WEBPII: Benchmarking Visual PII Detection for Computer-Use Agents

Nathan Zhao [1]

Abstract

Computer use agents create new privacy risks: training data collected from real websites inevitably contains sensitive information, and cloud-hosted inference exposes user screenshots. Detecting personally identifiable information in web screenshots is critical for privacy-preserving deployment, but no public benchmark exists for this task. We introduce WEBPII, a fine-grained synthetic benchmark of 44,865 annotated e-commerce UI images designed with three key properties: extended PII taxonomy including transaction-level identifiers that enable re-identification, anticipatory detection for partially-filled forms where users are actively entering data, and scalable generation through VLM-based UI reproduction. Experiments validate that these design choices improve layout-invariant detection across diverse interfaces and generalization to held-out page types. We train WEBREDACT to demonstrate practical utility, more than doubling text-extraction baseline accuracy (0.753 vs 0.357 mAP@50) at real-time CPU latency (20ms). We release the dataset and trained model at webpii.github.io to support privacy-preserving computer use research.

1. Introduction

Computer use agents—language models that operate graphical user interfaces through vision and action—represent a significant capability advance toward general-purpose AI assistants. Unlike traditional web automation that operates on structured HTML or APIs, vision-based systems observe rendered web pages as images and produce mouse and keyboard actions to accomplish user goals. Recent systems such as Claude Computer Use (Anthropic, 2024) and Gemini 2.5 (Comanici et al., 2025) demonstrate purely vision-driven agents that can book flights, complete checkout flows, navigate e-commerce sites, and manage user accounts across arbitrary websites without access to DOM structure. As these systems scale from research prototypes to production deployments serving millions of sessions, their visual-first architecture creates fundamental privacy problems: every screenshot observation contains rendered PII, and standard cloud-hosted inference exposes sensitive user data during routine operation.

The privacy challenges span both training and inference. Training data collected from real websites inevitably contains PII that models memorize and leak (Lukas et al., 2023; Nasr et al., 2023), while cloud-hosted inference routinely exposes user screenshots. Existing mitigations are insufficient: sandboxed benchmarks (Zhou et al., 2023; Xie et al., 2024) use fabricated data that does not transfer to real sessions, crowdsourced datasets (Deng et al., 2023; Lù et al., 2024) lack real-time authenticated content, agentic pipelines (Wang et al., 2025d) lack visual PII detection, and federated approaches (Wang et al., 2025c;b) still leak information through gradient updates.

Critically, **no public benchmark exists for visual PII detection in web interfaces**. Text-based PII systems (Microsoft, 2024b; ai4Privacy, 2023; Selvam & Ghosh, 2025) operate on extracted strings, missing rendered content where sensitivity derives from visual context rather than surrounding words. Document-focused datasets (Bulatov et al., 2021; Lerouge et al., 2024) target fixed-layout identity documents with predictable field positions. Scene text datasets (Veit et al., 2016; Gupta et al., 2016) localize text without distinguishing sensitive content from UI chrome.

This paper presents a dataset-first contribution to address this gap. We introduce WEBPII, a fine-grained synthetic benchmark of e-commerce interface images designed with properties essential for visual PII detection in web interfaces. Our experiments validate key design choices—fill state diversity, data injection density, extended identifier taxonomy—demonstrating that the dataset enables layout-invariant PII detection across diverse e-commerce layouts.

Figure 1 illustrates the diversity of PII contexts across e-commerce layouts that we can dynamically vary through data injection. Beyond this visual diversity, the dataset introduces three validated properties:

[1]Department of Computer Science, Stanford University, Stanford, CA, USA. Correspondence to: Nathan Zhao <nathanzh@stanford.edu>.

Published at the Second Workshop on Agents in the Wild: Safety, Security, and Beyond (AIWILD) at ICML 2026. Copyright 2026 by the author(s).

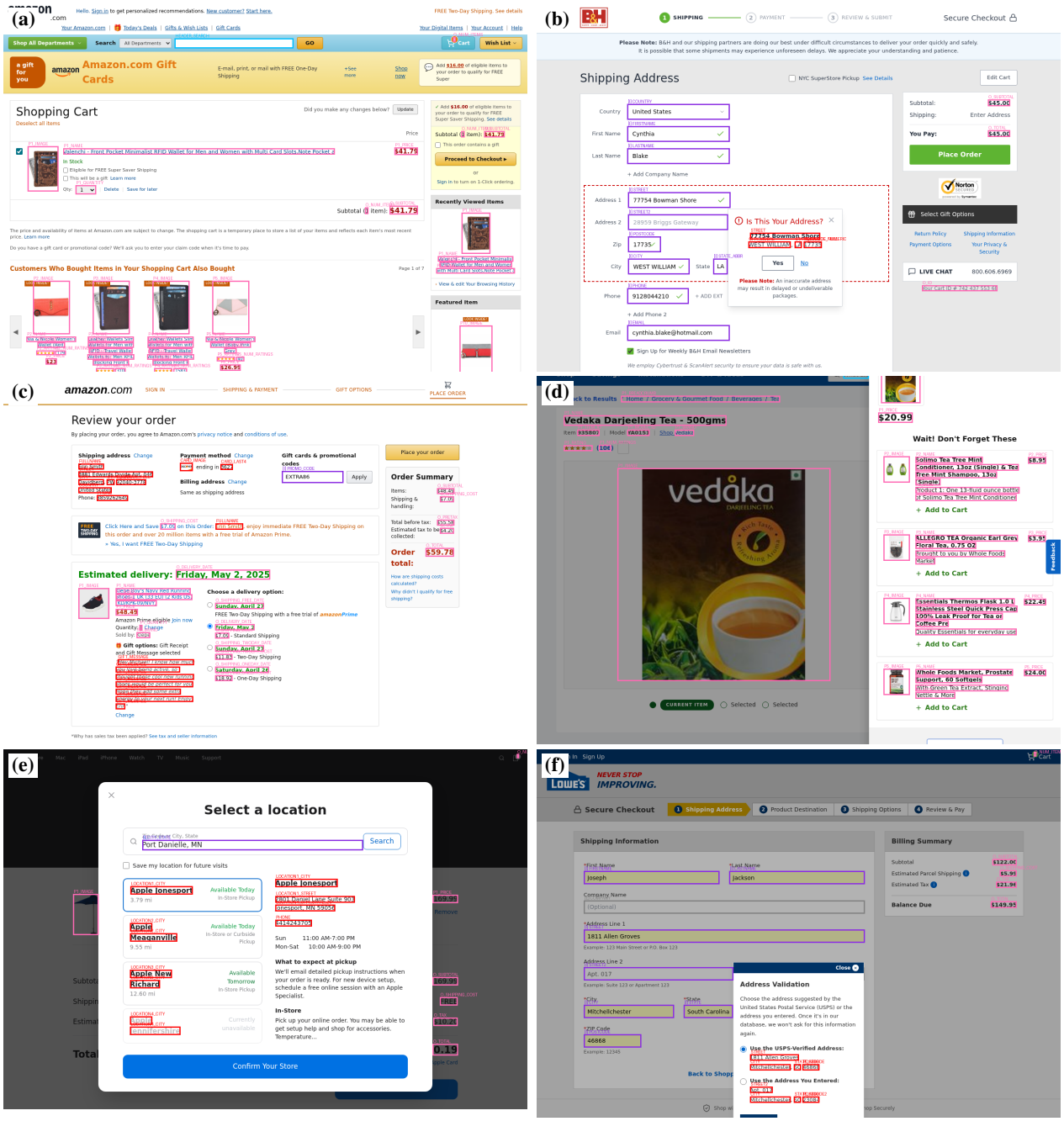


Figure 1. Sample images from WEBPII, rendered with different injected data. The dataset captures the visual complexity of e-commerce interfaces: variable page heights reflecting diverse checkout flows and product displays (compare compact cart in (a) with extended layout in (d)), input fields and dropdown selectors, modal overlays with backdrops that occlude underlying content, ad hoc identifying information such as gift messages (c) and proposed pickup locations (e), and derived values requiring computation of taxes and totals. Bounding boxes respect occlusion boundaries. Pink indicates product annotations, purple denotes empty input fields, and red identifies PII.

1. **Extended PII taxonomy.** Beyond traditional PII, WEBPII annotates *extended identifiers*—order numbers, tracking IDs, delivery dates, purchase histories—that enable re-identification but fall outside conventional PII definitions.

2. **Anticipatory detection.** Web interfaces expose PII as it is being typed. WEBPII generates annotations at each progressive state of form-filling, enabling models to trigger redaction proactively rather than after sensitive data is fully visible.

3. **Scalable generation pipeline.** Our VLM-based approach dynamically generates diverse layouts and injects varied PII configurations, producing pixel-level bounding boxes without manual annotation.

Our generation pipeline exploits VLM-based UI reproduction (Anthropic, 2025a; SST, 2025) to generate functional frontend code from screenshots, embedding annotation attributes during code generation for programmatic bounding box extraction. Section 2.1 documents the PII taxonomy underlying our annotations.

We validate the dataset's utility by training **WEBREDACT**, a visual detection model that more than doubles text-extraction baseline accuracy (0.753 vs. 0.357 mAP@50) at real-time CPU latency (20ms). Section 3 presents ablation studies validating our design choices.

We release WEBPII, WEBREDACT, and WEBREDACT-LARGE at webpii.github.io to support development of privacy-preserving computer use systems.

2. The WEBPII Dataset

E-commerce interfaces present PII challenges distinct from documents or scene text. While an email address in a scanned form appears as static pixels, the same email in a web UI may be rendered through JavaScript, styled with CSS, and wrapped in interactive elements. Moreover, web forms require *anticipatory detection*—identifying sensitive fields before users finish typing, as privacy interventions should trigger during entry rather than after completion. Beyond traditional PII, these interfaces expose *extended identifiers*—order IDs, tracking numbers, delivery dates—that enable reidentification despite not constituting traditional PII. Our benchmark must capture this complexity.

Generating annotated PII data at scale presents a dilemma: real screenshots contain real PII requiring manual annotation, while synthetic generation risks unrealistic layouts. We resolve this through *reproduction with annotation injection*: collecting real e-commerce screenshots, then using vision-language models to recreate them as functional code with annotations embedded during generation. This produces pixel-accurate bounding boxes without manual labeling.

2.1. Extended Identifiers in E-commerce

Beyond traditional PII (names, addresses, payment details), e-commerce interfaces display transaction-level attributes—order dates, merchants, item quantities, financial totals, delivery information—that enable reidentification despite not constituting traditional PII. Four credit card transactions containing just the merchant and date reidentify 90% of 1.1M users (de Montjoye et al., 2015); eight movie ratings with dates reidentify 99% of Netflix users (Narayanan & Shmatikov, 2008). Cross-platform attacks link accounts by matching transaction patterns (Archie et al., 2018).

These identifiers also reveal sensitive personal attributes—purchase patterns predict health conditions (Aiello et al., 2019; Sasaya et al., 2026), socioeconomic status (Hashemian et al., 2017), and personal characteristics (Duhigg & Pole, 2012; Kosinski et al., 2013)—informing insurance underwriting (Allen, 2018), data broker segmentation (Callanan et al., 2021), and surveillance (Toksson, 2024; Kerr, 2021; Sobel, 2023).

E-commerce interfaces also expose fields that are directly identifying yet absent from existing PII benchmarks: gift messages contain both sender and recipient names alongside personal notes; B2B purchase order numbers link individuals to corporate procurement systems and company names; delivery security codes enable package retrieval. Visual PII detection for agentic commerce must address these fields alongside traditional PII.

2.2. Data Collection and Generation

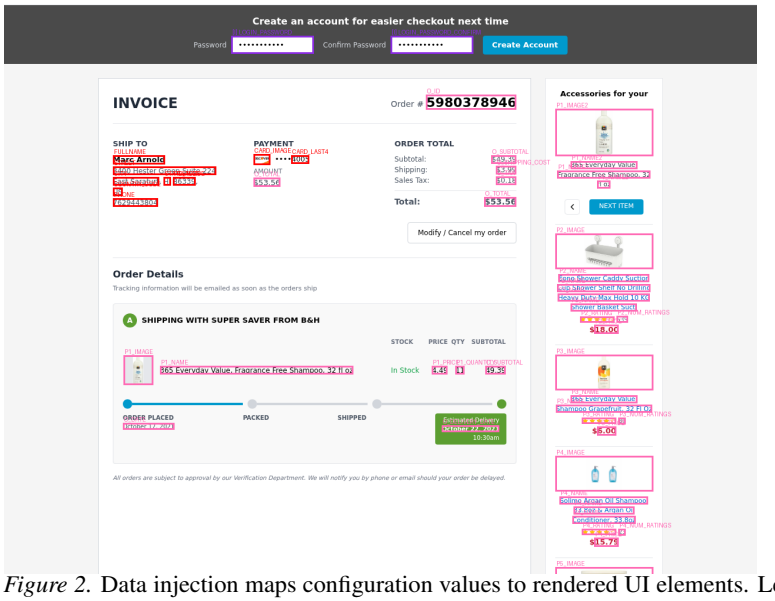
We collected screenshots from 10 e-commerce brands across 19 page types, including cart, checkout, billing, order history, delivery, gifting, store pickup, and product flows. Selection prioritized layouts with modal overlays, dynamic sidebars, embedded maps, cluttered multi-step forms, and responsive behavior. This process yielded 408 unique source layouts, which serve only as visual targets for reproduction rather than as dataset images.

For generation, we evaluated GPT-5.2 (OpenAI, 2025) with OpenCode (SST, 2025) and Claude Opus 4.5 (Anthropic, 2025b) with Claude Code (Anthropic, 2025a); Claude more reliably followed annotation constraints, so we use it throughout. The model reconstructs each screenshot as a React component with two enforced requirements: sensitive content must reference data variables rather than hardcoded values, and each annotatable element must expose a data attribute for extraction. A Vite (You, 2024) server renders the component, while a Playwright (Microsoft, 2024a) harness captures screenshots and extracts bounding boxes from the rendered DOM. Additional pipeline details appear in Appendix B.

Generation remains semi-automatic rather than push-button. We decompose reproduction into structure generation, attribute marking, input-state handling, and visual refinement, with additional pipeline details reported in Appendix B.

2.2.1. QUALITY CONTROL AND HUMAN REVIEW

Quality control is central to the dataset because mistakes in a generated page structure would be amplified across many rendered examples. We built a Flask review interface for comparing each source screenshot against its generated lay-



Configuration Key	Value
Injected (Faker)	
PII_FULLNAME	Marc Arnold
PII_STREET	3400 Hester Green Suite 224
ORDER_DATE	October 17, 2021
ORDER_DELIVERY_DATE	October 22, 2021
Injected (ABO)	
PRODUCT1_NAME	365 Everyday Value, Fragra...
PRODUCT1_IMAGE	(product image)
LLM-Extracted	
SHIPPING_COST	5.99
PRODUCT1_BRAND	365 Everyday Value
Randomized	
PRODUCT1_PRICE	4.49
PRODUCT1_RATING	4.7
Derived at Render Time	
ORDER_ID	from SEED (647926)
ORDER_TOTAL	subtotal + ship + tax

Figure 2. Data injection maps configuration values to rendered UI elements. Left: annotated screenshot with bounding boxes. Right: selected subset of data injected for this page—Faker-generated PII, ABO product data, LLM-extracted metadata, and values derived at render time. The same layout rendered with different configurations produces diverse training examples with automatic annotations.

out, and reviewers submitted natural-language fixes rather than drawing boxes or editing code. Each layout underwent three review passes.

Figure 3 summarizes the correction workload. The main finding is that refinements were not cosmetic: viewport/layout failures and missing or spurious data attributes dominated, creating errors that would otherwise become false annotations, missing labels, or invalid anticipatory states. These failures directly informed the final prompts and filtering checks.

Because review occurs before variant rendering, one correction applies to all examples produced from that page structure. This kept validation scalable while preserving a human check on errors that automated DOM extraction would otherwise propagate.

The workflow also reduces privacy exposure from real websites. Original screenshots are used only as visual targets for reproduction and are not part of the released benchmark. Sensitive strings visible in those screenshots are replaced by synthetic or derived variables before dataset rendering, while product imagery and metadata come from public product datasets or model-cleaned configurations. In practice, the hardest part was obtaining broad coverage of authenticated checkout, account, and order-history states across retailers. The model-mediated reproduction loop made the remaining step feasible: without automated code editing and rerendering, manually authoring 408 responsive web layouts with annotation instrumentation would have dominated dataset construction.

2.2.2. DATA INJECTION

Each layout is a generated React component encoding one page structure, while a data configuration is one injected set of PII, product, and order values for that structure. The components contain variable references rather than hard-coded values; at render time, data configurations populate these references with values from multiple sources. This enables the same layout to produce diverse screenshots with automatic annotations. We annotate three categories, illustrated in Figure 2: (a) **PII**, covering traditional identifiers and context-specific fields such as delivery instructions and security codes, generated primarily with Faker (Faraglia, 2024) and lightweight templates; (b) **product information**, covering displayed merchandise such as names, descriptions, prices, images, ratings, and quantities, with product images sourced from Amazon Berkeley Objects (Collins et al., 2021) (~147K products, 400K images) and missing metadata cleaned or extracted with GPT-4o-mini (OpenAI, 2024); and (c) **order information**, covering extended identifiers such as order IDs, dates, tracking numbers, and financial totals, with platform-specific identifier formats inferred from the source screenshot and derived values computed at render time.

Additional fields are generated via GPT-4o-mini when building data configuration files to produce realistic content: navigation breadcrumbs referencing browsing history (Figure 1d), gift messages mentioning both recipient names and purchased items (Figure 1c), and promotional copy. Each component C_i is rendered with $m = 25$ distinct data configurations, producing diverse screenshots with varied

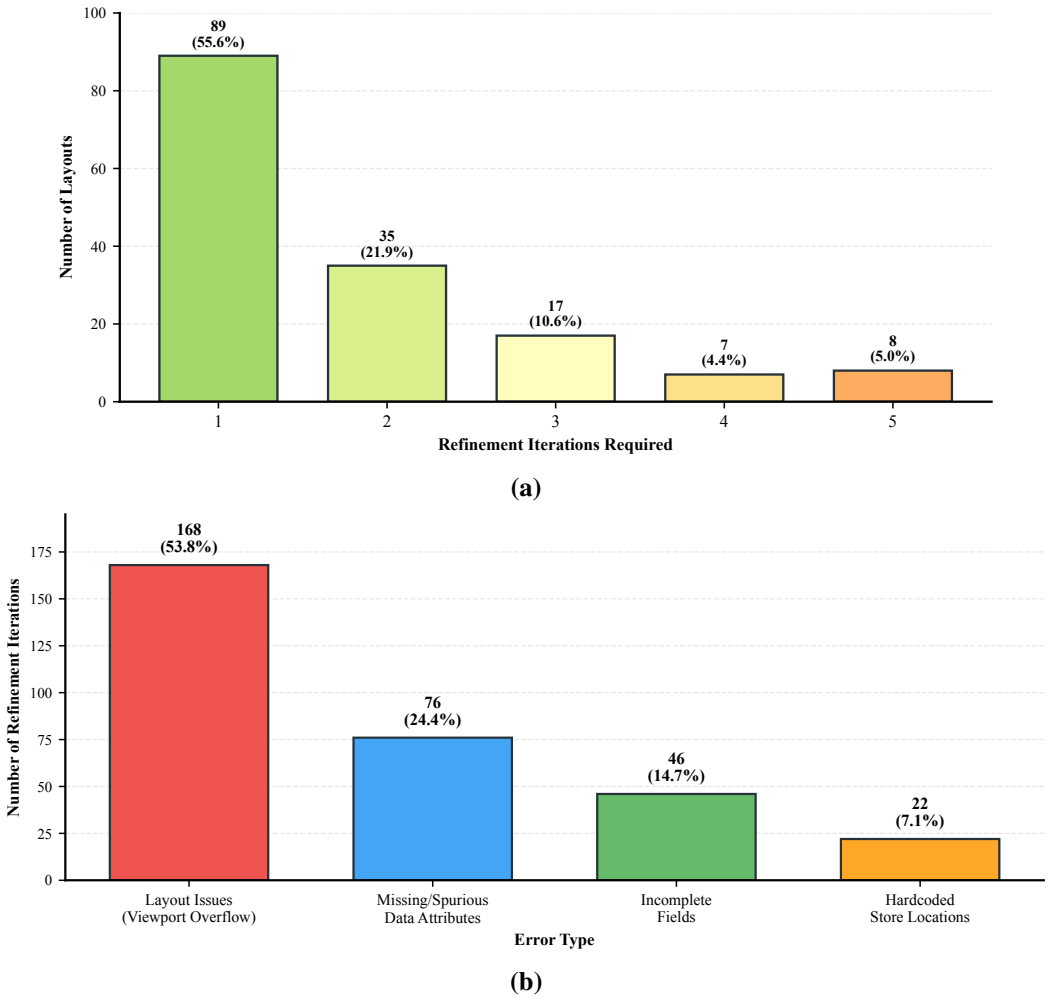


Figure 3. Human review statistics for the generation pipeline. (a) Distribution of correction iterations among the 160 layouts that required refinement; most corrected layouts converged after one or two iterations. (b) Breakdown of all 312 refinement iterations by error type, showing that viewport/layout errors and data-attribute issues dominated the correction workload.

PII values, product images, and transaction details while maintaining automatic annotation alignment through DOM queries.

2.3. Annotation Extraction

By requiring data attributes during generation, we extract pixel-coordinate bounding boxes directly from the rendered DOM using the browser's Range API to match text content with PII values. Wrapped text is split into separate bounding boxes per line via vertical overlap detection. This approach adapts to responsive layouts and ensures derived values receive proper attribution. Our extraction pipeline performs visibility analysis, excluding fully occluded elements and clipping partially visible ones to their visible extent.

2.4. Anticipatory Detection

A key differentiator from text-based PII datasets is our support for *anticipatory detection*—identifying PII before the user has finished typing. In deployed systems, privacy-preserving interventions should trigger as sensitive data is being entered, not after completion. For each component C_i containing N_i form fields, we generate anticipatory variants across progressive fill states: for stage k (where $1 \leq k < N_i$), fields 1 through $k-1$ are fully filled, field k is partially filled (mid-typing), and fields $k+1$ through N_i remain empty. Figure 4 illustrates two stages of this progression.

This systematic progression mirrors actual user interaction and ensures balanced representation across fill states. Models observe each field in three contexts: empty (before the user reaches it), partially filled (active input), and fully filled (after completion). Rather than biasing towards any particu-

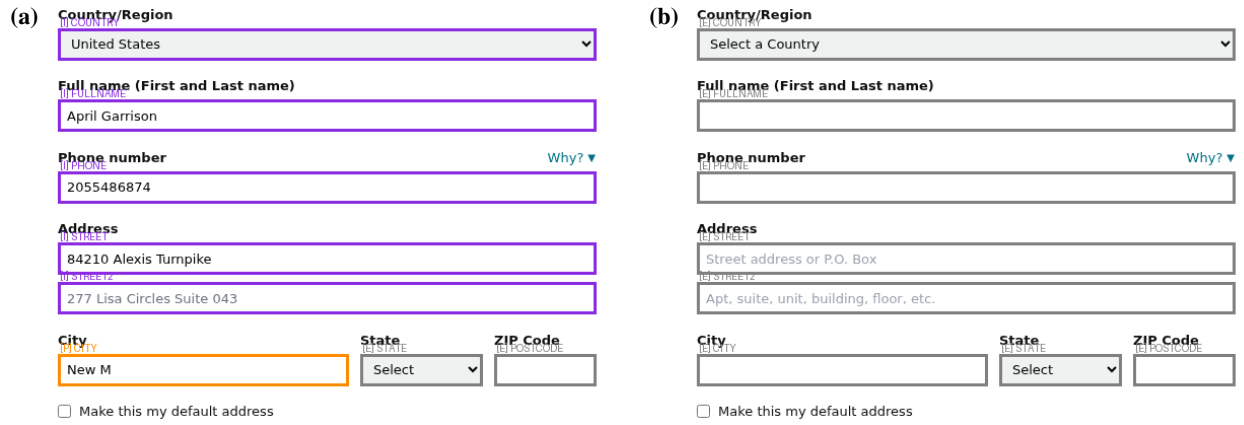


Figure 4. Form fill states for anticipatory detection. (a) Partial: mid-entry state with later fields incomplete (city field shows "New M" mid-typing). (b) Empty: pristine form with placeholder text and input field annotations. Yellow indicates partially filled fields; grey denotes empty fields.

lar fill state, the dataset provides equal exposure to all stages of form completion. Additionally, forms with optional fields are rendered with these fields randomly included or excluded across data injections, exposing models to structural variability in form layouts.

2.5. Summary and Comparison

WEBPII comprises 44,865 images spanning 10 e-commerce websites and 19 distinct page types, with 993,461 visible bounding-box annotations. This includes 10,200 base images with fully-filled forms, augmented by our anticipatory detection methodology that generates 28,653 partial-fill variants and 6,012 empty-form variants. Annotations span three categories (PII, product information, and order identifiers) across diverse HTML contexts (rendered text, input fields, and images). Figure 5 summarizes the core dataset composition statistics and compares WEBPII against prior benchmarks, while Appendix A provides additional company- and page-type breakdowns.

3. Experiments

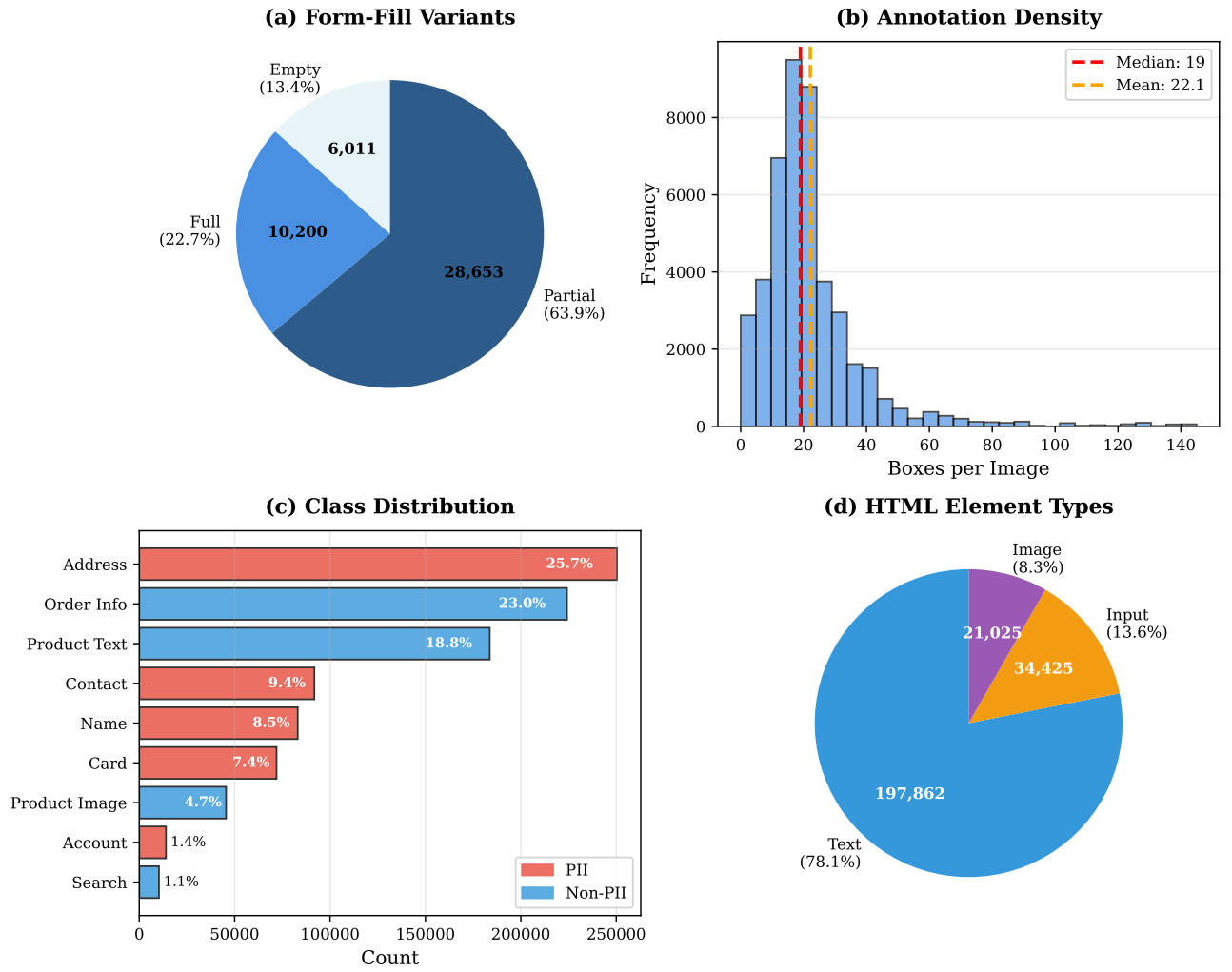
3.1. Experimental Setup

The diversity of WEBPII enables evaluation of generalization at different levels. We render each of the 408 unique layouts with 25 data injection variants (different PII values, addresses, and product information), and for layouts with input fields, generate the fill states described in Section 2.4: *full*, *partial*, and *empty*. We evaluate three split strategies: $\textbf{Test}_{\textbf{Cross-Page}}$ holds out 20% of layouts randomly (82 layouts, 298 fill states), testing whether models learn layout-invariant features within a company's design system. $\textbf{Test}_{\textbf{Cross-Company}}$ holds out all Amazon layouts (56 layouts, 152 fill states) while training on 9 other companies (352 layouts, 1,416 fill states), evaluating generalization to entirely new visual styles and brand identities. $\textbf{Test}_{\textbf{Cross-Type}}$ holds out all receipt pages (20 layouts, 50 fill states) while training on 18 other page types (388 layouts, 1,518 fill states), measuring whether detection strategies transfer across functionally different page categories with distinct UI patterns. For all splits, we ensure no data leakage: the specific PII values, addresses, and product information in test images never appear in training.

3.2. Baseline Methods

3.2.1. TEXT-BASED METHODS

We evaluate text-based baselines using a two-stage pipeline: (1) OCR extraction to obtain text spans with bounding boxes, (2) classification to identify sensitive content. This approach mirrors existing PII detection systems that operate on extracted text rather than raw pixels. We compare three approaches: Presidio (Microsoft, 2024b), an NER-based system using pattern matching and named entity recognition; GPT-4o-mini for LLM-based classification; and document understanding models LayoutLMv3 (Huang et al., 2022) and Donut (Kim et al., 2021) that encode visual layout alongside text. All document understanding pipelines use GPT-4o-mini for final classification to ensure fair comparison. We evaluate multiple OCR engines Tesseract (Smith, 2007), EasyOCR (JaidedAI, 2024), and PaddleOCR (Cui et al., 2025), reporting the highest-accuracy configuration (LayoutLMv3) and fastest configuration (Tesseract + Presidio). Note that text-based approaches can only evaluate text content—product images are excluded as OCR provides no signal for visual elements. Appendix D.1 presents detailed ablations across all OCR engines, language models, and classification approaches.



Dataset	Domain	Size	Visual	Semantic	Web	Ext.	Antic.
AI4Privacy (ai4Privacy, 2023)	Text	300K	✗	✓	✗	✗	✗
PANORAMA (Selvam & Ghosh, 2025)	Text	385K	✗	✓	✗	✗	✗
Nemotron-PII (Steier et al., 2025)	Text	100K	✗	✓	✗	✗	✗
Gretel Finance (Gretel AI, 2024)	Text	56K	✗	✓	✗	✗	✗
BigCode PII (Li et al., 2023)	Code	12K	✗	✓	✗	✗	✗
PIILO (Holmes et al., 2023)	Essays	22K	✗	✓	✗	✗	✗
MIDV-2020 (Bulatov et al., 2021)	ID Docs	72K	✓	✓	✗	✗	✗
DocXPand-25k (Lerouge et al., 2024)	ID Docs	25K	✓	✓	✗	✗	✗
COCO-Text (Veit et al., 2016)	Scene	63K	✓	✗	✗	✗	✗
SynthText (Gupta et al., 2016)	Scene	800K	✓	✗	✗	✗	✗
WEBPII (Ours)	Web UI	44K	✓	✓	✓	✓	✓

Figure 5. Dataset composition and benchmark comparison. Panels (a)–(d) summarize WEBPII: (a) distribution across form-fill variants—empty forms (13.4%), fully-filled forms (22.7%), and partial-fill states (63.9%)—enabling anticipatory detection training; (b) visible annotation density distribution with median 19 boxes per image (mean 22.1), ranging from 0 to 145; (c) breakdown of all 9 semantic annotation classes, where address (25.7%), order info (23.0%), and product text (18.8%) dominate, with PII classes comprising 52.4% of semantic annotations and non-PII classes 47.6%; and (d) HTML element type distribution, showing most annotations target rendered text (78.1%) rather than input fields (13.6%) or images (8.3%). The table above compares WEBPII against prior PII benchmarks and highlights that it is the first to combine visual localization, semantic categories, web interface targeting, extended identifiers, and anticipatory form-state support.

3.2.2. WEBREDACT

We train object detection models on WEBPII as WEBREDACT, targeting real-time inference suitable for live

Table 1. Baseline results on $\text{Test}_{\text{Cross-Company}}$ (Amazon, full-fill images only). OCR+Presidio uses Tesseract; LayoutLMv3 uses GPT-4o-mini for classification.

METHOD	MAP@50	LATENCY
OCR + PRESIDIO	0.183	1.3S
LAYOUTLMV3 + GPT-4O-MINI	0.357	2.9S
WEBREDACT (OURS)	0.753	**20MS**
WEBREDACT-LARGE (OURS)	**0.842**	312MS

redaction. While WEBPII contains fine-grained annotations for different PII types, we train on two classes grouped by visual appearance: `text` (all text-based elements including PII fields, product descriptions, and order information) and `image` (product images). This simplification improves detection reliability by providing more training examples per class and clearer visual distinctions between categories. Training uses 100 epochs, batch size 16, with standard augmentations (random crops, flips, color jittering). We train two model variants: WEBREDACT at 640×640 resolution for real-time CPU inference, and WEBREDACT-LARGE at 1280×1280 resolution for higher accuracy when near-real-time constraints can be relaxed.

3.3. Results

Table 1 presents main results on $\text{Test}_{\text{Cross-Company}}$, evaluating generalization to entirely new visual styles without seeing Amazon's design system during training. We evaluate text-based baselines only on full-filled images, as these approaches cannot identify empty or partially-filled input fields. Even on this favorable subset, both WEBREDACT variants substantially outperform text-based methods: WEBREDACT achieves 0.753 mAP@50, more than double the best text-based baseline (LayoutLMv3 at 0.357), while WEBREDACT-LARGE reaches 0.842 mAP@50. Detailed failure mode analysis for OCR+LLM systems appears in Appendix D.1.

WEBREDACT processes images in ∼20ms on mid-range consumer CPUs (Intel i5/AMD Ryzen 5), meeting real-time constraints for 30fps redaction, while WEBREDACT-LARGE requires ∼312ms (∼3fps). Both models use OpenVINO for CPU inference. Text-based methods are substantially slower: Tesseract (Smith, 2007) OCR extraction alone (453ms, excluding classification) is slower than WEBREDACT-LARGE's full detection pipeline.

3.4. Dataset Ablations

We conduct ablations to validate WEBPII's design choices; full tables appear in Appendix D.

Split strategies. We compare three split strategies (Table 12). $\text{Test}_{\text{Cross-Page}}$ achieves 0.797 mAP@50, indicating models learn layout-invariant features within a company's design system. $\text{Test}_{\text{Cross-Company}}$ degrades to 0.753 when generalizing to Amazon's distinct visual style. $\text{Test}_{\text{Cross-Type}}$ shows the largest degradation (0.728), revealing that page-type-specific patterns transfer less effectively than company-specific design conventions.

Fill state diversity. Training on full screenshots alone achieves 0.771 mAP@50 (Table 13). Adding empty screenshots *without* partials degrades performance to 0.758, while full+partial achieves 0.797 (+0.026). Combining all three states achieves 0.825 mAP@50, demonstrating that partial fills provide essential intermediate visual grounding.

Progressive fill density. Increasing partial-fill stages per layout from 1 to 5 improves performance from 0.758 to 0.802 mAP@50 (Table 14), with the strongest gains on partial-fill test images (0.774 to 0.835). Each additional stage exposes the model to new intermediate form states, providing non-redundant learning signal for field detection across the full spectrum of user interaction.

Text variant density. Increasing text variants per layout from 1 to 25 improves performance from 0.795 to 0.820 mAP@50 (Table 15). Both precision and recall increase with more variants (0.805 to 0.842 and 0.713 to 0.731 respectively), confirming that diverse data injections—varying names, addresses, and products—improve generalization beyond layout diversity alone.

4. Discussion

4.1. Why CPU-First Detection

Our emphasis on lightweight CPU-executable models reflects deployment constraints for privacy-sensitive applications: on-device processing keeps screen content local rather than transmitting to cloud APIs (Zhou et al., 2019), continuous detection must not overload local compute, and high-volume scenarios (browser extensions, OS-level privacy layers) make per-inference GPU costs prohibitive. This aligns with broader trends toward on-device computer use agents (Awadallah et al., 2025; Wang et al., 2025a; Lin et al., 2024), where privacy-preserving detection must operate within the same resource envelope.

4.2. Limitations and Future Work

Several limitations suggest directions for future work: domain scope is restricted to English-language e-commerce interfaces, with extension to other languages and domains (banking, healthcare, social media) requiring additional data collection; static image annotation does not capture challenges present in video streams with scrolling and state transitions; and our PII taxonomy conservatively labels all products and order identifiers as potentially identifying, which

may cause over-redaction without contextual understanding.

Beyond privacy protection, WEBPII enables applications in computer use agent systems. PII, product, and input region detection provides semantic grounding for agent inference—extending GUI grounding approaches (Lu et al., 2024; Cheng et al., 2024; Gou et al., 2024; Yang et al., 2023; Awadalla et al., 2025; Feizi et al., 2025) to allow agents to condition behavior on field sensitivity. The synthetic generation pipeline's controllability could enable RL environment construction for agent training, where programmatic trajectory specification (Pahuja et al., 2025; Wang et al., 2025e) enables scalable production of training data compared to expensive manual demonstration (Deng et al., 2023; Lù et al., 2024).

4.3. Data Governance and Release

The benchmark is designed to support privacy research without redistributing real user data. Original website captures are used only as private visual targets for reproduction; released images are rendered from generated components with synthetic PII, public product imagery, and derived order metadata. This separation is important for agentic settings because real checkout and account pages often contain sensitive combinations of names, addresses, orders, delivery details, and payment context. Reproduction lets us preserve the visual and interaction patterns that matter for detection while replacing the underlying identities.

We release the dataset, trained detectors, and generation metadata needed to audit benchmark construction at Hugging Face and GitHub. The released assets include annotation schemas, split definitions, and model checkpoints so that future work can compare against the same cross-page, cross-company, and cross-type generalization settings. We do not release original screenshots or any private account state used to create visual targets.

This governance model also clarifies the role of the detector. WEBREDACT is not intended to decide whether a workflow is legally permitted to process user data; it provides a local perceptual layer that can identify sensitive visual regions before screenshots leave the device or enter a larger agent pipeline. Downstream systems can then choose stricter policies such as redaction, selective disclosure, confirmation prompts, or local-only processing for sensitive fields.

5. Conclusion

We introduced WEBPII, a fine-grained synthetic benchmark for visual PII detection in web interfaces, designed with three validated properties: extended PII taxonomy including transaction-level identifiers, anticipatory detection for partially-filled forms, and scalable generation through VLM-based UI reproduction. Our generation pipeline produces annotated data at scale by instrumenting LLM-driven UI reproduction with programmatic annotation extraction, requiring human validation for 39% of layouts. Ablation studies validate these design choices, confirming that fill state diversity and data injection density each provide significant gains across split strategies.

We trained WEBREDACT to validate practical utility: the model more than doubles text-extraction baseline accuracy (0.753 vs 0.357 mAP@50) while achieving real-time CPU inference (20ms). As computer use agents transition from research prototypes to deployed systems, WEBPII provides a foundation for integrating privacy protection into their design. We release the dataset and model at webpii.github.io to support privacy-preserving computer use research.

Impact Statement

This work aims to improve privacy in computer use systems by enabling automated detection and redaction of sensitive information in web interfaces. While PII detection models could theoretically be misused for surveillance, actors with such intent can already extract PII through existing methods (foundation models, offline OCR) that do not require real-time visual localization. The availability of privacy-protective tools for the broader research community represents a net positive for user privacy.

We also recognize that redaction systems can create secondary risks if treated as complete privacy solutions. A missed detection may create false confidence, while overly broad detection may interfere with accessibility tools, fraud prevention, customer support, or user-authorized automation. For this reason, WEBPII should be viewed as an evaluation resource and a building block for layered defenses rather than a standalone compliance mechanism. Deployments should combine local visual detection with user consent, policy controls, and clear failure handling.

The dataset construction process is designed to minimize direct exposure of private information: released images are synthetic renderings, original website captures are not redistributed, and the benchmark emphasizes local detection before screenshots are sent to remote services. We expect the primary beneficial use to be privacy-preserving computer-use agents, browser extensions, operating-system privacy layers, and research on agent safety under realistic web-interface conditions.

References

ai4Privacy. pii-masking-43k (Revision c47c98d), 2023. URL https://huggingface.co/datasets/ai4privacy/pii-masking-43k.

Aiello, L., Schifanella, R., Quercia, D., and Prete, L. D. Large-scale and high-resolution analysis of food purchases and health outcomes. *EPJ Data Science*, 8, 2019.

Allen, M. Health insurers are vacuuming up details about you — and it could raise your rates. ProPublica, July 2018. URL https://www.propublica.org/article/health-insurers-are-vacuuming-up-details-about-you-and-it-could-raise-your-rates.

Anthropic. Claude 3 Model Card Addendum: October 2024, 10 2024. URL https://assets.anthropic.com/m/1cd9d098ac3e6467/original/Claude-3-Model-Card-October-Addendum.pdf. Introducing computer use capability in Claude 3.5 Sonnet.

Anthropic. Claude Code, 2025a. URL https://github.com/anthropics/claude-code.

Anthropic. System Card: Claude Opus 4.5. Technical report, Anthropic, November 2025b. URL https://assets.anthropic.com/m/64823ba7485345a7/Claude-Opus-4-5-System-Card.pdf.

Archie, M., Gershon, S., Katcoff, A., and Zeng, A. Who ' s watching ? de-anonymization of netflix reviews using amazon reviews, 2018.

Awadalla, A., Ghosh, D., Akkus, A., Zhang, Y., Nezhurina, M., Jitsev, J., Choi, Y., and Schmidt, L. Gelato — From Data Curation to Reinforcement Learning: Building a Strong Grounding Model for Computer-Use Agents. https://github.com/mlfoundations/gelato, 2025.

Awadallah, A., Lara, Y., Magazine, R., Mozannar, H., Nambi, A., Pandya, Y., Rajeswaran, A., Rosset, C., Taymanov, A., Vineet, V., Whitehead, S., and Zhao, A. Fara-7B: An Efficient Agentic Model for Computer Use, 2025.

Bulatov, K., Emelianova, E., Tropin, D., Skoryukina, N., Chernyshova, Y., Sheshkus, A., Usilin, S., Ming, Z., Burie, J., Luqman, M., and Arlazarov, V. MIDV-2020: A Comprehensive Benchmark Dataset for Identity Document Analysis. *ArXiv*, abs/2107.00396, 2021.

Callanan, G., Perri, D. F., and Tomkowicz, S. M. Targeting vulnerable populations: The ethical implications of data mining, automated prediction, and focused marketing. *Business and Society Review*, 2021.

Cheng, K., Sun, Q., Chu, Y., Xu, F., Li, Y., Zhang, J., and Wu, Z. SeeClick: Harnessing GUI Grounding for Advanced Visual GUI Agents. In *Annual Meeting of the Association for Computational Linguistics*, pp. 9313–9332, 2024.

Collins, J., Goel, S., Luthra, A., Xu, L. L., Deng, K., Zhang, X., Vicente, T. F. Y., Arora, H., Dideriksen, T., Guillaumin, M., and Malik, J. ABO: Dataset and Benchmarks for Real-World 3D Object Understanding. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 21094–21104, 2021.

Comanici, G., Bieber, E., Schaekermann, M., Pasupat, I., Sachdeva, N., Dhillon, I., Blistein, M., Ram, O., Zhang, D., Rosen, E., Marris, L., Petulla, S., Gaffney, C., Aharoni, A., Lintz, N., Pais, T. C., Jacobsson, H., Szpektor, I., Jiang, N.-J., Haridasan, K., Omran, A., Saunshi, N., Bahri, D., Mishra, G., Chu, E., et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *ArXiv*, abs/2507.06261, 2025.

Cui, C., Sun, T., Lin, M., Gao, T., Zhang, Y., Liu, J., Wang, X., Zhang, Z., Zhou, C., Liu, H., Zhang, Y., Lv, W., Huang, K., Zhang, Y., Zhang, J., Zhang, J., Liu, Y., Yu, D., and Ma, Y. PaddleOCR 3.0 Technical Report, 2025. URL https://arxiv.org/abs/2507.05595.

de Montjoye, Y.-A., Radaelli, L., Singh, V. K., and Pentland, A. Unique in the Shopping Mall: On the Reidentifiability of Credit Card Metadata. *Science*, 347(6221):536–539, 2015.

Deng, X., Gu, Y., Zheng, B., Chen, S., Stevens, S., Wang, B., Sun, H., and Su, Y. Mind2Web: Towards a Generalist Agent for the Web. *ArXiv*, abs/2306.06070, 2023.

Duhigg, C. and Pole, A. How companies learn your secrets, 2012.

Faraglia, D. Faker: A Python Package That Generates Fake Data, 2024. URL https://github.com/joke2k/faker.

Feizi, A., Nayak, S., Jian, X., Lin, K. Q., Li, K., Awal, R., Lù, X. H., Obando-Ceron, J., Rodriguez, J. A., Chapados, N., Vazquez, D., Romero-Soriano, A., Rabbany, R., Taslakian, P., Pal, C., Gella, S., and Rajeswar, S. Grounding Computer Use Agents on Human Demonstrations, 2025. URL https://arxiv.org/abs/2511.07332.

Gou, B., Wang, R., Zheng, B., Xie, Y., Chang, C., Shu, Y., Sun, H., and Su, Y. Navigating the Digital World as Humans Do: Universal Visual Grounding for GUI Agents. *ArXiv*, abs/2410.05243, 2024.

Gretel AI. GLiNER Models for PII Detection through Fine-Tuning on Gretel-Generated Synthetic Documents, 10 2024.

Gupta, A., Vedaldi, A., and Zisserman, A. Synthetic Data for Text Localisation in Natural Images. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2315–2324, 2016.

Hashemian, B., Massaro, E., Bojic, I., Arias, J. M., Sobolevsky, S., and Ratti, C. Socioeconomic characterization of regions through the lens of individual financial transactions. *PLoS ONE*, 12, 2017.

Holmes, L., Crossley, S., Sikka, H., and Morris, W. PII-ILO: An Open-Source System for Personally Identifiable Information Labeling and Obfuscation. *Information and Learning Sciences*, 124(9-10):266–284, 2023. doi: 10.1108/ILS-04-2023-0032.

Huang, Y., Lv, T., Cui, L., Lu, Y., and Wei, F. LayoutLMv3: Pre-training for Document AI with Unified Text and Image Masking. In *Proceedings of the 30th ACM International Conference on Multimedia*, 2022.

JaidedAI. EasyOCR, 2024. URL https://github.com/JaidedAI/EasyOCR. Computer software.

Kerr, O. S. Buying data and the fourth amendment. Aegis Series Paper 2109, Hoover Institution, 2021. Available at SSRN: https://ssrn.com/abstract=3880130.

Kim, G., Hong, T., Yim, M., Nam, J., Park, J., Yim, J., Hwang, W., Yun, S., Han, D., and Park, S. OCR-Free Document Understanding Transformer. In *European Conference on Computer Vision*, pp. 498–517, 2021.

Kosinski, M., Stillwell, D., and Graepel, T. Private traits and attributes are predictable from digital records of human behavior. *Proceedings of the National Academy of Sciences*, 110:5802 – 5805, 2013.

Lerouge, J., Betmont, G., Bres, T., Stepankevich, E., and Berges, A. DocXPand-25k: a large and diverse benchmark dataset for identity documents analysis. *ArXiv*, abs/2407.20662, 2024.

Li, R., Allal, L. B., Zi, Y., Muennighoff, N., Kocetkov, D., Mou, C., Marone, M., Akiki, C., Li, J., Chim, J., Liu, Q., Zheltonozhskii, E., Zhuo, T. Y., Wang, T., Dehaene, O., Davaadorj, M., Lamy-Poirier, J., Monteiro, J., Shliazhko, O., Gontier, N., Meade, N., Zebaze, A., Yee, M.-H., Umapathi, L. K., Zhu, J., Lipkin, B., Oblokulov, M., Wang, Z., Murthy, R., Stillerman, J., Patel, S. S., Abulkhanov, D., Zocca, M., Dey, M., Zhang, Z., Fahmy, N., Bhattacharyya, U., Yu, W., Singh, S., Luccioni, S., Villegas, P., Kunakov, M., Zhdanov, F., Romero, M., Lee, T., Timor, N., Ding, J., Schlesinger, C., Schoelkopf, H., Ebert, J., Dao, T., Mishra, M., Gu, A., Robinson, J., Anderson, C. J., Dolan-Gavitt, B., Contractor, D., Reddy, S., Fried, D., Bahdanau, D., Jernite, Y., Ferrandis, C. M., Hughes, S. M., Wolf, T., Guha, A., Werra, L. V., and Vries, H. D. StarCoder: may the source be with you! *Trans. Mach. Learn. Res.*, 2023, 2023.

Lin, K. Q., Li, L., Gao, D., Yang, Z., Wu, S., Bai, Z., Lei, W., Wang, L., and Shou, M. Z. ShowUI: One Vision-Language-Action Model for GUI Visual Agent, 2024.

Lu, Y., Yang, J., Shen, Y., and Awadallah, A. OmniParser for Pure Vision Based GUI Agent. *ArXiv*, abs/2408.00203, 2024.

Lukas, N., Salem, A., Sim, R., Tople, S., Wutschitz, L., and Zanella-Béguelin, S. Analyzing Leakage of Personally Identifiable Information in Language Models. In *2023 IEEE Symposium on Security and Privacy (SP)*, pp. 346–363, 2023. doi: 10.1109/SP46215.2023.00038.

Lù, X. H., Kasner, Z., and Reddy, S. WebLINX: Real-World Website Navigation with Multi-Turn Dialogue. *ArXiv*, abs/2402.05930, 2024.

Microsoft. Playwright, 2024a. URL https://github.com/microsoft/playwright.

Microsoft. Presidio: Data Protection and De-identification SDK, 2024b. URL https://github.com/microsoft/presidio.

Narayanan, A. and Shmatikov, V. Robust de-anonymization of large sparse datasets. *2008 IEEE Symposium on Security and Privacy (sp 2008)*, pp. 111–125, 2008.

Nasr, M., Carlini, N., Hayase, J., Jagielski, M., Cooper, A. F., Ippolito, D., Choquette-Choo, C. A., Wallace, E., Tramèr, F., and Lee, K. Scalable Extraction of Training Data from (Production) Language Models. *ArXiv*, abs/2311.17035, 2023.

OpenAI. GPT-4o mini: Advancing cost-efficient intelligence, July 2024. https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/.

OpenAI. GPT-5 System Card Update: GPT-5.2. Technical report, OpenAI, December 2025. URL https://openai.com/index/gpt-5-system-card-update-gpt-5-2/.

Pahuja, V., Lu, Y., Rosset, C., Gou, B., Mitra, A., Whitehead, S., Su, Y., and Awadallah, A. Explorer: Scaling Exploration-driven Web Trajectory Synthesis for Multimodal Web Agents. In *Annual Meeting of the Association for Computational Linguistics*, pp. 6300–6323, 2025.

Sasaya, N., Kishida, S., Kikuchi, R., and Tajima, A. Validating behavioral proxies for disease risk monitoring via large-scale e-commerce data, 2026.

Selvam, S. and Ghosh, A. PANORAMA: A synthetic PII-laced dataset for studying sensitive data memorization in LLMs. *ArXiv*, abs/2505.12238, 2025.

Smith, R. An Overview of the Tesseract OCR Engine. In *Proceedings of the 9th International Conference on Document Analysis and Recognition (ICDAR 2007)*, volume 2, pp. 629–633. IEEE Computer Society, 2007. doi: 10.1109/ICDAR.2007.4376991.

Sobel, A. End-running warrants: Purchasing data under the fourth amendment and the state action problem. *Yale Law & Policy Review*, 2023. Available at SSRN: https://ssrn.com/abstract=4480782.

SST. OpenCode: The Open Source AI Coding Agent, 2025. URL https://github.com/opencode-ai/opencode.

Steier, A., Manoel, A., Haushalter, A., and Segbroeck, M. V. Nemotron-PII: Synthesized Data for Privacy-Preserving AI, 2025. URL https://huggingface.co/datasets/nvidia/Nemotron-PII.

Tokson, M. J. Government purchases of private data. *Wake Forest Law Review*, 59:269, 2024. University of Utah College of Law Research Paper No. 573.

Veit, A., Matera, T., Neumann, L., Matas, J., and Belongie, S. J. COCO-Text: Dataset and Benchmark for Text Detection and Recognition in Natural Images. *ArXiv*, abs/1601.07140, 2016.

Wang, H., Zou, H., Song, H., et al. UI-TARS-2 Technical Report: Advancing GUI Agent with Multi-Turn Reinforcement Learning, 2025a.

Wang, W., Yu, Z., Ye, R., Zhang, J., Chen, S., and Wang, Y. FedMABench: Benchmarking Mobile Agents on Decentralized Heterogeneous User Data. *ArXiv*, abs/2503.05143, 2025b.

Wang, W., Yuan, M., Yu, Z., Liu, G., Ye, R., Jin, T., Chen, S., and Wang, Y. MobileA3gent: Training Mobile GUI Agents Using Decentralized Self-Sourced Data from Diverse Users. *Proceedings of the Fourth Workshop on Bridging Human-Computer Interaction and Natural Language Processing (HCI+NLP)*, 2025c.

Wang, X., Wang, B., Lu, D., Yang, J., Xie, T., Wang, J., Deng, J., Guo, X., Xu, Y., Wu, C. H., Shen, Z., Li, Z., Li, R., Li, X., Chen, J., Zheng, B., Li, P., Lei, F., Cao, R., Fu, Y., Shin, D., Shin, M., Hu, J., Wang, Y., Chen, J., Ye, Y., Zhang, D., Du, D., Hu, H., Chen, H., Zhou, Z., Yao, H., Chen, Z., Gu, Q., Wang, Y., Wang, H., Yang, D., Zhong, V., Sung, F., Y.Charles, Yang, Z., and Yu, T. OpenCUA: Open Foundations for Computer-Use Agents. *ArXiv*, abs/2508.09123, 2025d.

Wang, Z., Liang, Y., Zhang, X., Wu, Q., Han, S., Bastos, A., Wang, R., Bansal, C., Peng, B., Gao, J., Rajmohan, S., and Yao, H. Adapting Web Agents with Synthetic Supervision. *ArXiv*, abs/2511.06101, 2025e.

Xie, T., Zhang, D., Chen, J., Li, X., Zhao, S., Cao, R., Hua, T., Cheng, Z., Shin, D., Lei, F., Liu, Y., Xu, Y., Zhou, S., Savarese, S., Xiong, C., Zhong, V., and Yu, T. OSWorld: Benchmarking Multimodal Agents for Open-Ended Tasks in Real Computer Environments. *ArXiv*, abs/2404.07972, 2024.

Yang, J., Zhang, H., Li, F., Zou, X., yue Li, C., and Gao, J. Set-of-Mark Prompting Unleashes Extraordinary Visual Grounding in GPT-4V, 2023.

You, E. Vite, 2024. URL https://github.com/vitejs/vite.

Zhou, S., Xu, F. F., Zhu, H., Zhou, X., Lo, R., Sridhar, A., Cheng, X., Bisk, Y., Fried, D., Alon, U., and Neubig, G. WebArena: A Realistic Web Environment for Building Autonomous Agents. *ArXiv*, abs/2307.13854, 2023.

Zhou, Z., Chen, X., Li, E., Zeng, L., Luo, K., and Zhang, J. Edge Intelligence: Paving the Last Mile of Artificial Intelligence With Edge Computing. *Proceedings of the IEEE*, 107:1738–1762, 2019.

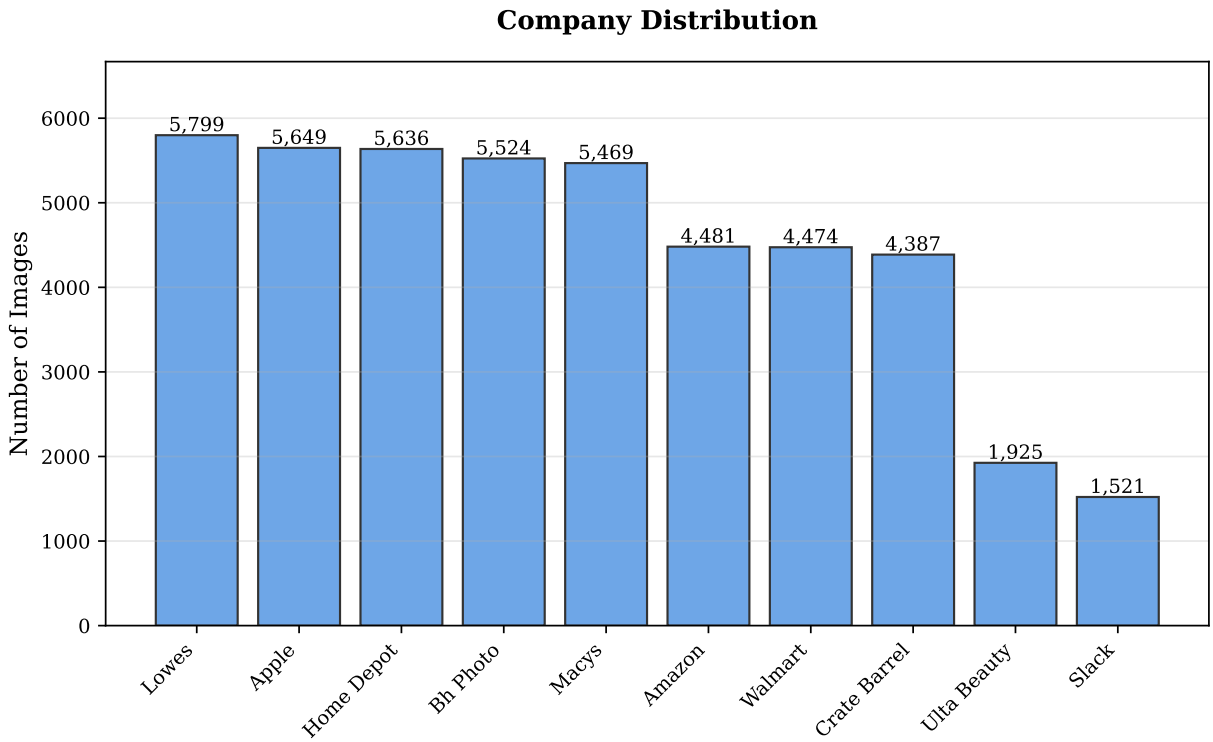


Figure 6. Distribution of base images across 10 e-commerce companies. Apple and Amazon have the most coverage (1,400 images each), while Slack and Ulta Beauty represent smaller verticals (300 images each). Total base images: 10,200 (multiplied across variants to produce 44,865 total dataset images).

A. Dataset Statistics

The dataset comprises 44,865 images spanning 10 e-commerce websites and 19 page types. Figures 6–7 show company and page type distributions omitted from the main paper for space.

The 52.4%/47.6% split between PII and non-PII annotations ensures models learn to distinguish sensitive user data from product and order metadata. Most annotations (78.1%) target rendered text rather than input fields (13.6%), reflecting that PII appears predominantly on confirmation and review pages where entered data is displayed, not just in the form fields where users type.

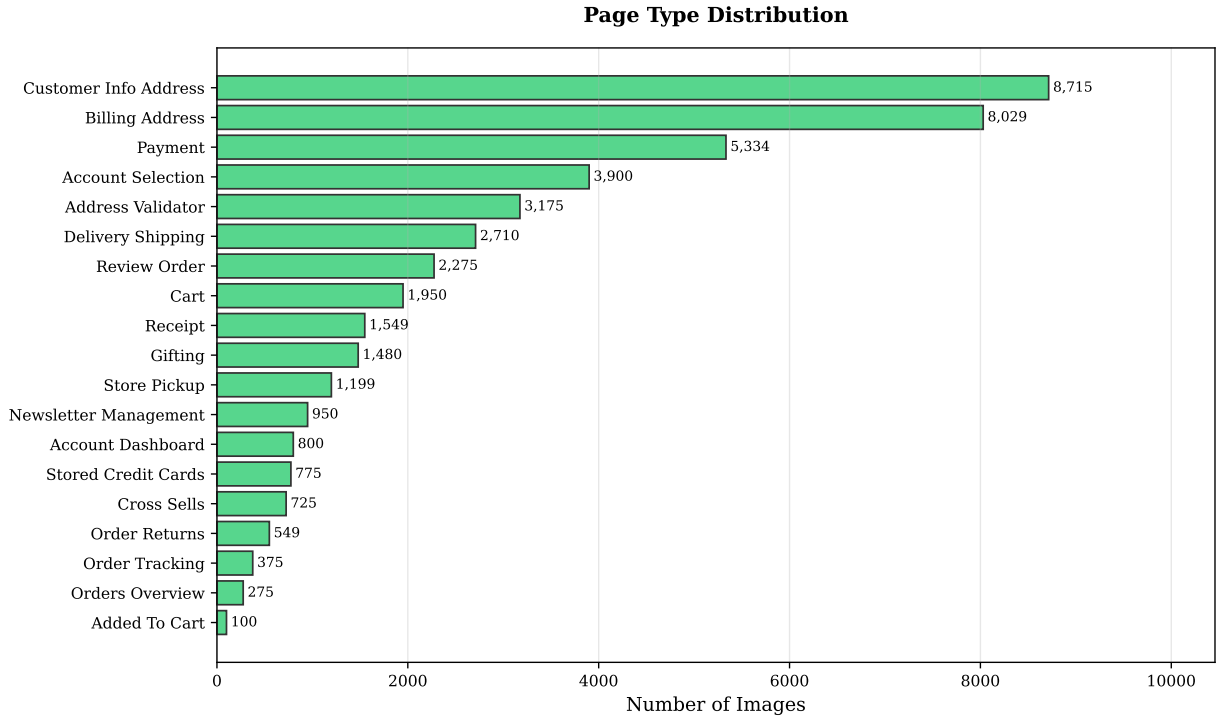


Figure 7. Distribution of base images across 19 page types spanning checkout flows, account management, and product pages. Account selection (900) and delivery/shipping (875) pages dominate, representing critical moments where PII exposure is highest.

B. Generation and Quality Control Pipeline

B.1. Generation Pipeline Details

We collected screenshots from 10 e-commerce brands across 19 page types—account dashboards, order history, order tracking, checkout flows, cart, billing address, delivery options, payment entry, gifting, store pickup, and product pages. Selection prioritized layouts exhibiting complex UI patterns: modal overlays, dynamic sidebars, embedded maps for store locators, cluttered multi-step forms, and responsive designs. We source multiple distinct layouts, including both current designs and prior designs that remain accessible, yielding 408 unique layouts. All original screenshots serve only as visual targets for reproduction; we do not use them directly in the dataset.

For generation, we evaluated GPT-5.2 with OpenCode and Claude Opus 4.5 with Claude Code; Claude exhibited superior rule-following on annotation conventions, requiring fewer iterations to reliably apply the schema. The model generates React components with two enforced constraints: all PII and product information must reference data variables rather than hardcoded values, and every sensitive element must include a data attribute for annotation extraction. A Vite development server renders each component, and a Playwright harness captures screenshots while extracting bounding boxes for all attributed elements via DOM queries.

We provide the model with common e-commerce assets—company and payment method logos, security badges, shipping carrier icons, user avatar images, and map tiles—and pre-install icon libraries (Lucide, Heroicons, Phosphor) to avoid token-heavy inline SVG generation.[1] Generating accurate reproductions requires decomposing the task into specialized prompts, with context cleared between stages to prevent degradation. The pipeline proceeds in four stages: (1) *structure*, where the model generates a React component from the source screenshot, replacing all PII and product information with data variable references; (2) *attribute marking*, where a second pass tags each reference with annotation attributes for bounding box extraction; (3) *input handling*, where form fields are configured for partial-fill simulation; and (4) *visual refinement*, where the rendered output is compared against the source in partitioned regions, with targeted fixes applied only to sections with discrepancies.

Beyond reproduction, the model extracts shipping costs and tax rates from source screenshots, derives order totals from item quantities, and infers platform-specific identifier formats. Because refinement occurs before rendering the full Cartesian product of fill states and data configurations, a single successful correction propagates to many final images.

B.2. Review Interface and Workflow

The main paper summarizes the human review loop and aggregate refinement statistics. Here we provide supplemental implementation details. The Flask review interface operates on a queue system: layouts are displayed sequentially, and a code viewer exposes the `App.jsx` source and `data.json` configuration for manual inspection when needed.

When an annotator identifies an issue, they submit a text instruction describing the required change. The system appends this instruction to a specialized fixing prompt that includes the current `App.jsx` source code, the original screenshot, and context from any previous fix iterations for that layout. Claude modifies the code and returns the corrected version. The interface then triggers re-rendering and re-annotation across all progressive fill states, allowing the annotator to verify the fix was applied correctly. Each layout underwent three review passes to ensure quality and provide redundancy in catching errors that might have been missed in earlier passes.

Refinement operates at the layout level rather than per-reproduction. Once a layout is corrected, the system re-screenshots across all three progressive fill states (empty, partial-fill, and fully-filled) to generate the final annotated dataset images, amortizing the cost of human review across multiple training samples.

We explicitly addressed the observed failure modes within the generation prompts themselves, providing specific instructions about viewport constraints, data attribute requirements, and form field initialization. However, rule-following remained imperfect for very nuanced cases—Claude would occasionally still produce viewport overflow or omit non-obvious data attributes despite explicit prompt guidance. Additionally, some errors were highly company-specific, and maintaining individually fine-tuned prompts per company would have undermined system scalability. We considered integrating more structured rubrics or checklists into the initial UI generation process to guide Claude through additional verification passes before human review. However, this would have introduced substantial costs through redundant forward passes that

[1] User avatars sourced from xsgames.co/randomusers; map tiles from openstreetmap.org (© OpenStreetMap contributors, ODbL).

might not address the primary failure modes (viewport overflow detection, which requires human visual inspection, and context-dependent decisions about which non-obvious elements merit annotation). The conversational fixing interface proved more cost-effective by allowing targeted corrections only where needed rather than exhaustive verification on every layout.

B.3. Refinement Statistics

Beyond the aggregate statistics in Figure 3, Table 2 breaks down refinements by company and page type. Home Depot and Apple layouts required the most corrections (56 and 39 iterations respectively). Billing and payment pages accounted for 57 iterations across 29 layouts, the highest refinement density of any page type.

Table 2. Distribution of human-initiated refinements across companies and page types, ordered by total fixes.

Category	Samples Refined	Total Fixes
By Company		
Home Depot	22	56
Lowes	23	44
Amazon	19	43
Apple	25	39
Macy's	13	39
BH Photo	21	33
Walmart	13	23
Crate & Barrel	15	20
Others	9	15
By Page Type		
Billing/Payment	29	57
Cart	23	38
Customer Info/Address	13	23
Address Validator	10	21
Store Pickup	8	20
Added to Cart	3	19
Others	74	134
Total	**160**	**312**

B.4. Error Analysis and Failure Modes

We manually categorized refinement iterations to understand concrete failure modes beyond the aggregate breakdown in Figure 3. Four primary error categories emerged:

Layout issues (53.8%). The dominant failure mode involved UI elements being pushed outside the viewport width due to incorrect responsive sizing or fixed-width constraints. Claude's initial reproduction iterations included a visual refinement phase where a separate model invocation would compare rendered output to the original screenshot and suggest corrections. However, depsite iterating over various prompts, Claude remained lacking robust spatial reasoning in images—when elements overflowed horizontally beyond the viewport boundary, the model could not detect the issue from the rendered screenshot alone. These errors required human annotators to identify clipped content (e.g., right-aligned cart summaries cut off at 1400px width, sidebar filters extending beyond viewport) and explicitly instruct Claude to adjust container widths or implement proper responsive constraints.

Missing or spurious data attributes (24.4%). These refinements predominantly involved product metadata fields like model numbers and item identifiers that were visually present but lacked proper `data-product` markup, likely do to the inapparent association with the general concept of PII. Apple interfaces exhibited a distinct failure mode: spurious `data-pii` or `data-product` attributes added to decorative elements or redundant container divs that should not have been annotated. These false positive annotations required explicit removal instructions.

Incomplete fields (14.7%). Claude occasionally omitted input fields that did not match the common conceptual model of PII, requiring explicit addition instructions. A related issue involved selection dropdown defaults: Claude would sometimes hardcode dropdowns to specific values (e.g., "United States" for country selection) instead of initializing them to placeholder states ("Select a country"), which prevented proper partial-fill behavior where the dropdown should appear unselected. Note

that certain fields were intentionally left unannotated—newsletter subscription email inputs, for example, were excluded from annotation as they represent marketing opt-ins rather than transaction-critical PII.

Hardcoded store locations (7.1%). Some layouts contained store-specific location references that were not necessarily the user's personal location but could still be identifying or context-dependent—for example, "Presque Isle's Lowes" in headers where the store location was not the key focus of the page. Claude initially rendered these as literal placeholder text rather than parametrizing them, requiring explicit correction.

B.5. Throughput and Pipeline Efficiency

The pipeline automated the full re-rendering and screenshot capture process after each submitted fix: it rebuilt the layout, captured screenshots across all three progressive fill states, reran annotation extraction, and presented updated results within seconds. This fast iteration cycle eliminated manual overhead and enabled annotators to verify fixes immediately without context switching.

The primary bottleneck in dataset construction proved to be the collection of original e-commerce page screenshots rather than the generation or refinement process itself. Capturing diverse checkout flows, account dashboards, and payment pages across 10 major retailers required navigating authenticated sessions, filling realistic test data, and ensuring full-page screenshot capture across various UI states. Once source screenshots were collected, the automated pipeline consumed them rapidly. This architectural decision—delegating both generation and refinement to the model—proved essential for dataset feasibility, as manual HTML/CSS authoring or code-level debugging would have required orders of magnitude more human time.

C. WEBREDACT-LARGE Ablations

While WEBREDACT targets real-time inference at 30fps, WEBREDACT-LARGE prioritizes accuracy while maintaining near-real-time performance (∼3fps on CPU). This section provides complete analysis of WEBREDACT-LARGE trained at 1280×1280 resolution, including architectural comparisons, per-variant performance breakdowns, and inference latency measurements across hardware configurations.

C.1. Architecture Comparison

Table 3 compares WEBREDACT and WEBREDACT-LARGE on $\text{Test}_{\text{Cross-Page}}$ with 10 text variants.

Table 3. Architecture comparison (2-class, cross-page, 10 text variants, trained on full+partial).

MODEL	RESOLUTION	MAP@50	PRECISION	RECALL
WEBREDACT	640×640	0.807	0.848	0.762
WEBREDACT-LARGE	1280×1280	**0.909**	**0.868**	**0.834**

WEBREDACT-LARGE achieves 0.909 mAP@50—a 12.6% relative improvement over WEBREDACT (0.807). Both precision and recall improve substantially (+2.0pp and +7.2pp respectively), indicating that higher resolution enables more accurate field boundary detection and reduces false positives.

C.2. Per-Variant Performance

Table 4 shows per-variant performance breakdown for WEBREDACT-LARGE across test fill states.

Table 4. WEBREDACT-LARGE per-variant performance (2-class, cross-page, 10 text variants).

TEST VARIANT	MAP@50	PRECISION	RECALL
FULL	0.886	0.896	0.815
PARTIAL	0.943	0.897	0.878
EMPTY	0.898	0.809	0.809
AVERAGE	**0.909**	0.868	0.834

The model achieves particularly strong performance on partial-fill images (0.943 mAP@50), where the progressive fill training provides rich learning signal. Performance on fully-filled forms (0.886) is slightly lower, likely due to increased visual complexity when all fields contain text.

C.3. Inference Latency

Table 5 compares inference latency between WEBREDACT and WEBREDACT-LARGE on both CPU and GPU hardware.

Table 5. Inference latency comparison (milliseconds per image).

MODEL	CPU (I5/RYZEN 5)	GPU (RTX 30XX)
WEBREDACT	∼20MS	<5MS
WEBREDACT-LARGE	∼312MS	∼13MS

WEBREDACT satisfies real-time constraints on CPU (<33ms per frame for 30fps), while WEBREDACT-LARGE requires 312ms (∼3fps)—nearly 16× slower but still suitable for near-real-time applications where higher accuracy justifies relaxed frame rate requirements. On GPU hardware, WEBREDACT-LARGE achieves 13ms latency, enabling real-time operation with GPU acceleration. For CPU-only deployment scenarios prioritizing maximum throughput, WEBREDACT remains the optimal choice.

D. Additional Results

D.1. Text-Based Baseline Details

D.1.1. DOCUMENT UNDERSTANDING MODELS VS. OCR PIPELINES

We compare three categories of text-based baselines on $\text{Test}_{\text{Cross-Company}}$ (Amazon, full-fill images): document understanding models (LayoutLMv3 (Huang et al., 2022), Donut (Kim et al., 2021)), OCR+LLM pipelines, and OCR+NER systems. Table 6 shows that document understanding models outperform simple OCR+LLM pipelines, with LayoutLMv3 achieving the highest mAP@50 (0.357) among all text-based approaches. To ensure fair comparison and bypass DocumentQA's single-span extraction limitation, all document understanding pipelines use GPT-4o-mini for final classification after visual/layout encoding rather than extractive question answering.

Table 6. Document understanding models vs. OCR+LLM baselines on $\text{Test}_{\text{Cross-Company}}$ (Amazon, full-fill images). All methods use GPT-4o-mini for classification.

METHOD	MAP@50	PREC.	REC.	F1	LATENCY
LAYOUTLMV3 (HUANG ET AL., 2022)	**0.357**	**58.5%**	45.1%	**50.9%**	2.9S
DONUT (KIM ET AL., 2021)	0.350	55.8%	43.6%	48.9%	3.5S
TESSERACT (SMITH, 2007) + LLM	0.302	49.8%	**49.6%**	49.7%	2.8S

Document understanding models leverage visual layout information alongside text content, providing better localization on complex layouts like Amazon's dense product grids and multi-column interfaces. However, this advantage remains modest (+18% mAP@50 over Tesseract (Smith, 2007)+LLM), and the architectural mismatch between word-level detections and field-level ground truth limits all text-based approaches.

D.1.2. OCR ENGINE ABLATION

Table 7 compares OCR engines with GPT-4o-mini classification. PaddleOCR (Cui et al., 2025) achieves the highest precision (78.7%) but lowest recall (31.0%), while Tesseract (Smith, 2007) provides the best balance with competitive mAP@50 (0.302) at the fastest speed (2.8s). The narrow performance band (0.286–0.329 mAP@50) reveals that OCR quality is not the primary bottleneck—the architectural mismatch between word-level detections and field-level annotations limits all engines.

Table 7. OCR engine ablation with GPT-4o-mini classification.

OCR ENGINE	MAP@50	PREC.	REC.	F1	LATENCY
PADDLEOCR (CUI ET AL., 2025)	**0.329**	**78.7%**	31.0%	44.5%	6.6S
TESSERACT (SMITH, 2007)	0.302	49.8%	**48.6%**	**49.2%**	**2.8S**
EASYOCR (JAIDEDAI, 2024)	0.286	67.6%	30.7%	42.2%	3.8S

The classification latency (2.8–6.6s) is dominated by LLM inference rather than OCR extraction. Table 8 shows OCR-only latencies. Tesseract (Smith, 2007) completes in 453ms on CPU, while GPU-accelerated engines (EasyOCR (JaidedAI, 2024), PaddleOCR (Cui et al., 2025)) are 1.6–4.7× slower despite hardware acceleration, indicating that OCR speed is not a primary bottleneck for text-based baselines, although still slower than our WEBREDACT and WEBREDACT-LARGE models.

Table 8. OCR engine latency (extraction only, no classification).

OCR ENGINE	LATENCY
TESSERACT (SMITH, 2007)	**453MS**
EASYOCR (JAIDEDAI, 2024)	715MS
PADDLEOCR (CUI ET AL., 2025)	2,143MS

D.1.3. OCR + PRESIDIO BASELINE

Table 9 shows OCR+Presidio performance across engines. Presidio's rule-based NER achieves only 0.176–0.183 mAP@50, approximately 40% worse than LLM-based classification. Tesseract (Smith, 2007) provides the fastest configuration (1.3s),

while PaddleOCR's (Cui et al., 2025) higher-quality extraction provides minimal benefit when the classifier lacks contextual understanding.

Table 9. OCR + Presidio NER (no LLM).

OCR ENGINE	MAP@50	PREC.	REC.	F1	LATENCY
TESSERACT (SMITH, 2007)	**0.183**	41.7%	**26.3%**	**32.2%**	**1.3S**
EASYOCR (JAIDEDAI, 2024)	0.178	42.1%	28.3%	33.9%	2.3S
PADDLEOCR (CUI ET AL., 2025)	0.176	**44.5%**	25.1%	32.1%	4.8S

D.1.4. DOCUMENT QUESTION ANSWERING

An alternative to LLM classification is Document Question Answering (DocQA), where models answer explicit questions like "What is the customer name?" to extract PII. We evaluate two pure DocQA approaches—Donut-DocVQA and LayoutLM with extractive QA heads—against the hybrid configurations (LayoutLMv3 + LLM, Donut + LLM) described above. Table 10 compares all approaches on Amazon test images.

Table 10. Document QA vs. LLM classification on Amazon test images.

METHOD	MAP@50	PREC.	REC.	F1	LATENCY
LAYOUTLMV3 (HUANG ET AL., 2022) + LLM	**0.357**	**58.5%**	45.1%	**50.9%**	**2.9S**
DONUT (KIM ET AL., 2021) + LLM	0.350	55.8%	43.6%	48.9%	3.5S
LAYOUTLM-QA	0.139	24.1%	**47.8%**	32.0%	1.5S
DONUT-DOCVQA	0.117	30.8%	37.7%	33.9%	11.5S

DocQA approaches fail for several reasons. First, DocVQA models are trained to extract single answer spans per question (e.g., "What is the date?" $\rightarrow$ "October 17"), not multiple items. This requires separate forward passes for each PII category (name, address, email, phone, card number, etc.), causing substantial latency: Donut's OCR-free architecture processes the full image for each question at $\sim$600ms per question (11.5s total), while LayoutLM's extractive QA operates on pre-extracted OCR text more efficiently (1.5s total). Second, both models are pre-trained on structured documents and forms, where PII appears in predictable locations with clear visual cues (labeled fields, tables). E-commerce screenshots present different visual hierarchies—gift recipient versus billing names, optional fields, promotional overlays—that deviate from the models' training distribution. Third, rigid question templates often mismatch actual page content, requiring flexible contextual understanding that the single-span extraction paradigm cannot provide. Fourth, LayoutLM-QA uses a randomly initialized QA head (not fine-tuned), further degrading span extraction quality.

LLM classification (sending all extracted text to GPT-4o-mini for contextual classification) achieves 2.6$\times$ higher mAP@50 (0.357 vs 0.139) than the best DocQA method while maintaining comparable or faster speed. The flexible classification paradigm handles diverse layouts without rigid question templates.

D.1.5. FAILURE MODES

The two-stage architecture creates cascading errors where OCR misdetections propagate to classification. On sparse pages with few ground truth elements, OCR misreads generic footer text or company names, causing the LLM to aggressively flag these misdetected strings as sensitive and producing false positives. Conversely, dense pages with complex product listings cause context rot in chunked LLM calls, where the model loses track of relevant fields as listings overwhelm the limited context window. Product names containing person names (e.g., "Kelsey Villages" as a street name versus "Kelsey" as a brand) are frequently misclassified, and gift messages containing potentially private data are sometimes missed.

D.1.6. CLASSIFICATION PROMPT

The OCR + LLM baselines use GPT-4o-mini to classify extracted text spans. The classification prompt is:

```
You identify data values in e-commerce screenshot text.
```

```
Flag ANY text that is actual data (not a UI label):
Names, addresses, cities, states, zip codes
Emails, phone numbers, dates
Card numbers, CVV, expiry dates
Product names, brands, prices, quantities, ratings
Order totals, shipping costs, tracking numbers
Search queries, gift messages

Only skip pure UI labels like "Price:", "Quantity:", "Add to cart".
When in doubt, include it.

Return JSON: {"pii_items": [{"text": "exact text"}]}
```

D.2. Per-Class Performance Breakdown

Table 11 shows per-class AP@50 on fully-filled test images across split strategies for 2-class detection.

Table 11. Per-class AP@50 on fully-filled test images (test_full variant).

SPLIT	TEXT	IMAGE
CROSS-PAGE	0.623	0.870
CROSS-COMPANY	0.471	0.855
CROSS-TYPE	0.552	0.875

Product images achieve robust detection across all splits (0.855–0.875 AP@50), confirming that visual elements have distinctive signatures that generalize well regardless of company branding or page type. Text detection shows more variation: cross-page performance reaches 0.623 AP@50, while cross-company and cross-type splits achieve 0.471 and 0.552 AP@50 respectively. This degradation reflects the challenge of generalizing text field detection when visual styling (borders, fonts, spacing) varies across unseen companies or form types.

D.3. Split Strategy Ablation

Table 12 compares three split strategies that evaluate generalization at different levels. $\text{Test}_{\text{Cross-Page}}$ achieves the strongest average performance (0.797 mAP@50), with particularly high partial-fill detection (0.842), indicating that models learn layout-invariant features within a company's design system. $\text{Test}_{\text{Cross-Company}}$ degrades to 0.753 when generalizing to Amazon's distinct visual style, with the largest drop on full-fill images (0.663) where dense product layouts and unique styling present the greatest challenge. $\text{Test}_{\text{Cross-Type}}$ shows the most uniform degradation across fill states (0.713–0.744), revealing that page-type-specific patterns—such as dense tabular receipts versus interactive checkout forms—transfer less effectively than company-specific design conventions.

Table 12. Split strategy comparison (WEBREDACT, 2-class, trained on full+partial). $\text{Test}_{\text{Cross-Page}}$ performs best, while $\text{Test}_{\text{Cross-Type}}$ presents the strongest generalization challenge.

SPLIT STRATEGY	FULL	PARTIAL	EMPTY	AVG
$\text{TEST}_{\text{CROSS-PAGE}}$	0.747	0.842	0.803	**0.797**
$\text{TEST}_{\text{CROSS-COMPANY}}$	0.663	0.839	0.756	0.753
$\text{TEST}_{\text{CROSS-TYPE}}$	0.713	0.727	0.744	0.728

D.4. Fill State Ablation

Table 13 ablates training data composition to isolate the contribution of each fill state. Training on full screenshots alone achieves 0.771 mAP@50. Counterintuitively, adding empty screenshots *without* partials degrades performance to 0.758, likely because the visual gap between empty forms (placeholder text, unfilled fields) and fully-filled forms is too large for the model to bridge without intermediate examples. Adding partial-fill data resolves this: full+partial achieves 0.797 (+0.026 over full-only), with the strongest gains on partial test images (0.842 versus 0.810), confirming that mid-entry states provide essential visual grounding. Combining all three states achieves the best overall performance (0.825 mAP@50), demonstrating that empty forms become useful once partial fills provide the intermediate signal that bridges the visual gap.

Table 13. Fill state ablation on $\text{Test}_{\text{Cross-Page}}$ (WEBREDACT, 2-class). Partial-fill data is essential; empty screenshots help only when combined with partials.

TRAIN	FULL	PART.	EMPTY	AVG
EMPTY	0.672	0.818	0.777	0.756
FULL	0.736	0.810	0.767	0.771
FULL+EMPTY	0.734	0.774	0.768	0.758
FULL+PART.	0.747	0.842	0.803	0.797
FULL+PART.+EMPTY	**0.762**	**0.881**	**0.831**	**0.825**

D.5. Progressive Fill Density Ablation

We ablate the number of partial-fill screenshots per layout to measure how progressive completion stages affect model performance. Each layout generates multiple partial-fill variants representing different stages of form completion. We evaluate 1, 3, and 5 partial stages; higher sampling counts would effectively approximate capturing all possible intermediate states (e.g., sampling 7 partials from a 7-field form would capture every field-by-field completion step).

Table 14. Progressive fill density ablation (WEBREDACT, 2-class, cross-page, trained on full+partial). Shows per-variant breakdown and overall metrics.

PARTIALS	TRAIN	TEST_FULL	TEST_PARTIAL	TEST_EMPTY	AVG MAP	PREC.	REC.
1	521	0.727	0.774	0.774	0.758	0.793	0.688
3	763	0.738	0.794	0.779	0.770	0.811	0.709
5	938	0.762	0.835	0.810	**0.802**	0.822	0.727

Performance improves consistently with more partial stages, increasing from 0.758 mAP@50 (1 partial) to 0.802 (5 partials), with gains across all test variants. Both precision and recall improve (0.793 to 0.822 and 0.688 to 0.727 respectively), demonstrating that each progressive completion stage provides non-redundant learning signal. The model shows particularly strong gains on test_partial images (0.774 to 0.835), confirming that training on multiple intermediate form-filling states enables more robust field detection patterns.

D.6. Data Variant Ablation Details

We ablate the number of text variants per layout to measure how data diversity affects generalization. Each unique layout is rendered with 1, 10, or 25 distinct PII data injections (varying names, addresses, products, etc.).

Table 15. Text variant ablation on $\text{Test}_{\text{Cross-Page}}$ (WEBREDACT, 2-class, trained on full+partial). Shows per-variant breakdown and overall metrics.

VARIANTS	TEST_FULL	TEST_PARTIAL	TEST_EMPTY	AVG MAP	PREC.	REC.
1	0.762	0.831	0.791	0.795	0.805	0.713
10	0.774	0.856	0.804	0.811	0.829	0.722
25	0.783	0.866	0.812	**0.820**	0.842	0.731

Performance improves consistently with more text variants, rising through 0.795, 0.811, and 0.820, demonstrating that the injectable data system successfully provides training diversity beyond layout diversity alone. Both precision and recall increase with more variants (0.805 to 0.842 and 0.713 to 0.731 respectively). The model shows particularly strong gains on test_partial images (0.831 to 0.866), confirming that diverse PII data injections (varying names, addresses, products) enable the model to learn more robust patterns that generalize across different data instances.

D.7. Qualitative Prediction Analysis

Figure 8 shows prediction patterns for WEBREDACT and WEBREDACT-LARGE on cross-page test images. Both models demonstrate strong performance on core detection targets: all input fields, product images, and prices achieve near-perfect detection. These well-represented patterns in the training data are reliably captured.

Failures predominantly occur on underrepresented edge cases. The top image exhibits numerous false positives (red boxes), where the model over-detects elements that the ground truth does not annotate. WEBREDACT also annotates promotional

text like "Save up to 15% on future auto deliveries", which represents marketing content rather than user-specific information. These false positives indicate the model responds to certain visual contexts (callouts, banners, emphasized text) where distinguishing promotional content from actual data fields remains challenging. Missed detections (blue boxes) occur when PII appears in atypical visual presentations: the recipient name "Gregory" rendered as a large bold callout at the top of the page rather than within a standard form field, and delivery dates like "Tuesday, May 18" styled in green text rather than conventional black typography. The bottom image shows detection patterns on a payment page, where both models successfully capture most standard form fields but exhibit characteristic failures. WEBREDACT misses hard-to-notice form fields without observable borders when empty, and both models struggle with small numbers in densely packed locations like "2 item" quantity indicators. Both models also produce false positives on an unlabeled dropdown menu—though this reflects legitimate annotation ambiguity, as the same dropdown could reasonably be tagged if rendered with a visible selected value. Despite these edge case failures, form field detection remains robust overall. These failures reflect the dataset's composition: while training data spans diverse e-commerce contexts, forms and cart pages dominate and establish the primary visual patterns for input fields, product images, and prices. The model learns these dominant patterns robustly but struggles when sensitive information appears in more information-dense or ambiguous contexts.

To assess generalization to non-synthetic data, we qualitatively evaluated the model on the original e-commerce screenshots used as generation targets, as well as 50 additional real e-commerce screenshots that were never synthetically reproduced from the same brands. We do not release these images due to their sensitive content. Upon observing the annotations, we recognize that input fields, prices, and product details demonstrate robust detection across varied contexts, confirming that patterns learned from synthetic reproductions transfer to authentic interfaces. However, shipping dates, cardholder names rendered on card images, and basket quantities exhibit inconsistent detection, consistent with the underrepresentation patterns observed on synthetic test data demonstrated in Figure 8.

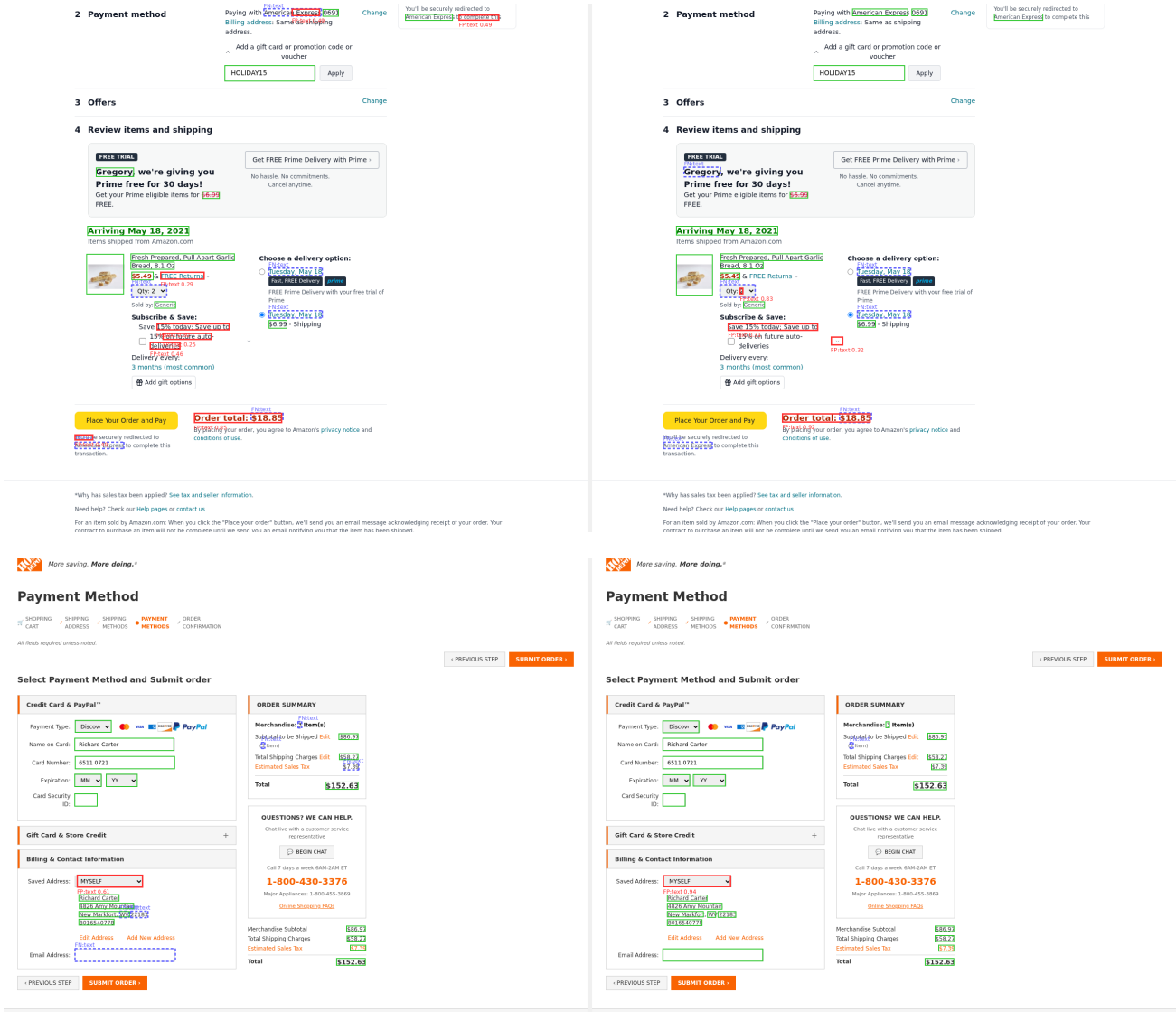


Figure 8. Qualitative comparison of WEBREDACT (left) and WEBREDACT-LARGE (right) predictions on cross-page test images. Green boxes indicate correct alignment with ground truth, red boxes indicate false positives, and blue boxes indicate false negatives.