A. Task Content Constraints

For document understanding tasks, the content safety is relatively controllable due to the constraints of task type and information sources. In contrast, web interaction tasks require the system to autonomously collect information from web pages, so we have implemented a multi-stage filtering mechanism to ensure safety.

Data Parsing Stage. We utilize LLMs to evaluate both web links and page content. During this process, the LLM evaluator is explicitly instructed to avoid pages containing personal or private information (e.g., pages with email addresses, phone numbers, or physical addresses) and to prioritize high-quality pages that do not involve sensitive information. After the page data is collected, GRAPH2EVAL utilizes LLMs to evaluate links and pages for privacy and compliance. The evaluation dimensions include: **(i) privacy_safe**: no exposure of personal or private information; **(ii) copyright_safe**: no proprietary or subscription content; **(iii) content_safe**: no harmful or inappropriate content; **(iv) robots_compliant**: compliance with `robots.txt` rules. Any page that fails the **privacy_safe** check is automatically filtered out and does not proceed to subsequent graph construction or task generation processes.

Task Generation Stage. In the task generation stage, constraints are explicitly defined in the prompt templates for web tasks, prohibiting tasks involving payment or privacy data operations. These constraints are applied both in LLM prompts based on subgraph analysis and those based on meta-path analysis, ensuring that generated web tasks do not require agents to perform operations involving sensitive information. Additionally, the system employs a business data placeholder mechanism, using the placeholder `[BUSINESS_DATA]` in task descriptions and steps to replace real data values. This prevents the generated tasks from directly containing any actual business data that might leak sensitive information. Task examples clearly demonstrate the use of placeholders instead of real data.

B. 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).

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` }}

{{ `question` }}
Provide a detailed comparison and cite your sources.
{{ `answer` }}

C. 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.

D. Limitations and Discussion

Our approach has two main limitations. (1) It relies on accurate knowledge graph modeling. If the graph is incomplete or contains errors, the quality of the generated tasks may be adversely affected. (2) The method currently focuses on structured knowledge and predefined relationships. This may limit its ability to generate tasks involving unstructured or novel information outside the knowledge graph, potentially constraining the diversity and coverage of tasks. In future work, these limitations could be addressed by integrating more robust knowledge-graph construction techniques and incorporating methods that allow the model to leverage unstructured data, thereby improving task quality, diversity, and coverage.

E. Agent Architecture

In this section, we describe the types of agents supported in GRAPH2EVAL. Their capabilities are summarized in Table 7.

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 multimodal 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. Multimodal input processing enables simultaneous analysis of screenshots and text, facilitating more accurate page understanding and element localization.

F. Metrics Formula

F.1. Document Understanding Task Metrics

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

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

The F1 score is their harmonic mean:

$$\text{F1} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}. \tag{2}$$

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

$$\text{ROUGE-L} = \frac{(1+\beta^2) \cdot \text{R}_{\text{LCS}} \cdot \text{P}_{\text{LCS}}}{\text{R}_{\text{LCS}} + \beta^2 \cdot \text{P}_{\text{LCS}}}, \tag{3}$$

where

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

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

Table 7. Capability Comparison of the Four Agents in the GRAPH2EVAL Framework. ✓ indicates a fully supported capability, △ indicates partial support, and × indicates that the capability is not supported.

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 Understanding 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.

F.2. Web 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:

$$\text{SR} = \frac{N_{\text{success}}}{N_{\text{total}}}, \tag{4}$$

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.

F.3. LLM Evaluator Reliability

Using LLMs as evaluators to assess task success rate in dynamic Web environments offers high scalability but may raise reliability concerns. To validate the reliability of LLM-as-a-Judge on web interaction tasks, We randomly sampled 50 task trajectories completed by Agent S 2.5 from 317 web interaction tasks. Two human annotators independently judged whether each trajectory successfully completed the task ("success" or "failure"), with agreement indicating success. The results show that 88% of the judgments matched those of `GPT-4o`, demonstrating high consistency.

G. GRAPH2EVAL-BENCH Overview

Table 8 and Figure 4 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 understanding datasets and web interaction datasets. The document understanding 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.

H. Case Study

We illustrate in Figure 5 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.

I. 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 Understanding. 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 judgment in web environments.

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"
}
```

Data Source	Tasks Num.
Document Understanding 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

Table 8. Task distribution across different data sources in GRAPH2EVAL-BENCH.

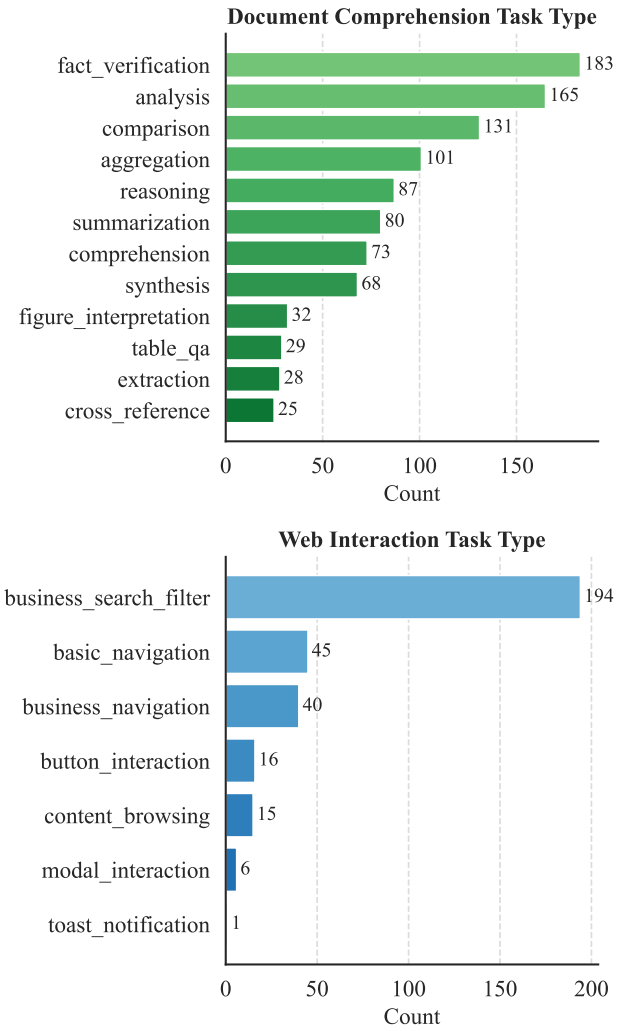

Figure 4. The number of tasks for each task type in GRAPH2EVAL-BENCH.

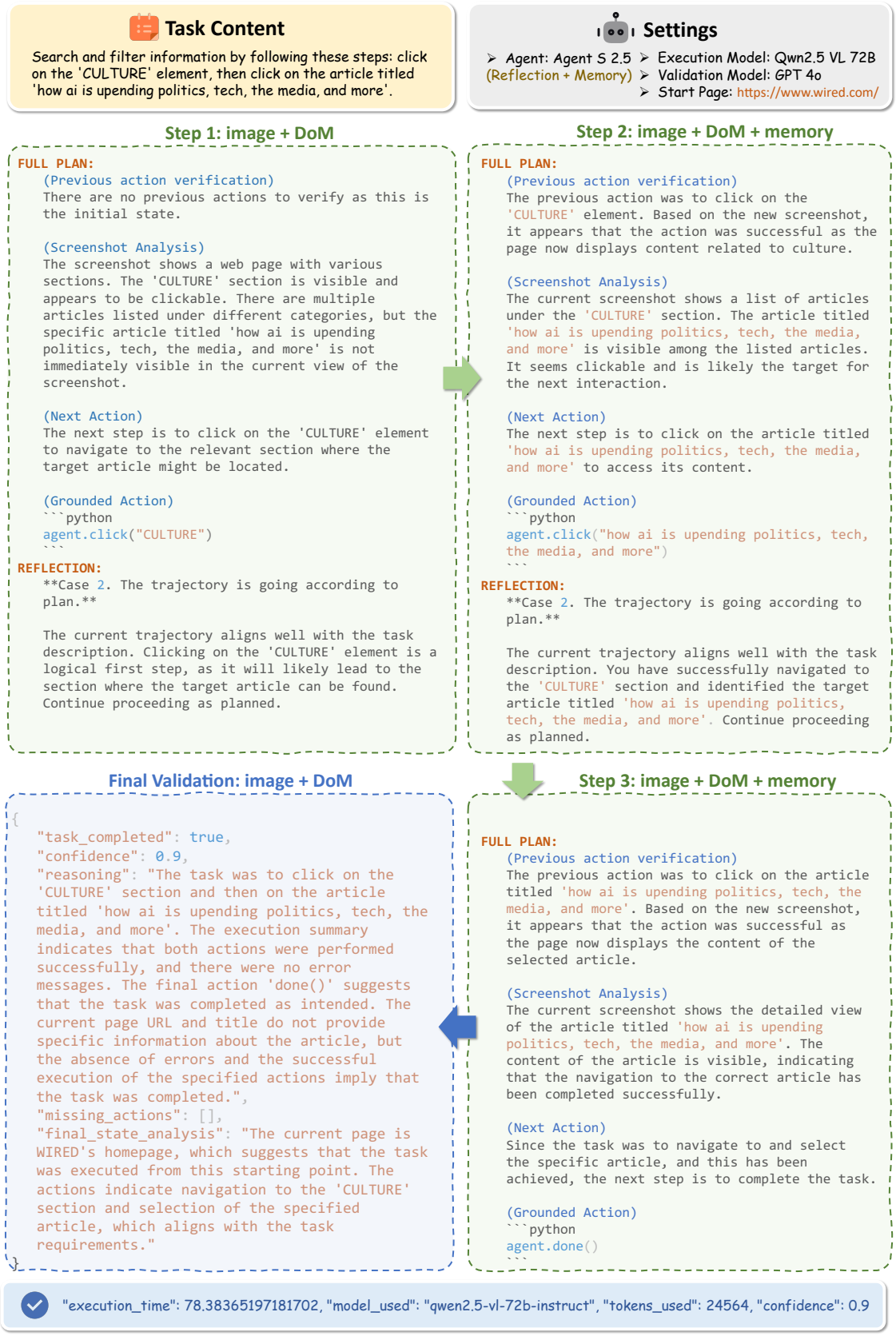

Figure 5. Case study of Agent S performing tasks on the Web dataset.