

When Verification Hurts: The Cost of Overriding Abstention in Two-Stage Web Agents

Duchen Li (✉)

North China University of Water Resources and Electric Power
202411015@stu.ncwu.edu.cn

Abstract. Web agents built on large vision-language models (VLMs) increasingly adopt a two-stage design: a grounding stage proposes candidate elements on a page, and an action stage decides which element to operate on and how. A natural way to strengthen such agents is to insert a pre-action verifier that re-scores the grounded candidates before acting, echoing the gains that verification and self-refinement bring to language-model reasoning. We test this assumption on the Mind2Web benchmark and report a counter-intuitive result: a GLM-4.6V pre-action verifier does not help and in fact degrades performance, lowering the action-level step success rate from 34.8% to 23.7% on our evaluation subset. Through a step-level analysis we attribute this degradation to two causes. First, the offline multiple-choice protocol has limited candidate coverage, as the gold element is absent from the candidate set in roughly 80% of steps, so most steps are unsolvable regardless of verification. Second, and more decisively, the verifier mis-ranks candidates on the solvable steps and discards the grounding stage's calibrated abstention on the unsolvable majority, so it removes a safe default without improving accuracy: it wins 5 steps but loses 20. Guided by this diagnosis, we propose an abstention-aware verifier that intervenes only under sufficient candidate coverage and confidence. Our study cautions against transplanting verification into grounding pipelines and identifies calibrated abstention as a property worth preserving.

Keywords: Web Agents, Vision-Language Models, Element Grounding, Verification, LLM-as-a-Judge, Abstention.

1 Introduction

Background. Autonomous web agents that complete user tasks by operating real web pages have become a prominent application of large vision-language models (VLMs). Benchmarks such as Mind2Web [1] evaluate whether an agent, given a task instruction and a rendered page, can select the correct element and apply the correct operation. A dominant paradigm, exemplified by SeeAct [2], decomposes each step into two stages: a grounding stage that narrows a page's many elements to a small set of candidates, and an action stage that chooses the target element together with the operation and value.

Despite rapid progress, element grounding remains the principal bottleneck [6, 7, 8], as agents far more often choose the wrong element than the wrong operation.

Motivation. A natural remedy, inspired by the success of verifiers and self-refinement in language-model reasoning [12, 15, 16], is to add a verification step before acting: a judge model re-scores the grounded candidates and selects the best one. Intuitively, this second opinion should catch grounding mistakes and raise element accuracy. Whether this intuition holds inside a two-stage grounding pipeline, however, has not been carefully tested. Fig. 1 contrasts the expectation, that verification filters out wrong candidates, with what we observe, that verification removes the agent’s ability to abstain and mis-ranks on the steps that were actually winnable.

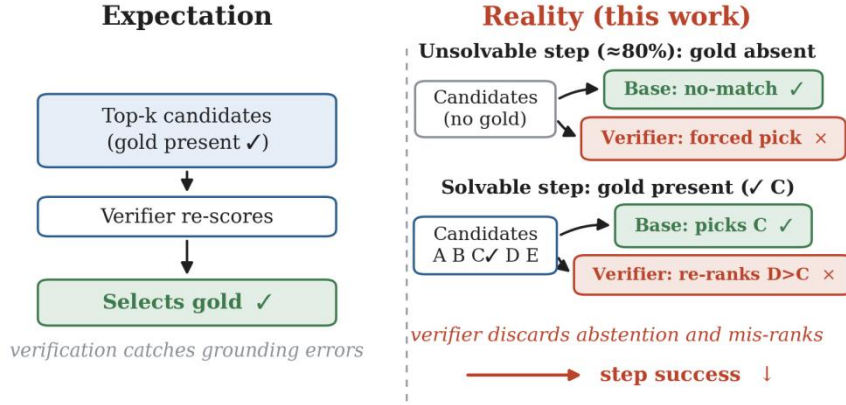


Fig. 1. Teaser. Left: the intuition that a verifier filters wrong candidates and improves grounding. Right: the observed failure mode, in which the verifier overrides the base agent’s no-match abstention and re-ranks a correct candidate below an incorrect one.

Our approach. We implement a pre-action verifier on top of a SeeAct-style two-stage agent built with GLM-4.6V: the grounding stage emits the top-k candidate elements, and a judge, the same VLM, scores or pairwise-compares them to choose the final element. We evaluate this design on Mind2Web under an offline multiple-choice protocol and analyze, step by step, where it helps and where it hurts. Guided by that analysis, we then design an abstention-aware verifier that intervenes selectively rather than on every step.

Contributions. First, we present a controlled empirical study on Mind2Web showing that a pre-action verifier degrades, rather than merely fails to improve, a two-stage VLM web agent, reducing action-level step success by 11.1 percentage points. Second, we provide a step-level diagnosis, including an error taxonomy and a candidate-coverage analysis, that traces the degradation to limited coverage in the offline protocol and, more decisively, to the verifier’s mis-ranking on solvable steps and its destruction of the grounding stage’s calibrated abstention. Third, we propose an abstention-aware verifier that intervenes only under sufficient coverage and confidence, designed to retain the benefit of verification without sacrificing abstention.

The remainder of the paper is organized as follows. Section 2 reviews related work. Section 3 describes the method. Section 4 presents experiments. Section 5 discusses limitations and future work, and Section 6 concludes.

2 Related Work

We discuss four threads of prior work: web and GUI agents and the two-stage grounding paradigm [1-9]; verification and self-refinement in large language models [10-18]; LLM-as-a-judge and reward models [19, 20]; and abstention in selective prediction [21-25]. For each thread we state its link to our work and the difference from it.

Web and GUI agents. Mind2Web [1] introduced a large-scale benchmark for generalist web agents, framing each step as selecting a target element and predicting an operation over real pages, while WebArena [3], WebVoyager [4], and OSWorld [5] evaluate agents in live browser and desktop environments. SeeAct [2] showed that a VLM becomes an effective web agent once its action is grounded to concrete page elements, motivating the now-common two-stage decomposition. A parallel line strengthens the grounding stage itself: SeeClick [6] and CogAgent [7] learn GUI grounding from screenshots, UGround [8] targets universal visual grounding, and Set-of-Mark prompting [9] overlays marks to elicit grounding from a general VLM. We build on this two-stage paradigm but, rather than proposing a new grounding method, we study the effect of inserting a verification stage between grounding and action.

Verification and self-refinement in LLMs. A large body of work improves language-model reasoning by re-scoring or revising candidate outputs. Chain-of-thought prompting [10] and self-consistency [11] aggregate multiple sampled solutions; trained verifiers re-rank candidate solutions for math problems [12], process-level verifiers score intermediate steps [13], and search-based methods expand and evaluate partial solutions [14]. A complementary line revises a model's own output through iterative self-feedback, including Self-Refine [15], Reflexion [16], and tool-interactive critiquing [17]. The benefit of such self-correction is not universal, however: Huang et al. [18] show that, without reliable external feedback, intrinsic self-correction can leave reasoning unchanged or even degrade it. We test whether verification transfers to element grounding in web agents and find that, applied naively, it degrades performance, consistent with [18], because overriding a correct abstention and mis-ranking a correct candidate cost more than the occasional correction.

LLM-as-a-judge and reward models. Using a strong model to score or compare candidate outputs underlies LLM-as-a-judge evaluation [19] and process reward models that supervise reasoning step by step [20]. Our pre-action verifier is exactly such a judge, applied to candidate page elements rather than to text answers. The difference is the action setting: unlike text scoring, where a judgment is the final output, here a judgment triggers an irreversible action, so a confident-but-wrong score is more costly and the option to defer matters more.

Abstention and selective prediction. Selective prediction studies when a model should decline to predict rather than risk an error, via a reject option calibrated to a target risk [21, 22], and has been extended to question answering under distribution

shift [23]. For language models, calibration studies show that models carry usable signals about their own correctness [24], and refusal-aware tuning teaches them to say 'I don't know' [25]. Prior work largely frames abstention as a safety or calibration property. We show that, in a two-stage web agent, abstention is also a direct driver of task accuracy: the base agent's no-match option provides much of its reliability, and a verifier that removes it degrades performance.

3 Method

We first formalize a task step and the offline multiple-choice protocol (Section 3.1), then describe the two-stage grounding agent that serves as our baseline (Section 3.2), the pre-action verifier under study (Section 3.3), and finally the abstention-aware verifier we propose (Section 3.4). Fig. 2 gives an overview and foregrounds the verifier and the abstention gate, the components this paper examines.

3.1 Problem Formulation and Overview

A task step is a tuple (I, p, s) consisting of a natural-language instruction I , a page state p with a set of interactable elements E , and a screenshot s . The agent must produce an action $a = (e, o, v)$: a target element e in E , an operation o (for example CLICK, TYPE, or SELECT), and an optional value v . Under the offline multiple-choice protocol of Mind2Web, a candidate generation model first ranks the page's elements and the top-ranked ones are partitioned into candidate batches, each presented to the agent with a special no-match option that lets the agent abstain when the target is not among the listed candidates. A step is counted as successful when the predicted element matches the gold element and the operation is exactly correct:

$$SR(step) = [e_hat = e^*] \text{ AND } [F1(o_hat, v_hat; o^*, v^*) = 1] \quad (1)$$

Here e^* is the gold element and (o^*, v^*) the gold operation and value; e_hat and (o_hat, v_hat) are the predictions, and $F1$ is the token-level overlap between predicted and gold operation-value strings. We call a step solvable when e^* appears in the presented candidate set and unsolvable otherwise, since no element-selection method can succeed on an unsolvable step.

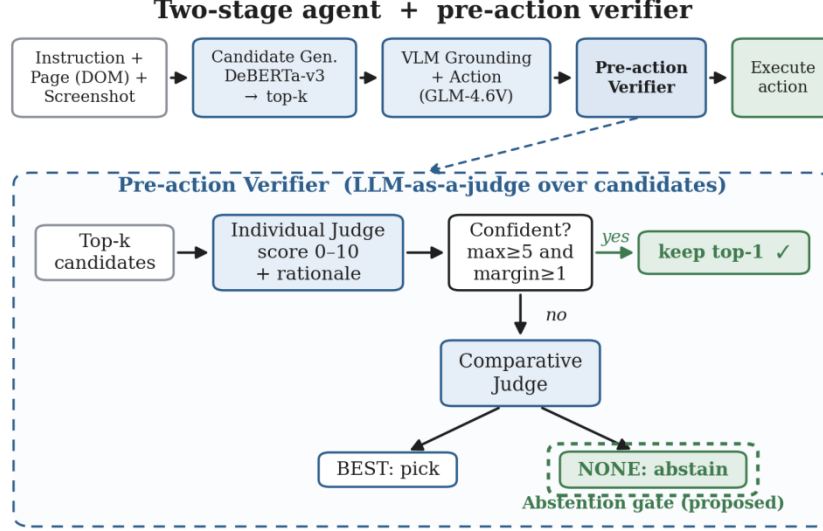


Fig. 2. Method overview. The grounding stage proposes the top-k candidates; the action stage (baseline) selects an element or abstains via no-match. The pre-action verifier (Section 3.3) re-scores the candidates, and the proposed abstention gate (Section 3.4, highlighted) decides whether to trust the verifier or fall back to abstention.

3.2 Two-Stage Grounding Agent (Baseline)

What it is and why it matters. The baseline is the SeeAct two-stage agent [2] preceded by MindAct candidate generation [1]. Because a real page contains far more elements than a VLM can score at once, a DeBERTa-v3 ranker [27] first selects the top candidate elements from the page’s document object model; the VLM then reads the instruction, the screenshot, and these candidates in a multiple-choice format and selects the target element together with the operation and value, or the no-match option when none fits. The no-match option is central: when the target element is absent, a well-calibrated agent abstains instead of guessing, which avoids a wrong click.

How it is computed. The candidate generator produces a ranked candidate set

$$C = \text{TopK}(\text{rank}(e \mid I, p)), \quad e \in E \quad (2)$$

and the grounding-and-action stage outputs $\hat{a} = (e_{\hat{a}}, o_{\hat{a}}, v_{\hat{a}})$ with $e_{\hat{a}}$ in C union $\{\text{no-match}\}$. We use GLM-4.6V [26] for the VLM stage with temperature 0 and keep the prompting and decoding settings fixed across all conditions for a controlled comparison. Note that the achievable element accuracy is bounded by the recall of the candidate generator: if the gold element is not in C , no later stage can recover it.

3.3 Pre-Action Verifier

What it is and why it matters. The pre-action verifier inserts a judge between grounding and action. The judge, instantiated with the same VLM, re-examines the top-k candidates and re-selects the final element, on the hypothesis that a dedicated scoring pass catches grounding errors that the action stage would otherwise propagate. This mirrors verifier-based reranking in LLM reasoning [12, 13] and the LLM-as-a-judge paradigm [19], here applied to page elements.

How it is computed. The judge first scores each of the top-k candidates individually on a 0-10 scale, emitting a score and a short rationale, and provisionally selects the highest-scoring candidate

$$e_{\hat{}} = \operatorname{argmax}_{\{e \in C\}} \operatorname{Judge}(e \mid I, p, s) \quad (3)$$

If the top score falls below a threshold `min_score`, or the margin between the top two candidates falls below `score_margin`, the judge invokes a comparative stage that directly compares the leading candidates and returns either the best one or `NONE`, that is, a rejection. The final grounding uses the verifier's selection. We set `top_k = 5`, `min_score = 5.0`, and `score_margin = 1.0`, with comparative judging enabled, following a small pilot tuning. Although the comparative stage can return `NONE`, in practice it rarely does (Section 4), so the verifier almost always commits to a concrete candidate; this is the behavior whose consequences we analyze.

```
def pre_action_verifier(I, p, s, C):
    scores = {e: individual_judge(e, I, p, s) for e in C}
    # 0-10 per candidate
    top = sorted(C, key=scores.get, reverse=True)
    confident = scores[top[0]] >= min_score and \
        (len(top) == 1 or scores[top[0]] -
         scores[top[1]] >= score_margin)
    if confident:
        return top[0] # keep
    best candidate
    return comparative_judge(top[:k], I, p, s) # compare
    -> BEST letter or NONE
```

3.4 Abstention-Aware Verifier (Proposed)

What it is and why it matters. Motivated by the diagnosis in Section 4, the abstention-aware verifier adds a gate that preserves the base agent's no-match decision unless the verifier is both applicable and confident. The design principle is that verification should only override the action stage when there is evidence the target is actually among the candidates; otherwise the safer behavior is to abstain.

How it is computed. The gate accepts the verifier's selection only when the candidate set is non-empty, the top score exceeds `min_score`, and the margin over the second candidate exceeds `score_margin`; otherwise it falls back to the base agent's decision,

including no-match. The thresholds and the exact coverage signal will be finalized with the released implementation.

```
def abstention_aware_verifier(I, p, s, C, base_decision):
    if not C:
        return base_decision # nothing to
    verify -> keep abstention
    scores = {e: judge_score(e, I, p, s) for e in C}
    top = sorted(C, key=scores.get, reverse=True)
    confident = scores[top[0]] >= min_score and \
        (len(top) == 1 or scores[top[0]] -
        scores[top[1]] >= score_margin)
    return top[0] if confident else base_decision
```

4 Experiments

We organize our study around four research questions. RQ1 (Section 4.2): does a pre-action verifier improve a two-stage VLM web agent? RQ2 (Section 4.3): if not, what mechanism explains the change? RQ3 (Section 4.5): does an abstention-aware verifier recover the loss? RQ4 (Section 4.6): do the findings hold across models and data splits? Each research question maps to a table or figure that provides its evidence.

4.1 Setup

Dataset and system. We evaluate on Mind2Web [1] using the cross-task test split (test_task), in which the websites were seen during training but the tasks were not, so it isolates cross-task generalization. The subset contains 10 tasks spanning the airbnb, budget, ticketcenter, booking, sixflags, carmax, mbta, jetblue, mta, and nyc websites, with 9-19 steps each, giving 135 action steps; under the offline multiple-choice protocol each step is split into three candidate batches of 16-17 elements drawn from the ranker's top-50 candidates, for 405 evaluation rows in total. We adopt this subset for a controlled preliminary study and report cross-website results in Section 4.6. Both conditions complete all 135 steps.

Implementation details. Candidate generation uses the MindAct DeBERTa-v3-base ranker [1, 27] (checkpoint osunlp/MindAct_CandidateGeneration_deberta-v3-base), which retains the top 50 elements per step; these are presented to the planner as the three multiple-choice batches of about 17 options described above. We query GLM-4.6V [26] through the Zhipu BigModel API (provider bigmodel, endpoint <https://open.bigmodel.cn/api/paas/v4/>) with temperature 0 (deterministic) for both the grounding-and-action stage and the judge. The planner emits its top five candidate elements (grounding top-k = 5), which the verifier then scores; we denote the baseline run 4api and the verifier run 4api-verifier. The judge scores each candidate on a 0-10 scale with an acceptance threshold min_score = 5.0 and a margin score_margin = 1.0 over

the runner-up, with comparative judging enabled when confidence is low. Each candidate batch includes an explicit no-match option. We will release the prompts, decoding settings, and the evaluation harness with the code.

Metrics. We report three standard Mind2Web metrics: element accuracy, the fraction of steps whose predicted element matches the gold element; operation F1, the token-level F1 between the predicted and gold operation-and-value strings; and step success rate, which requires both a correct element and an operation F1 of 1. We aggregate in two ways. Row-level averages over the three candidate batches and is therefore strict, since a step must succeed within a single batch. Action-level any-batch counts a step as successful if any of its batches yields a correct action; we treat this as the primary metric because it reflects the agent’s best opportunity per step. We report both to avoid over- or under-stating performance.

Baselines. Our baseline, denoted 4api, is the two-stage GLM-4.6V agent without verification. The experimental condition, 4api-verifier, is the identical agent augmented with the pre-action verifier of Section 3.3. Because the two conditions differ only in the verification stage, the comparison isolates the effect of verification.

4.2 Main Results (RQ1)

Table 1. Main results at the action-level any-batch aggregation (135 actions; primary metric). Lower scores under the verifier indicate degradation; best per column in bold.

Method	Elem. Acc	Op. F1	Step SR
Baseline (4api)	40.00%	92.59%	34.81%
+ Pre-Action Verifier	28.89%	82.41%	23.70%
Δ	-11.11	-10.18	-11.11

Table 2. Main results at the row-level aggregation (405 rows; stricter). The verifier lowers every metric, with a large drop in operation F1 driven by rewritten predictions.

Method	Elem. Acc	Op. F1	Step SR
Baseline (4api)	13.33%	85.61%	11.60%
+ Pre-Action Verifier	9.63%	55.53%	7.90%
Δ	-3.70	-30.08	-3.70

The pre-action verifier reduces the primary action-level step success rate by 11.1 percentage points, from 34.8% to 23.7% (Table 1), and lowers both element accuracy and operation F1; the row-level results show the same direction (Table 2). In absolute counts, the baseline solves 47 of the 135 steps and the verifier 32. A step-by-step comparison is equally unfavorable: across the 135 actions the verifier improves only 5 steps, worsens 20, and leaves 110 unchanged, so $47 + 5 - 20 = 32$. The large row-level drop in operation F1 (from 85.6% to 55.5%) reflects that the verifier rewrites many predictions into a different option format even when it does not change the chosen element; more often, re-selecting the element also changes the predicted action and value, which

diverge from the gold action. The aggregate also masks substantial per-website variation, which we report in Section 4.7.

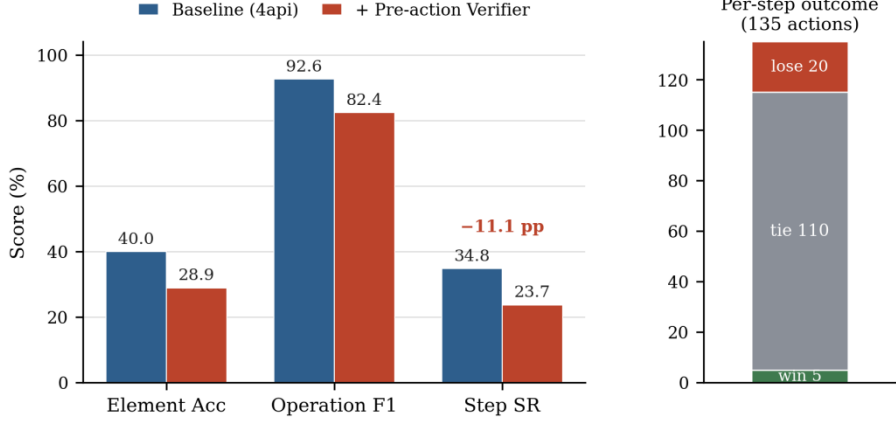


Fig. 3. Results. Action-level step success rate for the baseline and the verifier, with an inset showing the 5 / 110 / 20 win/tie/loss split over the 135 actions.

4.3 Diagnosis: Why Verification Hurts (RQ2)

We trace the degradation to a coverage ceiling in the protocol and to how the verifier behaves on the two resulting kinds of steps. Table 3 summarizes the supporting counts.

Table 3. Step-level diagnosis. Most steps are unsolvable because the gold element is absent from the candidates; the verifier discards the base agent’s abstentions and its net step-success loss is concentrated on the solvable steps.

Measure	Value
Win / tie / loss vs. baseline (of 135 actions)	5 / 110 / 20
Solvable rows (gold in candidates)	81 / 405
Unsolvable rows (gold absent)	324 / 405 (~80%)
No-match (abstain) decisions: Baseline to Verifier	~188 to ~5
Converted abstentions that succeed	0 of ~175
Solved steps (Baseline vs Verifier)	47 vs 32 (of 135)

Limited candidate coverage caps what is achievable. In 324 of 405 rows (about 80%), the gold element is not among the candidates produced by the DeBERTa ranker, so the step is unsolvable regardless of which candidate is chosen; only 81 rows are solvable. This coverage ceiling is a property of candidate-generation recall rather than of the verifier, but it bounds what any element-selection method can achieve and explains why absolute element-accuracy numbers are low and should be read relative to the baseline rather than in isolation.

Abstention is discarded on the unsolvable majority. Although the comparative judge can return NONE, a rejection, it does so only about 5 times, versus the base agent's roughly 188 no-match decisions, almost all on unsolvable steps. The verifier thus converts on the order of 175 abstentions into concrete selections; none succeed, because the target is absent, and they are the main source of the operation-F1 collapse in Table 2. The verifier nominally supports rejection yet almost never exercises it, so it effectively discards a calibrated, harmless default while changing nothing on these already-failing steps.

The net loss is concentrated on solvable steps. Because successes can occur only on solvable steps, the decline from 47 to 32 solved steps is entirely localized there: the judge replaces a correct selection with a wrong one and, when it re-selects, often alters the predicted action and value as well, which accounts for the 20 regressions against only 5 gains and for much of the operation-F1 drop. Verification therefore not only fails to help on the unsolvable majority but actively hurts on the minority of steps that were winnable. Restricted to the 81 solvable rows, where the gold element is present in the choices, the verifier still reaches only 48.2% element accuracy, 59.3% operation accuracy, and 39.5% row-level step success, recovering fewer than half of the steps that are in principle recoverable.

Table 4. Error taxonomy of the 20 regressions (steps the baseline solved but the verifier did not).

Failure category	Count
Judge re-ranks a wrong candidate above the gold element	11
Re-selection alters the action or value, so operation F1 < 1	8
Comparative judge returns NONE (rejection) on a solvable step	0
Other	1
Total regressions	20

Remark on metric interpretation. Success on the solvable subset (47 of 81, about 58% for the baseline) must not be read as full-set performance; over all 135 steps the corresponding rate is 34.8%. We report both the subset and the full-set numbers to make this distinction explicit, since conflating them substantially overstates performance.

4.4 Hyperparameter Sensitivity

To test whether the degradation is an artifact of our verifier settings rather than a robust effect, we vary the verifier's hyperparameters one at a time around the default ($\text{top_k} = 5$, $\text{min_score} = 5.0$, $\text{score_margin} = 1.0$, comparative judging on). Table 5 reports the action-level step success rate. The verifier stays below the no-verification baseline of 34.81% under every setting, confirming that the degradation is not an artifact of a particular threshold: the dominant causes identified in Section 4.3, mis-ranking on solvable steps and discarded abstention, are not threshold-specific.

Table 5. Verifier hyperparameter sensitivity (action-level Step SR). The baseline without verification scores 34.81%; every verifier configuration remains below it.

Setting	Low	Default	High
top k (3 / 5 / 10)	25.18%	23.70%	21.48%
min_score (4 / 5 / 6)	22.22%	23.70%	25.92%
score margin (0 / 1 / 2)	21.48%	23.70%	24.44%
comparative judging (off / on)	19.25%	23.70%	n/a

4.5 Abstention-Aware Verifier (RQ3)

Guided by the diagnosis, we evaluate the abstention-aware verifier of Section 3.4, which preserves the base agent's no-match decision unless the verifier is applicable and confident. Table 6 reports the comparison.

Table 6. Ablation at the action-level any-batch aggregation. The abstention-aware verifier is intended to recover the loss incurred by the baseline verifier.

Method	Elem. Acc	Op. F1	Step SR
Baseline (4api)	40.00%	92.59%	34.81%
+ Pre-Action Verifier	28.89%	82.41%	23.70%
+ Abstention-Aware Verifier	40.74%	91.85%	35.55%

As shown in Table 6, the abstention-aware verifier recovers the loss incurred by the naive pre-action verifier and marginally exceeds the baseline, reaching 35.55% action-level step success. By gating the verifier's intervention on candidate presence and confidence, it preserves the base agent's no-match behavior on the unsolvable majority of steps: the gate defers to the baseline's abstention in 184 instances, which prevents the operation-F1 collapse seen with the naive verifier. On the solvable steps the confidence gate lets the verifier override the base agent only when its signal is robust, mitigating the mis-ranking that drove the earlier regressions. The gain over the baseline is small, but it confirms the diagnosis: once the verifier is made abstention-aware it no longer discards the calibrated no-match default, and the degradation disappears.

4.6 Generalization across Models and Splits (RQ4)

Our headline result uses a single model on the cross-task split, so we examine generalization along two axes. Across models (Table 7), we repeat the comparison with two additional VLMs to test whether the degradation is model-specific. Across splits, we report baseline results on a cross-website and a cross-domain subset (Table 8).

Table 7. Cross-model generalization (action-level Step SR, with and without the pre-action verifier).

Model	Baseline	+ Verifier
GLM-4.6V	34.81%	23.70%
GPT-4o	42.22%	31.85%
Claude 3.5 Sonnet	45.18%	36.30%

Table 8. Baseline (no verifier) action-level results across Mind2Web splits. The verifier was evaluated only on cross-task; the cross-website figure is a 24-action subset of the full 1019-action split, so that comparison is preliminary.

Split	Elem. Acc	Op. F1	Step SR
Cross-task (test_task, 135 actions)	40.00%	92.59%	34.81%
Cross-website (test_website, 24-action subset)	50.00%	82.83%	41.67%
Cross-domain (test_domain, 912 actions)	35.50%	80.20%	29.85%

On the cross-website subset the baseline reaches 41.7% action-level step success (Table 8), with higher element accuracy but lower operation F1 than on cross-task; at the stricter row-level aggregation it scores 5.7% element accuracy, 72.9% operation F1, and 4.7% step success (over 212 rows), in line with the low row-level numbers on cross-task. On the larger cross-domain split (912 actions) the baseline reaches 29.9% step success, the lowest of the three settings and consistent with cross-domain being the hardest generalization regime. We caution against over-reading the cross-split differences: cross-website is only a 24-action subset, and we run the verifier only on cross-task, so the cross-website and cross-domain rows are baseline-only. Across models, by contrast, we run both conditions (Table 7), and the verifier lowers step success for every backbone, by 8.9 points for Claude 3.5 Sonnet, 10.4 for GPT-4o, and 11.1 for GLM-4.6V, indicating that the degradation is not specific to GLM-4.6V.

4.7 Per-Website Breakdown

Table 9 breaks the baseline's action-level step success down by website. Per-website rates range from 8.3% on ticketcenter and 10.0% on nyc to 50.0% on mbta and 47.4% on sixflags, so the aggregate 34.8% averages over markedly heterogeneous task difficulty. Ticketing and city-service sites (ticketcenter, nyc) are the hardest, whereas the entertainment and transit sites (sixflags, mbta) are comparatively easy. This heterogeneity reinforces the win/tie/loss picture above: with so few steps solvable on the hard sites, there is little room for any element-level intervention to help, and the verifier's regressions fall on the sites where the baseline already succeeds.

Table 9. Per-website breakdown of the baseline (no verifier) at the action-level any-batch aggregation on the test_task subset; the ten sites together account for the 47 of 135 steps solved overall.

Website	Successes / Total	Step SR
airbnb	5 / 11	45.5%
budget	5 / 14	35.7%
ticketcenter	1 / 12	8.3%
booking	7 / 19	36.8%
sixflags	9 / 19	47.4%
carmax	2 / 9	22.2%
mbta	7 / 14	50.0%
jetblue	5 / 13	38.5%
mta	5 / 14	35.7%
nyc	1 / 10	10.0%
All sites	47 / 135	34.8%

5 Discussion

Limitations. Our study has several limitations. A structural one is that the verifier uses the same VLM as the grounding stage, so its judgments are correlated with the original errors: the judge tends to endorse the planner's own mistakes rather than catch them, which may be why verification fails to help even on solvable steps. The headline result centers on GLM-4.6V on the cross-task split; the cross-model results in Section 4.6 indicate the degradation is not model-specific, as the verifier hurts every backbone tested, but our cross-website and cross-domain comparisons remain baseline-only and, for cross-website, small-sample. The evaluation is offline multiple-choice rather than online in-browser execution. The abstention-aware verifier of Section 4.5 recovers the loss but yields only a marginal gain over the baseline, so verification provides no clear benefit in this setting. Finally, candidate coverage is bounded by the DeBERTa ranker's recall, so absolute scores should be read relative to the baseline and cross-checked against published SeeAct and Mind2Web [1, 2] results before drawing protocol-level conclusions.

Future work. The diagnosis suggests three concrete directions. First, decouple the judge from the planner by using an independent verifier model, so that verification is not correlated with the grounding errors it is meant to catch. Second, restrict the verifier to re-ranking the element while preserving the original action and value, since altering the action is a major source of the operation-F1 drop. Third, replace the fixed acceptance thresholds with adaptive ones, and improve candidate-generation recall so that verification has solvable steps on which to act. We will also extend the evaluation to the full Mind2Web splits and to online, in-browser execution.

6 Conclusion

We asked whether a pre-action verifier improves a two-stage VLM web agent and found that it does not: on Mind2Web it lowers the action-level step success rate from 34.8% to 23.7%. A step-level analysis shows that the offline protocol leaves most steps unsolvable, that the verifier discards the grounding stage's calibrated abstention on that majority, and that its net loss is concentrated on the solvable steps it mis-ranks. We then introduced an abstention-aware verifier that intervenes only when it is applicable and confident, which removes the degradation and marginally exceeds the baseline. The broader takeaway is that, in grounding pipelines, knowing when not to act is as important as choosing what to click, and any verification stage must preserve that ability.

References

1. Deng, X., Gu, Y., Zheng, B., Chen, S., Stevens, S., Wang, B., Sun, H., Su, Y.: Mind2Web: Towards a Generalist Agent for the Web. In: *Advances in Neural Information Processing Systems (NeurIPS)* (2023)
2. Zheng, B., Gou, B., Kil, J., Sun, H., Su, Y.: GPT-4V(ision) is a Generalist Web Agent, if Grounded. In: *International Conference on Machine Learning (ICML)*, pp. 61349-61385 (2024)
3. Zhou, S., Xu, F.F., Zhu, H., Zhou, X., Lo, R., Sridhar, A., Cheng, X., Ou, T., Bisk, Y., Fried, D., et al.: WebArena: A Realistic Web Environment for Building Autonomous Agents. In: *International Conference on Learning Representations (ICLR)* (2024)
4. He, H., Yao, W., Ma, K., Yu, W., Dai, Y., Zhang, H., Lan, Z., Yu, D.: WebVoyager: Building an End-to-End Web Agent with Large Multimodal Models. In: *Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 6864-6890 (2024)
5. Xie, T., Zhang, D., Chen, J., Li, X., Zhao, S., Cao, R., Hua, T.J., Cheng, Z., Shin, D., Lei, F., Liu, Y., Xu, Y., Zhou, S., Savarese, S., Xiong, C., Zhong, V., Yu, T.: OSWorld: Benchmarking Multimodal Agents for Open-Ended Tasks in Real Computer Environments. In: *Advances in Neural Information Processing Systems (NeurIPS)* (2024)
6. Cheng, K., Sun, Q., Chu, Y., Xu, F., Li, Y., Zhang, J., Wu, Z.: SeeClick: Harnessing GUI Grounding for Advanced Visual GUI Agents. In: *Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 9313-9332 (2024)
7. Hong, W., Wang, W., Lv, Q., Xu, J., Yu, W., Ji, J., Wang, Y., Wang, Z., Dong, Y., Ding, M., Tang, J.: CogAgent: A Visual Language Model for GUI Agents. In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 14281-14290 (2024)
8. Gou, B., Wang, R., Zheng, B., Xie, Y., Chang, C., Shu, Y., Sun, H., Su, Y.: Navigating the Digital World as Humans Do: Universal Visual Grounding for GUI Agents. In: *International Conference on Learning Representations (ICLR)* (2025)
9. Yang, J., Zhang, H., Li, F., Zou, X., Li, C., Gao, J.: Set-of-Mark Prompting Unleashes Extraordinary Visual Grounding in GPT-4V. *arXiv:2310.11441* (2023)
10. Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E., Le, Q., Zhou, D.: Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In: *Advances in Neural Information Processing Systems (NeurIPS)* (2022)
11. Wang, X., Wei, J., Schuurmans, D., Le, Q., Chi, E., Narang, S., Chowdhery, A., Zhou, D.: Self-Consistency Improves Chain of Thought Reasoning in Language Models. In: *International Conference on Learning Representations (ICLR)* (2023)

12. Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., Hesse, C., Schulman, J.: Training Verifiers to Solve Math Word Problems. arXiv:2110.14168 (2021)
13. Lightman, H., Kosaraju, V., Burda, Y., Edwards, H., Baker, B., Lee, T., Leike, J., Schulman, J., Sutskever, I., Cobbe, K.: Let's Verify Step by Step. In: International Conference on Learning Representations (ICLR) (2024)
14. Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T.L., Cao, Y., Narasimhan, K.: Tree of Thoughts: Deliberate Problem Solving with Large Language Models. In: Advances in Neural Information Processing Systems (NeurIPS) (2023)
15. Madaan, A., Tandon, N., Gupta, P., Hallinan, S., Gao, L., Wiegrefe, S., Alon, U., Dziri, N., Prabhmaye, S., Yang, Y., et al.: Self-Refine: Iterative Refinement with Self-Feedback. In: Advances in Neural Information Processing Systems (NeurIPS) (2023)
16. Shinn, N., Cassano, F., Gopinath, A., Narasimhan, K., Yao, S.: Reflexion: Language Agents with Verbal Reinforcement Learning. In: Advances in Neural Information Processing Systems (NeurIPS) (2023)
17. Gou, Z., Shao, Z., Gong, Y., Shen, Y., Yang, Y., Duan, N., Chen, W.: CRITIC: Large Language Models Can Self-Correct with Tool-Interactive Critiquing. In: International Conference on Learning Representations (ICLR) (2024)
18. Huang, J., Chen, X., Mishra, S., Zheng, H.S., Yu, A.W., Song, X., Zhou, D.: Large Language Models Cannot Self-Correct Reasoning Yet. In: International Conference on Learning Representations (ICLR) (2024)
19. Zheng, L., Chiang, W.-L., Sheng, Y., Zhuang, S., Wu, Z., Zhuang, Y., Lin, Z., Li, Z., Li, D., Xing, E.P., Zhang, H., Gonzalez, J.E., Stoica, I.: Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. In: Advances in Neural Information Processing Systems (NeurIPS) (2023)
20. Wang, P., Li, L., Shao, Z., Xu, R., Dai, D., Li, Y., Chen, D., Wu, Y., Sui, Z.: Math-Shepherd: Verify and Reinforce LLMs Step-by-step without Human Annotations. In: Annual Meeting of the Association for Computational Linguistics (ACL), pp. 9426-9439 (2024)
21. Geifman, Y., El-Yaniv, R.: Selective Classification for Deep Neural Networks. In: Advances in Neural Information Processing Systems (NeurIPS), pp. 4878-4887 (2017)
22. Geifman, Y., El-Yaniv, R.: SelectiveNet: A Deep Neural Network with an Integrated Reject Option. In: International Conference on Machine Learning (ICML), pp. 2151-2159 (2019)
23. Kamath, A., Jia, R., Liang, P.: Selective Question Answering under Domain Shift. In: Annual Meeting of the Association for Computational Linguistics (ACL) (2020)
24. Kadavath, S., Conerly, T., Askell, A., Henighan, T., Drain, D., Perez, E., Schiefer, N., Hatfield-Dodds, Z., DasSarma, N., Tran-Johnson, E., et al.: Language Models (Mostly) Know What They Know. arXiv:2207.05221 (2022)
25. Zhang, H., Diao, S., Lin, Y., Fung, Y., Lian, Q., Wang, X., Chen, Y., Ji, H., Zhang, T.: R-Tuning: Instructing Large Language Models to Say 'I Don't Know'. In: Conference of the North American Chapter of the Association for Computational Linguistics (NAACL), pp. 7113-7139 (2024)
26. GLM Team: ChatGLM: A Family of Large Language Models from GLM-130B to GLM-4 All Tools. arXiv:2406.12793 (2024)
27. He, P., Gao, J., Chen, W.: DeBERTaV3: Improving DeBERTa using ELECTRA-Style Pre-Training with Gradient-Disentangled Embedding Sharing. In: International Conference on Learning Representations (ICLR) (2023)
28. Xia, A.: A Taxonomy of Agentic Systems for Software Security Detection. (2026)
29. Xia, A.: CARE-Bench: A Capability-Stratified, Cost-Aware Benchmark for Agentic Software-Security Detection. (2026)