025a. URL <https://arxiv.org/abs/2508.04010>.

Zhaorun Chen, Mintong Kang, and Bo Li. Shieldagent: Shielding agents via verifiable safety policy reasoning, 2025b. URL <https://arxiv.org/abs/2503.22738>.

Sanjiban Choudhury and Paloma Sodhi. Better than your teacher: Llm agents that learn from privileged ai feedback, 2024. URL <https://arxiv.org/abs/2410.05434>.

DeepMind. Gemini 2.5 flash, 2025. URL <https://deepmind.google/models/gemini/flash/>. Accessed: 2025-09-27.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2web: Towards a generalist agent for the web, 2023. URL <https://arxiv.org/abs/2306.06070>.

Ivan Evtimov, Arman Zharmagambetov, Aaron Grattafiori, Chuan Guo, and Kamalika Chaudhuri. Wasp: Benchmarking web agent security against prompt injection attacks, 2025. URL <https://arxiv.org/abs/2504.18575>.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Xueyu Hu, Tao Xiong, Biao Yi, Zishu Wei, Ruixuan Xiao, Yurun Chen, Jiasheng Ye, Meiling Tao, Xiangxin Zhou, Ziyu Zhao, et al. Os agents: A survey on mllm-based agents for general computing devices use. *arXiv preprint arXiv:2508.04482*, 2025.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Os-trow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

Ido Levy, Ben Wiesel, Sami Marreed, Alon Oved, Avi Yaeli, and Segev Shlomov. St-webagentbench: A benchmark for evaluating safety and trustworthiness in web agents, 2025. URL <https://arxiv.org/abs/2410.06703>.

Jia Li, Edward Beeching, Lewis Tunstall, Ben Lipkin, Roman Soletskyi, Shengyi Huang, Kashif Rasul, Longhui Yu, Albert Q Jiang, Ziju Shen, et al. Numinamath: The largest public dataset in ai4maths with 860k pairs of competition math problems and solutions. *Hugging Face repository*, 13(9):9, 2024.

Junkai Li, Yunghwei Lai, Weitao Li, Jingyi Ren, Meng Zhang, Xinhui Kang, Siyu Wang, Peng Li, Ya-Qin Zhang, Weizhi Ma, and Yang Liu. Agent hospital: A simulacrum of hospital with evolvable medical agents, 2025. URL <https://arxiv.org/abs/2405.02957>.

Zeyi Liao, Jaylen Jones, Linxi Jiang, Eric Fosler-Lussier, Yu Su, Zhiqiang Lin, and Huan Sun. Redteamcua: Realistic adversarial testing of computer-use agents in hybrid web-os environments, 2025. URL <https://arxiv.org/abs/2505.21936>.

Evan Zheran Liu, Kelvin Guu, Panupong Pasupat, Tianlin Shi, and Percy Liang. Reinforcement learning on web interfaces using workflow-guided exploration, 2018. URL <https://arxiv.org/abs/1802.08802>.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. Agentbench: Evaluating llms as agents. *arXiv preprint arXiv:2308.03688*, 2023.

Yuhang Liu, Pengxiang Li, Zishu Wei, Congkai Xie, Xueyu Hu, Xinchen Xu, Shengyu Zhang, Xiaotian Han, Hongxia Yang, and Fei Wu. Infiguiagent: A multimodal generalist gui agent with native reasoning and reflection, 2025a. URL <https://arxiv.org/abs/2501.04575>.

Yuhang Liu, Pengxiang Li, Congkai Xie, Xavier Hu, Xiaotian Han, Shengyu Zhang, Hongxia Yang, and Fei Wu. Infigui-r1: Advancing multimodal gui agents from reactive actors to deliberative reasoners, 2025b. URL <https://arxiv.org/abs/2504.14239>.

Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. Gaia: a benchmark for general ai assistants, 2023. URL <https://arxiv.org/abs/2311.12983>.

Shuofei Qiao, Ningyu Zhang, Runnan Fang, Yujie Luo, Wangchunshu Zhou, Yuchen Eleanor Jiang, Chengfei Lv, and Huajun Chen. Autoact: Automatic agent learning from scratch for qa via self-planning, 2024. URL <https://arxiv.org/abs/2401.05268>.

Christopher Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William Bishop, Wei Li, Folawiyo Campbell-Ajala, Daniel Toyama, Robert Berry, Divya Tyamagundlu, Timothy Lillicrap, and Oriana Riva. Androidworld: A dynamic benchmarking environment for autonomous agents, 2025. URL <https://arxiv.org/abs/2405.14573>.

Dingfeng Shi, Jingyi Cao, Qianben Chen, Weichen Sun, Weizhen Li, Hongxuan Lu, Fangchen Dong, Tianrui Qin, King Zhu, Minghao Liu, Jian Yang, Ge Zhang, Jiaheng Liu, Changwang Zhang, Jun Wang, Yuchen Eleanor Jiang, and Wangchunshu Zhou. Taskcraft: Automated generation of agentic tasks, 2025. URL <https://arxiv.org/abs/2506.10055>.

Tianlin Shi, Andrej Karpathy, Linxi Fan, Jonathan Hernandez, and Percy Liang. World of bits: An open-domain platform for web-based agents. In Doina Precup and Yee Whye Teh (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 3135–3144. PMLR, 06–11 Aug 2017. URL <https://proceedings.mlr.press/v70/shi17a.html>.

Shubham Toshniwal, Wei Du, Ivan Moshkov, Branislav Kisacanin, Alexan Ayrapetyan, and Igor Gitman. Openmathinstruct-2: Accelerating ai for math with massive open-source instruction data, 2024. URL <https://arxiv.org/abs/2410.01560>.

Ada Defne Tur, Nicholas Meade, Xing Han Lù, Alejandra Zambrano, Arkil Patel, Esin Durmus, Spandana Gella, Karolina Stańczak, and Siva Reddy. Safearena: Evaluating the safety of autonomous web agents, 2025. URL <https://arxiv.org/abs/2503.04957>.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-instruct: Aligning language models with self-generated instructions, 2023. URL <https://arxiv.org/abs/2212.10560>.

Zhen Xiang, Linzhi Zheng, Yanjie Li, Junyuan Hong, Qinbin Li, Han Xie, Jiawei Zhang, Zidi Xiong, Chulin Xie, Carl Yang, Dawn Song, and Bo Li. Guardagent: Safeguard llm agents by a guard agent via knowledge-enabled reasoning, 2025. URL <https://arxiv.org/abs/2406.09187>.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments, 2024a.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments, 2024b. URL <https://arxiv.org/abs/2404.07972>.

Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, Qingwei Lin, and Daxin Jiang. Wizardlm: Empowering large pre-trained language models to follow complex instructions, 2025. URL <https://arxiv.org/abs/2304.12244>.

Canwen Xu, Daya Guo, Nan Duan, and Julian McAuley. Baize: An open-source chat model with parameter-efficient tuning on self-chat data, 2023. URL <https://arxiv.org/abs/2304.01196>.

Yunhe Yan, Shihe Wang, Jiajun Du, Yexuan Yang, Yuxuan Shan, Qichen Qiu, Xianqing Jia, Xinge Wang, Xin Yuan, Xu Han, Mao Qin, Yinxiao Chen, Chen Peng, Shangguang Wang, and Mengwei Xu. Mcpworld: A unified benchmarking testbed for api, gui, and hybrid computer use agents, 2025. URL <https://arxiv.org/abs/2506.07672>.

Jianwei Yang, Hao Zhang, Feng Li, Xueyan Zou, Chunyuan Li, and Jianfeng Gao. Set-of-mark prompting unleashes extraordinary visual grounding in gpt-4v, 2023. URL <https://arxiv.org/abs/2310.11441>.

Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T. Kwok, Zhen-guo Li, Adrian Weller, and Weiyang Liu. Metamath: Bootstrap your own mathematical questions for large language models, 2024. URL <https://arxiv.org/abs/2309.12284>.

Xiang Yue, Tuney Zheng, Ge Zhang, and Wenhu Chen. Mammoth2: Scaling instructions from the web, 2024. URL <https://arxiv.org/abs/2405.03548>.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. Star: Bootstrapping reasoning with reasoning, 2022. URL <https://arxiv.org/abs/2203.14465>.

Guibin Zhang, Junhao Wang, Junjie Chen, Wangchunshu Zhou, Kun Wang, and Shuicheng Yan. Agentracer: Who is inducing failure in the llm agentic systems?, 2025. URL <https://arxiv.org/abs/2509.03312>.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Yonatan Bisk, Daniel Fried, Uri Alon, et al. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*, 2023. URL <https://webarena.dev>.

A METRICS FORMULA

A.1 DOCUMENT COMPREHENSION METRICS

F1 Score. For a predicted answer span P and a ground-truth span G , precision and recall are defined as

$$\text{Precision} = \frac{|P \cap G|}{|P|}, \quad \text{Recall} = \frac{|P \cap G|}{|G|}. \quad (1)$$

The F1 score is their harmonic mean:

$$F1 = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}. \quad (2)$$

ROUGE-L. ROUGE-L measures the longest common subsequence (LCS) between P and G :

$$\text{ROUGE-L} = \frac{(1 + \beta^2) \cdot R_{\text{LCS}} \cdot P_{\text{LCS}}}{R_{\text{LCS}} + \beta^2 \cdot P_{\text{LCS}}}, \quad (3)$$

where

$$R_{\text{LCS}} = \frac{\text{LCS}(P, G)}{|G|}, \quad P_{\text{LCS}} = \frac{\text{LCS}(P, G)}{|P|}.$$

Here, β is a weighting parameter that balances the importance of recall versus precision. Following common practice, we set $\beta = 1$ to weigh precision and recall equally.

LLM-as-a-Judge. LLM is prompted to evaluate the predicted answer P against the ground-truth G and task instruction along multiple dimensions, including quality, relevance, and completeness. The scores are normalized to $[0, 1]$, and can be aggregated into an overall similarity judgment, with higher values indicating stronger semantic alignment. The instruction used to guide the LLM in this evaluation are provided below.

LLM Instructions for Evaluating Document Comprehension Tasks

You are an expert evaluator assessing the quality of an AI-generated answer. Please evaluate the following:

TASK: {task.prompt}

GOLD STANDARD ANSWER: {gold_answer}

GENERATED ANSWER: {pred_answer}

Rate the generated answer on these 3 key dimensions (0.0 to 1.0):

1. ANSWER_QUALITY: Overall quality and accuracy of the answer compared to the gold standard

2. RELEVANCE: How well the answer addresses the specific task/question

3. COMPLETENESS: How complete and comprehensive the answer is

Provide your assessment in JSON format:

```
{
    "answer_quality": <score>,
    "relevance": <score>,
    "completeness": <score>
}
```

Be objective and focus on the most important aspects of answer quality.

A.2 WEB-BASED TASK METRICS

Success Rate (SR). We evaluate web interaction tasks using *Success Rate (SR)*, defined as the fraction of tasks judged successful by an LLM evaluator:

$$\text{SR} = \frac{N_{\text{success}}}{N_{\text{total}}}, \quad (4)$$

where N_{success} is the number of tasks determined to be successfully completed by LLMs, and N_{total} is the total number of tasks. The use of LLMs for evaluation is motivated by the fact that web interaction tasks are executed in live, dynamic online environments, where the complexity and variability of web pages render rule-based evaluation (*e.g.*, checking system states or explicit signals) unreliable. To address this challenge, we employ LLMs to determine task success by analyzing the sequence of executed actions, the final page state, and any encountered error messages. By leveraging the LLM’s ability to interpret online content and reason about task goals, this approach provides a consistent, scalable, and generalizable measure of task completion. The instructions used to guide the LLM in evaluating task success are provided below.

LLM Instructions for Evaluating Web Interaction Tasks

Task: {task.prompt if hasattr(task, 'prompt') else f'Complete task: {task.task_id}'}

Execution Summary:

- Actions executed: {len(trajectory.actions_executed)}
- Success: {trajectory.success}
- Error message: {trajectory.error_message or 'None'}

Current page URL: {page_info.get('url', 'Unknown')}

Current page title: {page_info.get('title', 'Unknown')}

Actions executed:

{chr(10).join([f'- {action.get('action', 'unknown')}]' for action in trajectory.actions_executed])}

Please evaluate if the task has been completed successfully by analyzing the current page state. Consider: 1. Whether all required actions were performed 2. Whether the final state matches the task requirements 3. Whether any errors occurred that prevent completion 4. Whether the current page content indicates task completion

Respond with valid JSON format (no markdown, no code blocks):

```
{
    "task_completed": true,
    "confidence": 0.8,
    "reasoning": "explanation of your evaluation",
    "missing_actions": ["list of any missing actions"],
    "final_state_analysis": "description of current page state"
}
```

B TASK TEMPLATE

Task templates serve as the core framework in GRAPH2EVAL for automatically generating evaluation tasks. They are designed as structured data classes and include fundamental information such as a template identifier, name, description, task type, difficulty level, and required capabilities. Each template also provides Jinja2-formatted prompt and reference answer templates to dynamically generate task content. Each template defines strict graph-structure requirements, including mandatory node types, edge types, minimum and maximum numbers of nodes, and maximum hop distances, ensuring that tasks are generated based on specific subgraph structures in the knowledge graph. The system supports 12 distinct text-based task types, ranging from basic information extraction, comprehension, and summarization to more complex tasks such as multi-hop reasoning, comparative analysis, fact verification, image interpretation, and cross-referencing. Each task type is associated with a designated difficulty level (easy, medium, hard, expert). Templates further specify detailed evaluation criteria, including a list of evaluation metrics, requirements for exact matching, reference citations, reasoning paths, as well as version control and tagging capabilities. Through the template library manager, the system intelligently selects suitable templates based on the given graph structural features (node types, edge types, node counts) and employs the Jinja2 engine to render variables into concrete task prompts and reference answers. This enables a fully automated transformation from abstract templates to specific task instances.

C TASK TEMPLATES

Task templates constitute the core structural modules in GRAPH2EVAL for automatically generating evaluation tasks. Each template is designed as a structured data class and encapsulates fundamental information including a *template ID*, *name*, *description*, *task type* and *difficulty level*. In addition, templates provide Jinja2-formatted prompt and reference answer templates to dynamically generate task content.

Each template imposes strict **graph-structure requirements**, specifying mandatory node types, edge types, minimum and maximum node counts, and maximum hop distances, ensuring that tasks are instantiated from specific subgraph structures within the knowledge graph. Graph2Eval supports twelve distinct **text-based task types**, ranging from fundamental tasks such as information extraction, comprehension, and summarization, to more complex tasks including multi-hop reasoning, comparative analysis, fact verification, image interpretation, and cross-referencing. Each task type is associated with a designated **difficulty level** (Easy, Medium, Hard, Expert).

Templates also define detailed **evaluation criteria**, including evaluation metrics, requirements for exact matching, reference citations, reasoning paths, as well as version control and tagging mechanisms. The **template library manager** intelligently selects suitable templates based on the structural features of a given graph (node types, edge types, and node counts) and leverages the Jinja2 engine to render variables into concrete task prompts and reference answers. This facilitates fully automated instantiation from abstract templates to concrete evaluation tasks.

Comparison Tasks in Task Templates

Compare the following pieces of information:

- Item 1: {{ comparison_items[0].content }}
- Item 2: {{ comparison_items[1].content }}

Provide a detailed comparison and cite your sources.

{}{{ answer }}

D META-PATH

The meta-path serves as a core mechanism in GRAPH2EVAL for generating web-based tasks, enabling automatic transformation from subgraphs of web pages into executable tasks. The system employs a hierarchical design comprising two key components: *MetapathPattern* and *MetapathInstance*. The pattern defines the structural template of a task, e.g.,

SearchBox(\$search) – [Fills] –> BusinessData(\$query) – [Controls] –> Button(\$submit)

while the instance represents the matching result of a pattern on a concrete subgraph. The system integrates a *Graph Regex Engine*, supporting regex-like graph pattern syntax, including node type matching, edge type matching, quantifiers (e.g., ?, *, +, {n,m}), and alternative constructs (e.g., `Toast` | `Modal`), thereby enabling flexible matching and dynamic composition.

GRAPH2EVAL follows a three-tier priority strategy: (1) **Business Data Patterns**: require subgraphs to contain real business data nodes (e.g., user data, product data, order data), enabling generation of high-value business-related tasks; (2) **General Interaction Patterns**: applicable to pages with common web elements such as search boxes, buttons, or navigation elements; (3) **Basic Interaction Patterns**: serve as fallback mechanisms to ensure that even the simplest page structures can generate basic interactive tasks. Variables in patterns (e.g., \$search, \$button) are bound to specific node IDs in the subgraph via a slot-binding mechanism. Based on the matched pattern type, the system generates corresponding task steps (e.g., click, input, navigate), ultimately producing a fully executable web task instance. This framework achieves automated transformation from abstract graph structures to concrete, actionable tasks.

E GRAPH2EVAL-BENCH OVERVIEW

Table 8 and Figure 9 summarize the data sources of the GRAPH2EVAL-BENCH, the number of tasks generated from each source, and the distribution across task types. In total, the GRAPH2EVAL-BENCH comprises 1,319 tasks and consists of two primary components: document comprehension datasets and web interaction datasets. The document comprehension datasets cover a wide range of task types, including reasoning, analysis, and aggregation, and enable diverse evaluations across multiple modalities. To mitigate potential risks to live websites, the web interaction datasets focus on navigation and interaction tasks, spanning various domains such as digital libraries, weather services, and news portals.

Data Source	Tasks Num.
Document Comprehension Datasets	
Agent AI	64
AgentHarm	65
AI Agent under Threat	50
The Dawn of GUI Agent	68
Data Shapley	63
DeepSeek-R1	58
Speculative Decoding	70
GPT-4o System Card	72
learning_dynamic	67
lightRAG	68
Navigating the Risks	47
OpenAI o3 and o4-mini	62
OS Agents	64
Qwen-VL	58
RTBAS	61
TaskCraft	59
Web Interaction Datasets	
Mozilla Developer	28
GitHub	17
Project Gutenberg	15
Open Library	78
OpenWeather	16
Stack Overflow	29
The Guardian	87
WIRED	47

Figure 8: Construction of the GRAPH2EVAL-BENCH

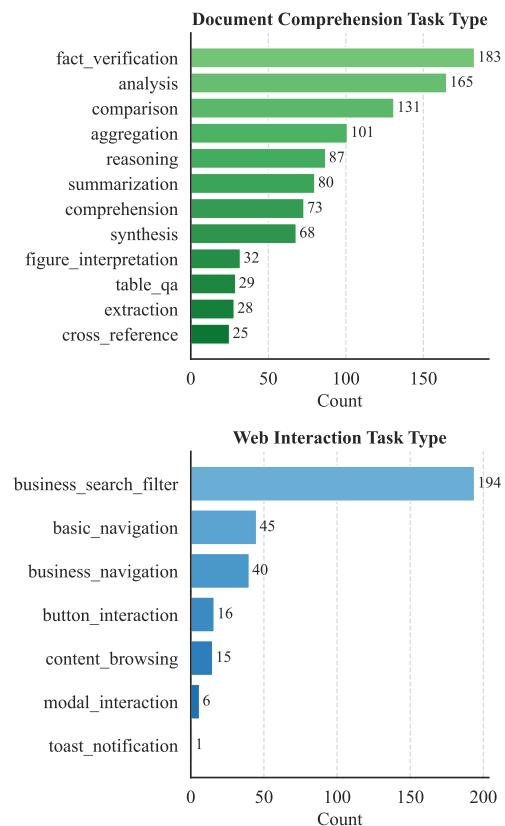


Figure 9: Case study of Agent S performing tasks on the Web dataset.

F AGENT ARCHITECTURE

In this section, we describe the types of agents supported in GRAPH2EVAL. Their capabilities are summarized in Table 4.

Single Agent. The Single Agent integrates a Retrieval-Augmented Generation (RAG) framework, following a four-step *retrieve-execute-evaluate-respond* workflow. It produces structured outputs that include references, reasoning paths, and confidence scores. A memory management module records task history and supports interactive dialogue, enabling more coherent and context-aware reasoning.

Multi-Agent. The Multi-Agent establishes a distributed collaborative reasoning architecture for complex task decomposition and coordination, which also integrates RAG capabilities. It defines

five core agent roles: *PLANNER* (task planning and decomposition), *RETRIEVER* (information retrieval), *REASONER* (logical inference), *VERIFIER* (result validation), and *SUMMARIZER* (information consolidation). Each agent maintains independent reasoning capabilities, recording reasoning steps via ReasoningStep structures that capture source/target nodes, edge relations, logic, and confidence. Agents communicate through a standardized messaging protocol, enabling dynamic task allocation and load balancing. This design allows the system to handle complex multi-hop reasoning tasks, achieving collaborative reasoning performance superior to single-agent setups.

SoM Agent. The SoM Agent integrates the SoM annotation to achieve precise element localization on web pages. Interactive elements are annotated with color-coded borders (*e.g.*, M1, M2, M3), where each mark uniquely represents a specific element type. To locate a target element, the system generates a screenshot containing all marks and leverages a dedicated SoM analysis prompt to guide the LLM in identifying the corresponding mark ID, establishing a complete mapping from textual description to visual mark to exact coordinates.

Agent S 2.5. Agent S 2.5 implements a reflective multi-modal reasoning system. Its four-layer architecture comprises LMMAgent (multi-modal language model agent), WebACI (browser interface), Worker (execution agent with procedural memory and reflection), and ProceduralMemory (structured task guidance). The core ability lies in the reflection mechanism, where an independent module analyzes execution trajectories, identifies issues, and provides improvement suggestions. Multi-modal input processing enables simultaneous analysis of screenshots and text, facilitating more accurate page understanding and element localization.

Capability	Single Agent	Multi-Agent	SoM Agent	Agent S 2.5
<i>Architecture & Knowledge</i>				
Knowledge Retrieval	✓	✓	✗	✗
Memory Management	✓	✓	△	✓
<i>Interaction & Web Automation</i>				
Web Automation	✗	✗	✓	✓
Visual Marking	✗	✗	✓	✓
Multimodal Processing	✓	✓	✓	✓
<i>Reasoning & Evaluation</i>				
Task Planning	△	✓	✓	✓
Error Handling	✓	✓	✓	✓
Reflection	✗	✗	✗	✓
<i>Execution & Performance</i>				
Concurrency	✗	△	✗	✗
Token Monitoring	✓	✓	✓	✓

Table 4: Capability Comparison of the Four Agents in the GRAPH2EVAL. ✓ indicates a fully supported capability, △ indicates partial support, and ✗ indicates that the capability is not supported.

G CASE STUDY

We illustrate in Figure 10 the overall workflow of Agent S in performing a web interaction task. The figure presents both the task specification and the agent settings. Given page screenshots and DOM representations, Agent S 2.5 leverages its memory and reflection mechanisms to reason about the current state, determine the next action, and execute it accordingly. The final outcome of the task is subsequently validated by another LLM, ensuring the reliability of the result. During this process, GRAPH2EVAL continuously monitors and records key performance indicators, including execution time and token consumption, which provide quantitative insights into the efficiency and cost of the agent. This case not only demonstrates how Agent S operates in a realistic web environment but also highlights the effectiveness of combining reasoning, memory, and external validation for reliable web-based task execution.

H EXPLORING SAFETY TASK GENERATION

We investigate the generation of safety-focused tasks with the aim of improving the evaluation of agents’ safety boundaries. Building on the overall Graph2Eval framework, we develop an initial pipeline for synthesizing safety tasks. For text-understanding benchmarks, we derive safety-oriented instances from existing tasks so as to preserve task naturalness while minimally perturbing inputs, thereby amplifying the ability to detect latent model risks.

Safety Task Generation for Document Comprehension. The generation pipeline is organized into three stages: **(1) Policy-document parsing and threat extraction:** Given policy or security documents as input, LLMs are used to automatically extract candidate threat types and convert them into structured representations (*e.g.*, threat category, examples, keywords, and severity) for downstream use. **(2) Threat-embedding strategy exploration:** We explore multiple embedding strategies—content injection, prompt manipulation, context switching, and indirect reference—to integrate threat information into original texts in natural, covert, or semantically implicit forms. This enables assessment of models’ ability to recognize and defend against explicit and implicit threats under different surface realizations. **(3) Safety-instance creation and preliminary quality control:** For each embedding strategy, we generate task instances and log metadata (task ID, difficulty level, prompt and reference answer, requisite reasoning traces, and evaluation criteria). We further perform an initial quality assessment of clarity, relevance, difficulty, and completeness, retaining only samples that meet preset thresholds to ensure test reliability.

Safety Task Generation for Web Interactions. For web interaction tasks, we primarily explore threat modeling in web environments. Because directly embedding threats into live online environments would create unacceptable real-world risks, we adopt a controlled sandbox approach: in isolated Docker environments or controlled browser sessions, we inject malicious environment artifacts via scripts. (*e.g.*, forged phishing elements, malicious forms, or suspicious redirects) to simulate online threat scenarios safely and reproducibly. We further augment existing web interaction tasks with safety nudges (*e.g.*, security warnings) and generate web-oriented safety tasks to evaluate models’ security judgement in web environment.



Figure 10: Case study of Agent S performing tasks on the Web dataset.