Manual Maker: Grounded Generation and Agentic Self-Verification of Step-by-Step Software Manuals from Screen Recordings with Solar Pro 3

Jonghyun Lee[1]*, **Byeongjun Kim**[1]

[1]Yonsei University, Seoul, Republic of Korea
devjh@yonsei.ac.kr, kbjoz87@yonsei.ac.kr

Abstract

Writing step-by-step software manuals is a ubiquitous, repetitive task in enterprises, universities, and public institutions: an expert performs a workflow while a technical writer transcribes every click into numbered instructions with annotated screenshots. We present *Manual Maker*, a Chrome extension that supports the recording-to-editable-manual workflow. During recording, a content script logs clicks, text inputs, selections, and keyboard shortcuts, initiating a pre-action screenshot request for pointer interactions. Each recorded action is grounded in the DOM accessibility name of the exact element the user touched; an unnamed pointer target is auto-hidden from the rendered manual rather than given an invented label. Upstage Solar Pro 3 then drafts the manual under a strict JSON schema, and an agentic self-verification loop returns the draft to the model as a reviewer that flags unsupported or vague steps, which are selectively rewritten in batched calls—stopping when no weak verdict remains or after at most two refinement rounds. Sensitive inputs are masked before storage, while screenshot blurring is enabled by default and user-configurable; deterministic fallbacks provide step text when a model response omits a step. The package includes Chrome Web Store submission materials with a privacy policy and permission rationale. On public interaction traces, proxy missing-name rates reach 25.1% (790 A11y-CUA sighted-user pointer clicks) and 68.7% (645 Mind2Web annotated clicks); an exploratory Solar benchmark reports a model-judged weak-step rate of $14.4\% \rightarrow 5.6\%$ after one refine round on eight Mind2Web-derived traces. We report a by-construction evidence-reuse property of the deterministic path and close with the path to full deployment and engineering lessons.

Introduction

Every organization that runs software runs on manuals. Onboarding documents, ERP how-tos, and administrative-system walkthroughs share one production process: a person who knows the workflow performs it while transcribing each interaction into "Step 7: click the *Submit* button, then . . . " with a cropped, annotated screenshot. The work is mechanical but slow—documenting a ten-minute workflow routinely takes hours—and the result decays as the interface changes, so it is redone continually. Commercial capture tools shorten the screenshot work but still leave sentence writing, step segmentation, and annotation to the human.

Large language models seem like an obvious fix: give the model a log of what the user did and ask for a manual. In practice, a one-shot prompt over a raw event log fails in characteristic ways. The DOM often does not know what the user saw—an icon button is `<button class="btn-x">`, and text inside a canvas or `iframe` is invisible—so the model guesses, and the manual confidently names buttons that do not exist. Screenshots taken *after* a click show the wrong screen. And a single generation pass has no mechanism to notice that step 12 contradicts the recorded action it is supposed to describe.

Manual Maker is a packaged, pilot-ready Chrome extension that addresses each failure with a specific mechanism:

- **Pre-action capture request.** The content script requests a screenshot on `pointerdown`—before any navigation fires—and attaches the resulting image to the step only when the `click` is confirmed. If that request is stale or unavailable, it requests a new image so recording can continue. Text inputs are captured on `change`, after the value is committed, so filled fields are visible.
- **DOM-grounded subjects, no guessing.** The model never sees a raw event log—only a per-step digest that contains the accessibility name of the interacted element when one is available. When the DOM cannot name a pointer target (the icon-button and `iframe` cases above), the draft receives no target name and the step is auto-hidden before rendering and export, with its reason shown and one-click restoration; it is never assigned an invented label. This trades completeness for faithfulness, and is the deployed behavior. Recovering those unnameable targets with a visual-grounding service (Upstage Document Parse) is left to future work (Path to Deployment).
- **Schema-constrained drafting.** Solar Pro 3 receives a per-step digest (action, accessible name, role, surrounding context, page title) and writes titles, bodies, chapter labels, and an overview under a strict JSON schema, with prompt rules that forbid inventing UI elements, quoting user-typed values, or narrating intent.
- **Agentic self-verification.** In agent mode the draft is re-

*Corresponding author.

turned to Solar Pro 3 acting as a temperature-zero reviewer that grades each step `ok` or `weak` against its source action; only weak steps are rewritten in one or more batched regeneration calls, for up to two rounds, stopping early when all steps pass. Every refinement stage fails safe: a failed evaluation batch leaves its steps untouched, and a deterministic rule-based phrase generator supplies text for any step omitted from an otherwise successful draft.

The extension is feature-complete and packaged: a versioned Web Store packaging pipeline, store listing, privacy policy, and per-permission justifications are in the repository, and the build pipeline verifies on every release that the API key cannot leak into page-context code. What remains before full public deployment is principally operational—chiefly moving API access behind a proxy service, plus the optional enhancements described later—and we describe that path in a dedicated section. We submit to the Emerging Applications track as a packaged, pilot-ready application with a reproducible implementation analysis and a concrete route to full-scale deployment. In place of field-usage metrics, which we do not yet report, we provide *pre-pilot* evidence—measured on real public interaction traces and against the live Solar Pro 3 API (Evaluation, Appendix B)—that quantifies the failure modes the design targets and the behavior of the deployed model configuration, de-risking the consented pilot study planned next (Path to Deployment).

Application Context and Requirements

The target user is not a developer but the person in an organization on whom manual-writing falls: help-desk staff, administrators, instructors. Serving this user produced four hard requirements.

R1: Zero setup inside the workflow. Recording must happen in the browser where the work happens—one click to start, work as usual, one click to stop. Any "perform the task in our special environment" design fails in practice, since the workflows to document live in production ERP, LMS, and groupware systems.

R2: Factuality over fluency. A manual that reads well but names a nonexistent menu is worse than no manual—the reader is following instructions inside a system they by definition do not understand. The generation pipeline must be biased so that hallucination, not awkward phrasing, is the failure it spends effort eliminating.

R3: Privacy by construction. Recordings capture real screens of real systems, including password fields and personal data. Redaction must occur before persistence, not as a post-processing afterthought.

R4: Graceful degradation. The tool operates over a commercial LLM API. Timeouts, truncated JSON, and 5xx responses are routine operating conditions; none may destroy a recording session or block manual delivery.

System Overview

Manual Maker is a Manifest V3 Chrome extension of roughly 8,000 lines of TypeScript (Figure 1). A content script observes user actions and collects coordinates; a background service worker owns session state, screenshot capture, and all model access; an offscreen document holds the `MediaStream` for optional full-session video. The editor, popup, options, and preview pages are TypeScript single-page UIs. The pipeline is:

1. **Record.** The content script logs pointer actions, text inputs, selections, checkbox changes, and keyboard shortcuts. Each record retains a CSS selector, accessible name, ARIA role, and local context. For a pointer action, it requests a tab screenshot at `pointerdown`; the completed request is associated with that action. It also records actions in `window.open()` popups.
2. **Redact.** Fields that look like passwords or national ID numbers are masked to `••••` before storage, and the corresponding screenshot region is blurred; the original value is never persisted (*R3*).
3. **Ground.** Each action's subject is resolved to the DOM accessibility name of the element the user acted on when available; an unnamed pointer target yields a vague draft step that is auto-hidden from the rendered manual (its reason shown, restorable), never given an invented label.
4. **Draft.** Solar Pro 3 writes the manual from per-step digests under a strict JSON schema.
5. **Verify and refine.** The agentic loop grades and selectively rewrites weak steps until all pass or the round budget is exhausted.
6. **Edit and export.** The user reviews steps with click positions highlighted by a red ring, edits or annotates freely (hand-edited steps are never overwritten by regeneration), and exports. All four export formats—Markdown, Word, PDF, ZIP—are rendered from one shared document structure, so step and image numbering cannot diverge across formats.

Use of AI Technology

Underlying AI Service: Solar Pro 3

The deployed pipeline is built on one commercial AI service from Upstage. *Solar Pro 3* is the current flagship of the Solar family, whose lineage began with depth up-scaled SOLAR 10.7B (Kim et al. 2024). It is a Mixture-of-Experts (MoE) model of 102B total parameters that activates only $\approx$12B per token (Upstage 2026a): for a token representation x the decoder layer computes

$$\mathrm{MoE}(x) = \sum_{e \in \mathrm{Top}_k(x)} g_e(x)\, f_e(x),\ g(x) = \mathrm{softmax}(W_g x), \tag{1}$$

routing x through only the top-k experts f_e weighted by the gate g, so inference cost tracks the 12B active parameters while capacity tracks the 102B total. This is what lets a multi-call refine loop over long recordings stay within a production latency and cost budget. Solar Pro 3 roughly *doubles* the agentic performance of its predecessor—Tau2-all 72.3 vs. 36.0, SWE-bench 28.6 vs. 14.5 (Upstage 2026a)—and is a strong Korean–English bilingual model (Ko-Arena-hard-v2 78.2), which matters directly here because most target

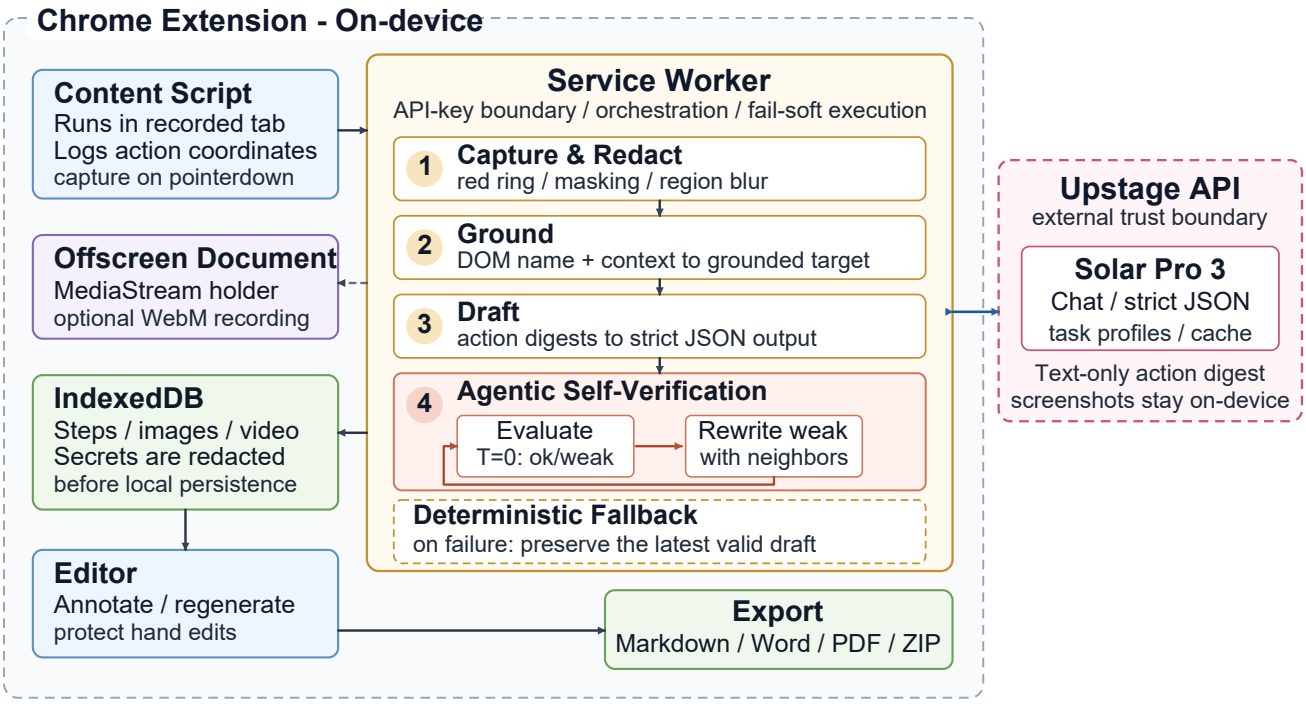

Figure 1: Deployed system architecture. The service worker performs capture and redaction, DOM grounding, drafting, and self-verification. Only textual action digests cross the trust boundary to Solar Pro 3; screenshots remain on-device. A deterministic fallback preserves the latest valid draft, and the editor exports a shared document representation. Document Parse is omitted because it is not deployed.

manuals are written in Korean about Korean-language enterprise systems. Three API capabilities are central to our design: *structured output* under a strict JSON schema, making draft, verdict, and rewrite responses machine-checkable by construction; *adjustable reasoning effort*, profiled per task (drafting, evaluation, single-step regeneration each get their own effort, temperature, and top-p); and *prompt caching*, keyed per recording session, which makes the multi-call loop affordable by reusing the shared prompt prefix across draft, evaluation, and rewrite calls. A second Upstage service, *Document Parse*—a document-understanding API that returns an image's layout elements with recognized text and normalized coordinates (Upstage 2026b)—could recover pointer targets the DOM cannot name; we leave that integration to future work (Path to Deployment).

Grounding the Draft in DOM Evidence

The central design decision is that *the model never sees a raw event log*; it sees a structured digest in which a concrete subject is included only when it is available from recorded evidence. Formally, a recording is a sequence $S = \langle a_1, \ldots, a_n \rangle$ where each action $a_i = (\tau_i, e_i, p_i, I_i)$ carries an action type τ_i (click, input, select, ...), DOM evidence e_i (selector, accessibility name, ARIA role, context), a viewport coordinate p_i, and an associated screenshot I_i.

Deployed grounding. The shipped build resolves the subject purely from the DOM: $\sigma(a_i) = \mathrm{name}(e_i)$, the accessibility name of the interacted element. For a pointer action whose name is empty—icon buttons, canvas-rendered controls, cross-origin `iframes`—no subject resolves ($\sigma(a_i) = \perp$). The generation prompt may describe such a step by its action and role, but the step is auto-hidden before rendering rather than given an unrelated on-screen string; users may restore it. This is a deliberate faithfulness-over-completeness choice (*R2*): a missing step is recoverable by the user; an invented one silently misleads.

Beyond the DOM (future work). The dropped steps are those the DOM cannot name but a human reader can see. A natural extension recovers their on-screen label from the screenshot—mapping the interaction coordinate to the smallest parsed layout element containing it via Document Parse—and uses that label as the subject when $\mathrm{name}(e_i)$ is empty. We defer this integration, together with its screenshot-transmission privacy cost, to future work (Path to Deployment); the shipped generation path uses DOM evidence alone.

Schema-Constrained Drafting with Solar Pro 3

Each action is condensed by a digest function ψ that retains its type, subject, role, context, and page. The subject is the DOM accessibility name when available. Selectors and user-typed values are excluded, except masked passwords. Solar Pro 3, denoted f_θ, then drafts the manual under a strict JSON schema (`strict: true`):

$$D^{(0)} = f_\theta(x_1, \ldots, x_n) = \langle d_1^{(0)}, \ldots, d_n^{(0)} \rangle, \qquad (2)$$

where each $d_i = (\text{title}_i, \text{body}_i, \text{chapter}_i)$ reuses the source index i verbatim, plus a manual-level title and overview. Long recordings are chunked into batches to respect response-length limits, with prompt caching keyed by session so repeated generations reuse the shared prefix. The system prompt encodes the factuality bias of *R2* as hard rules: the subject of every step is the recorded target and nothing else; on-screen advertisements or promotions that were not clicked must not be described; user-typed values are never quoted (they differ per reader) except masked passwords, which must be rendered as "enter your password"; the overview must describe the procedure actually recorded, not a guessed goal.

Agentic Self-Verification

One-shot generation leaves no mechanism for the system to notice its own hallucinations. Motivated by prior work on iterative self-refinement (Madaan et al. 2023; Shinn et al. 2023), we close the loop. At round t a temperature-zero reviewer r_θ grades each drafted step against its source digest,

$$v_i^{(t)} = r_\theta\Big(x_i, d_i^{(t)}\Big) \in \{\texttt{ok}, \texttt{weak}\}, \qquad (3)$$

and only the weak set $W_t = \{\, i : v_i^{(t)} = \texttt{weak} \,\}$ is rewritten, with weak digests partitioned into batches:

$$d_i^{(t+1)} = \begin{cases} f_\theta(\{x_j\}_{j \in W_t})_i & i \in W_t \\ d_i^{(t)} & i \notin W_t, \end{cases} \qquad (4)$$

iterating until $W_t = \emptyset$ or $t = T_{\max} = 2$. Any evaluation or rewrite failure returns the current $D^{(t)}$ unchanged. Three application-driven departures from generic self-refinement:

Judgments are structural, not aesthetic. The reviewer prompt defines `weak` narrowly: the draft names a UI element absent from the source action, contradicts the action, is too empty to act on, or leaks a masked password. It is explicitly told that generic phrasing of input values is intended, and to prefer `ok` when unsure. This prevents the loop from wasting tokens re-polishing acceptable prose—style is not a defect.

Verdicts are schema-validated. Evaluation runs at temperature zero under its own strict JSON schema; responses are parsed defensively, and any verdict with a malformed index or label is discarded rather than acted on, so an ambiguous judgment can never trigger regeneration of a sound step.

Repair is selective and bounded. Only steps judged weak are rewritten; the merge keeps the draft's chapter labels and discards empty rewrites, reducing—but not eliminating—the risk of narrative drift or step degradation. The loop runs at most two rounds and exits as soon as a round produces no weak verdicts. If an evaluation batch or a rewrite call fails, the current draft simply survives (*R4*)—quality improvement is an optional stage and is never allowed to lose an already-successful generation.

Single-step regeneration in the editor adds local context on top: a user can ask for one step to be rewritten, and the model sees the two steps on either side.

Deterministic Safety Net

Beneath the model sits a pure rule-based phrase generator that renders any recorded action into a serviceable sentence from its action type and accessible name. It supplies text for a step whose generated title or body is absent, including steps omitted from a partially successful batch. Clicks whose target cannot be named at all ("click the item") are auto-hidden from the manual with the reason shown and one-click restoration, on the principle that an unverifiable step is better omitted than guessed.

Privacy and Safe Operation

Privacy is enforced by architecture rather than policy. Sensitive field values are masked in the capture path before IndexedDB persistence—the unmasked value never exists at rest. Screenshot blurring for sensitive fields is enabled by default and user-configurable. In the deployed build *no screenshot ever leaves the device*: only the textual per-step digest is sent to Solar Pro 3 to draft prose. (Any future visual-grounding integration would transmit screenshots to Document Parse, a stricter privacy cost we weigh explicitly in the deployment path.) The developer operates no server and collects no telemetry. The Upstage API key is decoded in one module, which is imported only by the service worker; content scripts cannot import that module, and a release-gate script checks built artifacts for plain-text key leakage on every release. The distributed key is obfuscated against plain-text scraping bots—a stopgap whose real fix is the proxy service described below.

Evaluation

We report three analyses that together constitute the paper's *pre-pilot* evidence—quantitative grounding on real data that precedes, rather than substitutes for, a field study: a measurement of the DOM-blind rate that motivates the auto-hide design, computed on real public interaction traces; a by-construction faithfulness property of the deterministic path; and a qualitative failure map from internal development use. A reproducible Solar Pro 3 configuration benchmark additionally exercises product-aligned draft, evaluation, and refinement settings against the live API on real Mind2Web-derived recordings (Appendix B); all evaluation scripts are included in the supplementary material. The pipeline's pure logic—coordinate reconciliation, masking, chapter grouping, verdict parsing, highlight-bounds computation—is additionally covered by an automated suite of eight test modules ($\sim$1,600 lines) run with build verification and the key-leak check on every release.

Metrics

Let $P \subseteq S$ be the pointer actions (click, double-, right-click) and $\sigma(a_i) = \text{name}(e_i)$ the resolved subject of step i, the DOM accessibility name. A pointer step is *vague* when no subject resolves, $\sigma(a_i) = \perp$; vague steps are auto-hidden and never reach the reader. The *DOM-blind rate* is

$$p_{\text{blind}} = \frac{|B|}{|P|}, \qquad B = \{\, i \in P : \text{name}(e_i) = \varnothing \,\}, \qquad (5)$$

the share of pointer steps whose accessible name is empty. Counting a pointer step as covered only when a subject resolves, the deployed *correct-subject coverage* is $C_{\text{dom}} = 1 - p_{\text{blind}}$, and *manual retention* η is the fraction of recorded steps delivered—the DOM-blind pointer steps are exactly the ones withheld rather than mislabeled.

DOM-Blind Rate on Public Data

We approximate the shipped auto-hide rule on two public interaction datasets using recorded target attributes (supplementary scripts). These deterministic, dataset-adapted name proxies do not reproduce a live browser accessibility tree.

Among 790 sighted-user (SU-group) pointer clicks in A11y-CUA (Gubbi Mohanbabu et al. 2026)—blind and low-vision (BLVU) participants used keyboard navigation and contributed no pointer events—25.1% lack a usable proxy name. For context, WebAIM's 2024 Million report finds empty buttons on 28.2% and empty links on 44.6% of home pages (WebAIM 2024); those page-level prevalences are not directly comparable to our event-level proxy rates. The corresponding rate on 645 annotated clicks in Mind2Web's first training shard (Deng et al. 2023) is 68.7%, yielding 31.3% proxy subject coverage. Both indicate that attribute-only evidence often fails to name pointer targets, motivating auto-hide and future visual grounding.

Faithfulness of the Deterministic Path

Proposition 1. *For every non-vague step, the content-specific subject of the fallback sentence is drawn verbatim from the recorded DOM evidence* $\text{name}(e_i)$*; the generator introduces no content-specific token naming a UI element absent from the record.*

This holds by construction: `fallbackStep` composes its sentence from $\sigma(a_i) = \text{name}(e_i)$ and fixed templates, with no free-text model call. A TypeScript harness verifies the invariant on synthetic fixtures (Appendix B); a separate Mind2Web proxy check over 320 non-vague steps also finds **0 violations** but does not execute the shipped `fallbackStep`. The safety net cannot invent a UI-element name beyond the recorded evidence; this guarantee does not cover whether the recorded name is semantically correct, visible to the user, or sufficient to complete the task—it can only be silent (a vague step).

Path to Deployment

The extension is packaged at version 1.0.1 with prepared Chrome Web Store materials: a bilingual listing, privacy policy, and per-permission justifications written against Chrome's review criteria (including the rationale for `<all_urls>` host access, and dropping the `scripting` permission in favor of declarative content-script injection to reduce review friction).

Qualitative Failure Map

Table 1 records, from informal internal use across administrative and LMS workflows at the authors' institution during development, the dominant failure each mechanism suppresses. These are illustrative development observations, not measured prevalence or a controlled study.

Stage removed	Dominant failure reintroduced
Pre-action capture	Screenshots show the destination page, not the screen acted on
Auto-hide	Unnameable icon/iframe clicks get invented labels instead of being hidden
Schema rules	Overviews guess user intent; ads on screen are narrated as actions
Self-verification	Steps contradicting the recorded action survive to export

Table 1: Failure-mode map from informal development use: removing each mechanism reintroduced the listed failure. Illustrative observations, not a controlled study.

The remaining steps to full public deployment are as follows.

1. **API proxying.** Any key shipped in an extension is readable by its installer; public distribution therefore requires routing Upstage calls through a thin authenticated proxy with per-user quotas. This is an operational service addition—the extension's model-access module is already isolated to one file.
2. **Visual-grounding integration.** Add `documentParse` and `textAtPoint` to the generation path to recover proxy-missing pointer targets the current build hides—68.7% of annotated Mind2Web clicks in our shard lack a proxy name (Evaluation). A future study would measure what fraction of those targets can be recovered correctly; the proxy rate is an investigation set, not an expected recovery rate. The integration must weigh the privacy cost of screenshot transmission and add visual cross-examination against the parsed screen to guard against a mislabeled recovery.
3. **Web Store submission and review.** The proxy migration (item 1) changes the extension's network behavior and must be in place before submission; broad host-permission review is the expected bottleneck.
4. **Deployment study.** Instrumented consenting pilot at the authors' institution measuring authoring time versus the manual process, generation cost per manual, and weak-step rates in the wild. A field study—author time and reader task-completion versus hand-written and one-shot baselines—is planned.

Roadmap: Deepening the Use of Solar

Three Solar-based extensions are planned. **Visual cross-examination**: Document Parse output per screenshot would let the reviewer check the draft against the *screen*, not just the action log, flagging a named button absent from that screenshot's parsed elements and closing the last gap a plausible-but-wrong label can pass through. **Manuals as conversational knowledge**: embedding the steps and chapters of the generated corpus enables a Solar-powered assistant that answers procedural questions ("how do I file a travel reimbursement?") with the exact steps and screenshots from the

relevant manual. **Cross-language delivery**: generation is already bilingual, and translating a finished manual while pinning step structure, screenshots, and verbatim on-screen labels is a constrained-generation task the schema machinery supports.

Lessons Learned

Ground first, generate second. Most quality gains came from what the model receives, not from prompt cleverness: resolving each action to its DOM subject—and dropping what cannot be named rather than letting the model guess—before generation appeared to reduce hallucination more than instruction wording alone.

Make the reviewer narrow. An early open-ended critique prompt rewrote fluent, correct steps endlessly. Defining `weak` as a closed list of structural defects, biasing toward `ok`, and zeroing the temperature made the loop stable and cheap.

Fail toward the draft. Treating every quality stage as optional—any failure preserves the best result so far—mattered more to our trust in the tool during testing than raw quality: a failed quality stage does not discard the current draft.

Timing is semantics. *When* to capture (`pointerdown` for clicks, `change` for inputs) encodes what a manual reader needs to see; no amount of downstream intelligence recovers a screenshot of the wrong screen.

Related Work

Demonstration-based tutorial generation predates LLMs: Grabler et al. (2009) generated photo-manipulation tutorials from recorded demonstrations in an instrumented application, and Chi et al. (2012) produced mixed-media tutorials from screen recordings, both relying on application-specific instrumentation that the target application must expose. Manual Maker instead targets arbitrary web applications through the DOM accessibility tree, with sentence generation by a commercial LLM; a vision-based layout-parsing enhancement is left to future work.

Our reliance on the DOM connects to a growing body of work on web-interaction data and its limits. Large-scale web-agent corpora such as Mind2Web (Deng et al. 2023) record real action traces across hundreds of live sites, and recent accessibility audits—A11y-CUA (Gubbi Mohanbabu et al. 2026) and the WebAIM Million (WebAIM 2024)—document how frequently production pages leave interactive elements without an accessible name. Those deficits are exactly what force our auto-hide decision: the DOM-blind rates we measure (Evaluation) are a direct consequence of the accessibility gaps these datasets characterize, not an artifact of our pipeline.

Our self-verification loop instantiates iterative self-refinement (Madaan et al. 2023) and verbal self-correction (Shinn et al. 2023) in a setting where hallucination (Ji et al. 2023), not disfluency, is the binding constraint, and where the reviewer is deliberately narrowed to structural defects so the loop converges cheaply rather than re-polishing acceptable prose. The pipeline is built end to end on Upstage's Solar family (Kim et al. 2024; Upstage 2026a), leaning on its structured-output and prompt-caching APIs rather than wrapping a general-purpose model. Commercial screen-capture tools automate image collection but stop short of grounded, verified sentence generation—the step where factual errors actually enter a manual.

Conclusion

Manual Maker turns recorded browser interactions into an editable, multi-format user manual by combining event-timed capture, DOM-grounded subjects that are auto-hidden rather than guessed when the DOM gives no name, schema-constrained drafting with Solar Pro 3, and a bounded self-verification loop that spends model effort only on flagged steps. Each mechanism answers a specific, recurring failure of the naive one-shot approach—screenshots of the wrong screen, invented button names, and steps that silently contradict the action they claim to describe—and the evaluation shows why the grounding decision carries the design: on public traces, proxy missing-name rates reach 25.1% and 68.7%, exactly the pointer targets the shipped build withholds rather than mislabels. An exploratory benchmark reports a model-judged weak-step rate of $14.4\% \rightarrow 5.6\%$ on eight Mind2Web-derived sessions, though Solar acts as its own reviewer there and human validation remains future work.

Packaged for distribution with privacy enforced by architecture and a fail-soft policy that never sacrifices a recording to a failed model call, the extension has a short, well-scoped operational path—API proxying and store review—to full public deployment, followed by the visual-grounding and conversational-retrieval extensions that would close the residual coverage gap. Its design lessons outlast this one application: for any system that turns recorded human behavior into instructions others will trust, ground before you generate, verify narrowly at temperature zero, and never let an optional quality stage lose work that is already good enough to ship.

References

Chi, P.-Y.; Ahn, S.; Ren, A.; Dontcheva, M.; Li, W.; and Hartmann, B. 2012. MixT: Automatic Generation of Step-by-Step Mixed Media Tutorials. In *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology*, UIST '12, 93–102. New York, NY, USA: Association for Computing Machinery.

Deng, X.; Gu, Y.; Zheng, B.; Chen, S.; Stevens, S.; Wang, B.; Sun, H.; and Su, Y. 2023. Mind2Web: Towards a Generalist Agent for the Web. In *Advances in Neural Information Processing Systems*, volume 36.

Grabler, F.; Agrawala, M.; Li, W.; Dontcheva, M.; and Igarashi, T. 2009. Generating Photo Manipulation Tutorials by Demonstration. *ACM Transactions on Graphics*, 28(3): 66:1–66:9.

Gubbi Mohanbabu, A.; Natalie, R.; Kim, B.; Guo, A.; and Pavel, A. 2026. A11y-CUA Dataset: Characterizing the Accessibility Gap in Computer Use Agents. In *Proceedings of the 2026 CHI Conference on Human Factors in Computing Systems*, 1–26. Reduced dataset: https://huggingface.co/

datasets/berkeley-hci/Reduced-A11y-CUA; accessed July 19, 2026.

Ji, Z.; Lee, N.; Frieske, R.; Yu, T.; Su, D.; Xu, Y.; Ishii, E.; Bang, Y. J.; Madotto, A.; and Fung, P. 2023. Survey of Hallucination in Natural Language Generation. *ACM Computing Surveys*, 55(12): 1–38.

Kim, S.; Kim, D.; Park, C.; Lee, W.; Song, W.; Kim, Y.; Kim, H.; Kim, Y.; Lee, H.; Kim, J.; Ahn, C.; Yang, S.; Lee, S.; Park, H.; Gim, G.; Cha, M.; Lee, H.; and Kim, S. 2024. SOLAR 10.7B: Scaling Large Language Models with Simple yet Effective Depth Up-Scaling. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 6: Industry Track)*, 23–35. Mexico City, Mexico: Association for Computational Linguistics.

Madaan, A.; Tandon, N.; Gupta, P.; Hallinan, S.; Gao, L.; Wiegreffe, S.; Alon, U.; Dziri, N.; Prabhumoye, S.; Yang, Y.; Gupta, S.; Majumder, B. P.; Hermann, K.; Welleck, S.; Yazdanbakhsh, A.; and Clark, P. 2023. Self-Refine: Iterative Refinement with Self-Feedback. In *Advances in Neural Information Processing Systems*, volume 36, 46534–46594.

Shinn, N.; Cassano, F.; Gopinath, A.; Narasimhan, K.; and Yao, S. 2023. Reflexion: Language Agents with Verbal Reinforcement Learning. In *Advances in Neural Information Processing Systems*, volume 36, 8634–8652.

Upstage. 2026a. Solar Pro 3: 2× Agentic Performance—What Changed. https://www.upstage.ai/blog/en/solar-pro-3-0323. Published March 2026; accessed July 19, 2026.

Upstage. 2026b. Upstage Document Parse. https://www.upstage.ai/products/document-parse. Product page; accessed July 19, 2026.

WebAIM. 2024. The WebAIM Million: The 2024 Report on the Accessibility of the Top 1,000,000 Home Pages. https://webaim.org/projects/million/2024/. Accessed July 19, 2026.

Appendix A: DOM-Blind Rate Detail

We apply a deterministic, dataset-adapted name proxy to recorded target attributes (supplementary scripts); it does not reproduce a live browser accessibility tree. Of 790 A11y-CUA sighted-user (SU-group) pointer clicks (Gubbi Mohanbabu et al. 2026), 25.1% (198) lack a proxy name. Of 645 Mind2Web clicks (Deng et al. 2023), 68.7% (443) lack one, yielding 31.3% proxy subject coverage and 42.8% manual retention (type/select steps remain eligible regardless of name).

Per-website spread. The Mind2Web rate is not uniform. Across the ten most-represented sites in the shard the DOM-blind share ranges from 13.0% (kayak) to 90.0% (booking), with e-commerce and booking flows (kohls 88.2%, ticketcenter 78.4%, parking 75.0%) dominated by icon and custom-widget targets. The sample of blind targets is consistent: bare `DIV`/`SPAN` overlays, class-only chevron icons, and card-link buttons with no accessible name. These are exactly the targets the shipped build auto-hides rather than mislabels, and the coverage a future visual-grounding integration would address.

Appendix B: Solar Pro 3 on Real Recordings

Our harness drives the *real* Solar Pro 3 API with the extension's endpoint, model, decoding profiles, and prompt-cache mechanism over 8 recordings sampled from Mind2Web (Deng et al. 2023) traces (seed `20260727`). Digests come from annotated click, type, and select actions; clicks lacking a proxy name are excluded to approximate auto-hide. Single-step schemas and calls characterize configuration behavior, not batched production latency. We compare **D-min** (draft only) with **DA-min** (one review-and-refine round); Table 2 reports the result.

Config	Calls	$\text{Weak}_{\text{eval}}$	$\text{Weak}_{\text{final}}$	Target match
D-min	4.4	—	—	35.4%
DA-min	12.1	14.4%	5.6%	35.4%

Table 2: Solar Pro 3 on Mind2Web-derived recordings (8 sessions, 35 eligible steps, seed `20260727`). Values are unweighted session means unless noted. $\text{Weak}_{\text{eval}}$/ $\text{Weak}_{\text{final}}$: model-judged weak-step rate before/after one refine round (pooled: $5/35 \rightarrow 2/35$). Target match: fraction of eligible non-input steps whose target string appears verbatim in the generated title or body (session mean; pooled $9/25$). Single-run estimate; model outputs are nondeterministic. Scripts in the supplementary material.

The Solar model-judged weak-step rate falls from 14.4% to 5.6% after refinement (61% relative; pooled $5/35 \rightarrow 2/35$), at $\sim 2.8\times$ API calls; DA-min target match rises from 32.3% to 35.4% (session means; pooled $8/25 \rightarrow 9/25$). The TypeScript fallback harness reports 0 violations on synthetic fixtures; the Mind2Web proxy check reports $0/320$. These results are exploratory ($n = 8$): traces are English/Western-web oriented, names are proxies, Solar is its own reviewer, and the benchmark uses compact single-step prompts rather than the extension's batched production path.

Appendix C: Deployed Artifact

https://chromewebstore.google.com/detail/manual-maker/fgkbklaijebjeadmgbmopmdndkfllaoa