EDITINK: LEARNING EDIT-SENSITIVE INK TO MAKE AI MANIPULATION LEGIBLE

Anonymous authors
Paper under double-blind review

ABSTRACT

Generative image editing can produce visually convincing local modifications, making manipulated regions difficult to identify. Proactive localization addresses this challenge by embedding signals before image release to provide additional localization cues. However, jointly learning a signal and its reader from actual editing outcomes remains challenging when closed-source editors allow forward queries but not backpropagation. To address this challenge, we propose a localization framework named EditInk that learns a shared edit-sensitive ink together with an ink developer. The ink is an image-embedded signal learned to remain imperceptible before editing and provide spatial cues when disrupted, while the developer reads these cues to identify manipulated regions from the edited image alone. Yet training the developer relies on recognizable disruption cues from the ink, while improving the ink relies on reliable localization guidance from the developer. We address this mutual dependence through an alternating cooperative learning strategy that coordinates two complementary phases. In the edit-driven developer learning phase, we hold the current ink fixed and train the developer on actual edits of protected images. In the subsequent developer-guided ink learning phase, we hold the updated developer fixed and optimize the ink using a protected view and differentiable simulated views constructed by local ink erasure followed by image degradation. Repeatedly alternating these phases allows actual editing outcomes to guide ink learning through the developer, without backpropagating through the editors. Experiments on natural image and face datasets show that EditInk achieves higher performance than the compared passive and proactive baselines on both open and closed source editors.

1 INTRODUCTION

Recent advances in generative image editing enable visually plausible local modifications through natural-language instructions (Brooks et al., 2023; Bai et al., 2024; Yu et al., 2025). Adding, removing, or replacing a single object can change what an image conveys while leaving its overall appearance convincing. Although these capabilities facilitate creative applications, they also lower the barrier to producing misleading visual content and falsified visual records (Zhang et al., 2023; 2024). Since a manipulation may affect only a small region of an authentic image, an image-level authenticity label provides limited information about what has changed (Ţânţaru et al., 2024). Reliable assessment therefore requires identifying manipulated regions, motivating pixel-level manipulation localization.

Existing approaches address this task through passive analysis or proactive signal embedding. Passive forensic methods identify inconsistencies in image content and forensic traces, such as noise and processing artifacts (Guillaro et al., 2023; Xu et al., 2024). However, these cues can be subtle, and their dependence on image content and editing processes makes localization under dataset or generator shifts challenging (Ţânţaru et al., 2024). Proactive methods instead embed a protective signal before image release, providing deliberately constructed cues for subsequent localization. Figure 1(a,b) contrasts two representative mechanisms for proactive localization. *Watermark-based methods* embed an imperceptible watermark before image release and subsequently detect or recover it from the edited image (Sander et al., 2024; Zhang et al., 2023; 2024). Local editing can remove or corrupt the embedded watermark, so regions where it is no longer detected or is incorrectly recovered provide spatial evidence of manipulation. *Pattern-based methods* instead optimize image

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

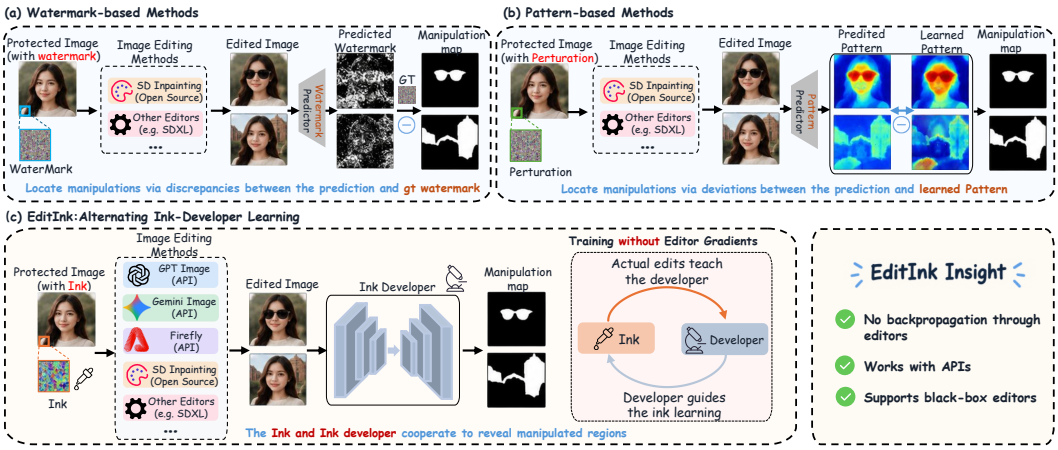

Figure 1: Comparison of proactive localization paradigms.

perturbations to induce prescribed responses in pretrained models on protected images (Song et al., 2025). After editing, they localize manipulated regions by identifying spatial deviations between the observed model responses and the prescribed patterns. However, deviation from a prescribed response does not necessarily distinguish manipulated from unmodified regions. This motivates learning the protective signal and its reader from localization performance on actual editing outcomes, rather than from a prescribed response alone. Yet optimizing the signal through the editing process is challenging: closed-source editors exposed through forward-query APIs provide outputs but no input gradients for backpropagation. This raises a central question: *How can a shared protective signal and its reader learn jointly from actual editing outcomes without backpropagating through the editor?*

To address this challenge, we propose **EditInk**, a proactive localization framework that learns a shared *Edit-Sensitive Ink* δ together with an *Ink Developer* D_θ. The ink is a bounded image-embedded signal regularized to remain visually invisible, while the developer predicts a manipulation map from the edited image alone. The two components need to co-adapt: the developer must learn to interpret the cues produced by the ink, and improving the ink requires meaningful guidance from the developer. Our key idea is to let actual edits teach the developer and use the updated developer to guide ink learning, as illustrated in Figure 1(c). We implement this interaction through an *alternating cooperative learning strategy*. Each cooperation round begins with an *edit-driven developer learning phase*, which fixes the current ink and trains the developer on actual edits of images carrying that ink. Because the edited outputs serve as fixed training observations, this phase requires only forward queries to the editors.

The subsequent *developer-guided ink learning phase* fixes the updated developer and optimizes the shared ink using a protected view and differentiable simulated views. We construct the simulated views by locally erasing the ink and then applying transformations from a differentiable degradation family to the entire resulting image. The ink is optimized so that the developer predicts no manipulation on the protected view and identifies the erased regions in the simulated views, preserving the distinction between ink-retained and ink-erased regions despite image degradation. Alternating the two phases allows actual editing outcomes to shape the developer, whose updated localization behavior in turn guides the next ink update, without differentiating through the editors. At deployment, the shared ink is directly embedded into new images, and localization requires only the subsequently edited image, without the original image or access to the editor.

Our contributions are summarized as follows:

- We introduce EditInk, a proactive localization framework that learns a shared edit-sensitive ink and an accompanying ink developer. The ink is embedded before image release, and the developer localizes manipulated regions from the edited image alone.
- We develop an alternating cooperative learning strategy in which the developer learns from actual editor outputs and then guides ink optimization through differentiable ink-erasure

views. This enables the shared ink to benefit from actual editing results without backpropagating through the editors.

- We evaluate EditInk on HumanEdit-Selected and Face Data using open-source and API-based editors. EditInk achieves higher performance than the compared baselines on both API editors across the two benchmarks.

2 Related Work

Passive manipulation localization. Passive methods localize manipulated regions by analyzing image content and forensic traces without modifying the image before release. TruFor (Guillaro et al., 2023) combines RGB information with a learned noise-sensitive fingerprint to capture complementary high- and low-level evidence. FakeShield (Xu et al., 2024) uses multimodal large language models to generate forgery explanations and guide pixel-level localization. Despite their different representations, both infer manipulation from evidence already present in the received image. Proactive approaches instead embed a signal before release to provide deliberately constructed localization cues.

Watermark-based proactive localization. Watermark-based methods(Chen et al., 2025; Liu et al., 2026; Wu et al., 2026) embed information whose spatial detection or recovery provides evidence of manipulation. EditGuard (Zhang et al., 2023) combines image and bit watermarks for localization and copyright protection, while OmniGuard (Zhang et al., 2024) fuses recovered-watermark cues with image content through a degradation-aware extractor. WAM (Sander et al., 2024) jointly learns watermark embedding, pixel-level watermark detection, and message decoding. Recent work broadens the watermark structure and embedding setting. FractalForensics (Wang et al., 2025) maps semi-fragile fractal watermark entries to image patches, aiming for robustness to benign processing and sensitivity to facial manipulation. StableGuard (Yang et al., 2025) jointly trains a watermarking VAE and a forensic network within latent diffusion pipelines, while GenPTW (Gan et al., 2026) supports latent watermark embedding both during and after generation for provenance tracing and tamper localization. RecoverMark (An et al., 2026) takes a content-recovery route: it embeds facial content into the surrounding background and localizes manipulation by comparing recovered and received facial content.

Template- and pattern-based proactive localization. A related line of work constructs localization cues through learned templates or prescribed model responses. MaLP (Asnani et al., 2023) jointly learns additive templates with a two-branch localization network that captures local and global information, incorporating a generative model into its end-to-end training pipeline. Blank Canvas (Song et al., 2025) optimizes perturbations to suppress the Segment Anything Model's segmentation responses on protected images, then identifies edited regions through the reappearance of these responses. APT (Ha et al., 2026) aligns spatial features with fixed anchors and exploits post-editing alignment differences to localize tampering in fully regenerated images, using either direct readout or a shallow learned decoder. EditInk instead co-adapts a shared ink and developer to actual editing outcomes: the developer learns from editor outputs and is then frozen to guide ink optimization through differentiable ink-erasure views. This alternation lets actual edits progressively shape the ink through the developer without requiring editor gradients.

3 Problem Formulation

Given a source image $x \in [0,1]^{H\times W\times 3}$, our goal is to protect it with a shared, bounded *edit-sensitive ink* $\delta \in \mathbb{R}^{H\times W\times 3}$, such that subsequent AI manipulations become spatially identifiable. The publisher constructs the protected image $x^+ = \mathrm{clip}_{[0,1]}(x+\delta)$. After an editor $\mathcal{E}$ processes x^+ under an editing condition c, producing $y = \mathcal{E}(x^+; c)$, an *ink developer* D_θ predicts the manipulation map $\widehat{M} = D_\theta(y)$ from the edited image alone.

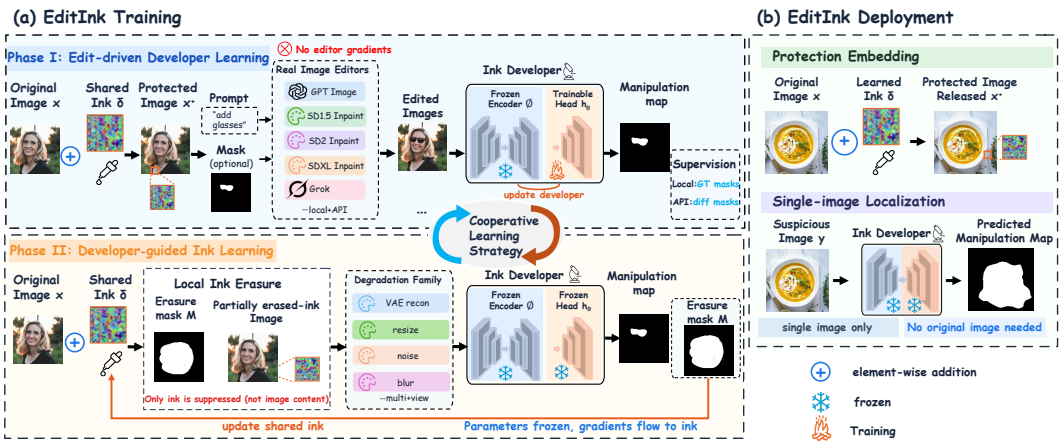

Figure 2: Overview of EditInk. (a) EditInk learns the edit-sensitive ink and ink developer through two complementary training phases coordinated by a cooperative learning strategy. (b) At deployment, the learned ink is embedded into an image before release, and the ink developer localizes manipulated regions from the edited image.

Ideally, the ink δ and developer D_θ should be jointly learned to minimize the localization error on actual editing outcomes:

$$\begin{aligned} \min_{\delta,\theta} \quad & \mathbb{E}_{(x,m,c),\,\mathcal{E}}\left[\mathcal{L}_{\text{loc}}\left(D_\theta(y), M\right)\right] \\ \text{s.t.} \quad & x^+ = \text{clip}_{[0,1]}(x+\delta), \qquad y = \mathcal{E}(x^+; c), \\ & \|\delta\|_\infty \le \epsilon. \end{aligned} \tag{1}$$

Here, M denotes the ground-truth manipulation mask associated with the edited image, and c denotes the editor-specific editing condition, such as an editing instruction and, when applicable, a control mask. The ink δ is shared across images rather than optimized for each individual input, while D_θ receives only the edited image at inference. The objective $\mathcal{L}_{\text{loc}}$ encourages editing-induced changes to the ink to provide spatial cues for manipulation localization.

Directly optimizing Eq. equation 1 poses two challenges. First, black-box editors allow forward queries but do not expose gradients. Their outputs can be used to train the developer, but the localization loss cannot be backpropagated through the editors to update the ink. Second, the ink and developer need to adapt to each other. The developer must learn to recognize the cues produced by the current ink after editing, while ink optimization relies on the developer's localization feedback. Updating the ink can change these cues, requiring the developer to adapt in turn.

4 METHOD

EditInk addresses the two challenges through two complementary learning phases coordinated by a cooperative learning strategy, as illustrated in Figure 2. The **Edit-Driven Developer Learning Phase** addresses the inaccessible editor gradient by fixing the current ink and learning the developer directly from actual outputs of the training editors, requiring only forward queries to the editing process. The **Developer-Guided Ink Learning Phase** then fixes the updated developer and optimizes the ink through differentiable ink-erasure views, allowing the localization objective to provide gradients to the ink without gradient through the editor. Finally, the **Cooperative Learning Strategy** alternates these two phases across multiple rounds, allowing the developer to learn from the disruption patterns produced by the current ink and the ink to adapt to the localization behavior of the updated developer. In this way, EditInk jointly learns the ink and developer using actual editing outcomes while requiring only query access, thus can supports any black-box editors.

4.1 EDIT-DRIVEN DEVELOPER LEARNING PHASE

The first phase addresses how the developer can learn from real black-box editors when their internal architectures and gradients are inaccessible. At cooperation round r, we fix the current ink $\delta^{(r)}$ and embed it into a clean image x:

$$x^{+,(r)} = \mathrm{clip}_{[0,1]}\left(x + \delta^{(r)}\right). \tag{2}$$

For each training editor $\mathcal{E}_j \in \mathscr{E}_{\mathrm{train}}$, we query the editor using the protected image and obtain

$$y_j^{(r)} = \mathcal{E}_j\left(x^{+,(r)}; M, p, \omega\right). \tag{3}$$

These images are the actual outputs of the local or API-based editors. We do not replace the editors with differentiable surrogates or reconstruct their outputs by compositing after editing.

We implement the ink developer as $D_\theta(z) = h_\theta(\phi(z))$, where ϕ is a frozen DINOv2 feature extractor and h_θ is a trainable spatial prediction head. For an input image z and manipulation mask M, we define the localization loss as

$$\mathcal{L}_{\mathrm{loc}}\left(D_\theta(z), M\right) = \mathrm{BCEWithLogits}\left(D_\theta(z), \mathcal{A}(M)\right), \tag{4}$$

where $\mathcal{A}(\cdot)$ resizes the manipulation mask to the developer output resolution using area interpolation. With the current ink and feature extractor fixed, we update only the prediction head by

$$\theta^{(r+1)} = \arg\min_\theta \mathbb{E}_{(x,M,p),\,\mathcal{E}_j,\,\omega}\left[\mathcal{L}_{\mathrm{loc}}\left(D_\theta\left(y_j^{(r)}\right), M\right)\right]. \tag{5}$$

Since each $y_j^{(r)}$ is treated as a fixed observation after querying the editor, optimizing Eq. equation 5 does not require gradients through $\mathcal{E}_j$. The developer therefore learns to localize the disruption patterns produced by the current ink in actual editor outputs.

4.2 DEVELOPER-GUIDED INK LEARNING PHASE

The second phase addresses how the ink can be optimized without backpropagating the localization loss through a black-box editor. We fix the updated developer $D_{\theta^{(r+1)}}$ and optimize the ink using a protected view and differentiable simulated views. The protected view contains the complete ink, whereas the simulated views are constructed by local ink erasure followed by image degradation.

Protected and simulated views. Given a training image–mask pair (x, M), we first construct the protected view

$$x_{\mathrm{pro}}^{(r)} = \mathrm{clip}_{[0,1]}\left(x + \delta^{(r)}\right). \tag{6}$$

Since no manipulation is applied to this view, its localization target is an all-zero mask. To construct the simulated views, we erase the ink inside the manipulation region while retaining it elsewhere:

$$x_{\mathrm{erase}}^{(r)} = \mathrm{clip}_{[0,1]}\left(x + (1 - M) \odot \delta^{(r)}\right), \tag{7}$$

where $\odot$ denotes element-wise multiplication. Thus, $M_i = 1$ removes the ink at pixel i, whereas $M_i = 0$ retains it. The erased region therefore simulates a spatial disruption of the ink with a known localization target M. This construction changes the embedded ink rather than performing a semantic edit, with M specifying the known erasure region.

Training with image distortions has been used to improve the robustness of learned features (Zhu et al., 2018). We further introduce a differentiable degradation family $\mathcal{T} = \{T_{\mathrm{vae}}, T_{\mathrm{resize}}, T_{\mathrm{noise}}, T_{\mathrm{blur}}\}$, including VAE reconstruction, resize round-trip, additive noise, and blur. At each ink update, we sample K transformations from $\mathcal{T}$ and construct simulated views

$$x_{\mathrm{sim}}^{(k,r)} = T_k\left(x_{\mathrm{erase}}^{(r)}\right), \qquad k = 1, \ldots, K. \tag{8}$$

Each transformation is applied to the complete image after local ink erasure. These views encourage the spatial disruption of the ink to remain recognizable under different image degradations.

Ink optimization. With $D_{\theta^{(r+1)}}$ fixed, we optimize the shared ink over the protected and simulated views:

$$\delta^{(r+1)} = \arg\min_{\|\delta\|_\infty \le \epsilon} \frac{1}{K+1}\Bigg[\mathcal{L}_{\text{loc}}\left(D_{\theta^{(r+1)}}(x_{\text{pro}}(\delta)), \mathbf{0}\right) + \sum_{k=1}^{K} \mathcal{L}_{\text{loc}}\left(D_{\theta^{(r+1)}}(x_{\text{sim}}^{(k)}(\delta)), M\right)\Bigg] + \lambda_{\text{reg}} \frac{\|\delta\|_2^2}{3HW}. \tag{9}$$

The protected-view loss encourages the complete ink pattern to be recognized as unmanipulated, whereas the simulated-view losses encourage local ink disruption to be localized according to M. The regularization term penalizes the energy of the shared ink. After each update, δ is projected onto the ℓ_∞ ball of radius ϵ. Since all simulated views are differentiable with respect to δ, the ink can be updated through the fixed developer without differentiating through an editor.

4.3 Cooperative Learning Strategy

The cooperative learning strategy addresses the mutual dependency between the ink and the developer by alternating the two learning phases above. Starting from $(\delta^{(0)}, \theta^{(0)})$, each cooperation round r first fixes $\delta^{(r)}$ and updates the developer on actual editor outputs according to Eq. equation 5, producing $\theta^{(r+1)}$. We then fix the updated developer and optimize the ink according to Eq. equation 9, producing $\delta^{(r+1)}$. The resulting pair $(\delta^{(r+1)}, \theta^{(r+1)})$ is passed to the next round.

Through this alternation, the developer continually adapts to the disruption patterns generated by the current ink, while the ink is updated according to the localization behavior of the current developer. The actual editors are used only in the edit-driven developer learning phase as forward mappings, whereas the developer-guided ink learning phase provides a differentiable path for updating the ink. Therefore, information from actual editing outcomes can progressively influence the learned ink without requiring gradients through the editors.

Deployment After N cooperation rounds, we obtain the learned shared ink $\delta^{(N)}$ and developer $D_{\theta^{(N)}}$. Given a new image x, the publisher constructs

$$x^+ = \text{clip}_{[0,1]}\left(x + \delta^{(N)}\right). \tag{10}$$

The protected image can then be processed by both closed and open source editor to produce y. The receiver predicts the manipulation region as $\widehat{M} = D_{\theta^{(N)}}(y)$. Localization only requires the edited image y and does not require the original image, editing prompt, manipulation mask, or access to the editor.

5 Experiments

5.1 Implementation Details

We conduct experiments on *HumanEdit-Selected* and *Face Data*. HumanEdit-Selected contains 2000 examples from HumanEdit (Bai et al., 2024), covering addition, removal, and replacement. We use 1600 training and 400 test examples with disjoint source-image identities. The source of the facial data and the segmentation settings were randomly selected from locally edit facial images in Bitdefender et al.Ţânţaru et al. (2024), comprising a training set of 1,600 images and a test set of 400 images. Training images are resized to 512×512. The editor suite comprises SD1.5, SD2, and SDXL inpainting models (Podell et al., 2024), together with two API-based editors, denoted as GPT and Grok. Open-source editors receive the source image, instruction, and control mask, whereas API editors receive the image and instruction.

The Ink Developer consists of a frozen DINOv2 ViT-B/14 encoder (Oquab et al., 2023) and a trainable spatial prediction head h_θ, using the reshaped features from the first intermediate DINOv2 block ($d = 768$) and a mask-prediction head composed of a 3×3 convolution with 256 hidden channels, GELU activation, a second 3×3 convolution with 256 hidden channels and GELU activation, and a final 1×1 convolution producing a single-channel localization map. We optimize h_θ with

Table 1: Localization performance on HumanEdit-Selected. IoU, F1, and AUC are on a $[0, 1]$ scale (higher is better). Bold and underline indicate the best and second-best editor-wise results.

	Methods	SD1.5			SD2			SDXL			GPT			Grok			Avg.
		IoU	F1	AUC	IoU	F1	AUC	IoU	F1	AUC	IoU	F1	AUC	IoU	F1	AUC	
Proactive	WAM	0.5676	0.6976	0.9772	0.5854	0.7096	0.9777	0.8022	0.8865	0.9980	0.2895	0.4071	0.8229	0.3643	0.5120	0.7600	0.6905
Proactive	EditGuard	0.1287	0.2127	0.5331	0.1267	0.2118	0.5508	0.1291	0.2223	0.5494	0.2318	0.3399	0.5688	0.3654	0.4939	0.5477	0.3475
Proactive	OmniGuard	0.2212	0.3285	0.7550	0.2324	0.3336	0.7436	0.2285	0.3353	0.7358	0.2240	0.3711	0.7376	0.2973	0.4346	0.6948	0.4449
Proactive	Blank Canvas	0.0163	0.0274	0.3779	0.0169	0.0287	0.3701	0.0129	0.0194	0.3823	0.0286	0.0480	0.3713	0.0494	0.0775	0.3645	0.1461
Passive	TruFor	0.0878	0.1379	0.6245	0.1289	0.1876	0.6670	0.0995	0.1577	0.6367	0.0877	0.1288	0.6432	0.1438	0.2009	0.6152	0.3031
Passive	FakeShield	0.0873	0.1306	0.6122	0.1182	0.1692	0.6316	0.0819	0.1288	0.6102	0.0648	0.0988	0.5043	0.0484	0.0831	0.6006	0.2647
	EditInk	0.4506	0.5821	0.9264	0.4575	0.5890	0.9271	0.7199	0.8162	0.9858	**0.3991**	**0.5111**	**0.8865**	**0.4424**	**0.5539**	**0.9697**	0.6811
	EditInk*	**0.6599**	**0.7724**	**0.9916**	**0.6821**	**0.7921**	**0.9934**	**0.8653**	**0.9248**	**0.9985**	–	–	–	–	–	–	0.8533

Table 2: Localization performance on Face Data. IoU, F1, and AUC are on a $[0, 1]$ scale (higher is better). Bold and underline indicate the best and second-best editor-wise results.

	Methods	SD1.5			SD2			SDXL			GPT			Grok			Avg.
		IoU	F1	AUC	IoU	F1	AUC	IoU	F1	AUC	IoU	F1	AUC	IoU	F1	AUC	
Proactive	WAM	0.6355	0.7360	0.9887	0.6490	0.7539	**0.9915**	0.7949	0.8753	0.9980	0.1483	0.2177	0.7986	0.6989	0.8167	0.8233	0.7284
Proactive	EditGuard	0.1881	0.2843	0.6318	0.1864	0.2836	0.6395	0.2017	0.3033	0.6858	0.2501	0.3983	0.4596	0.4640	0.6233	0.5341	0.4089
Proactive	OmniGuard	0.5239	0.6242	0.9690	0.4856	0.5947	0.9549	0.2907	0.4038	0.8750	0.2441	0.3555	**0.8693**	0.2608	0.3803	0.8740	0.5804
Proactive	Blank Canvas	0.0117	0.0229	0.3198	0.0127	0.0246	0.2941	0.0308	0.0582	0.2984	0.0578	0.0107	0.4301	0.0155	0.0264	0.3997	0.1342
Passive	TruFor	0.2107	0.2946	0.8050	0.2261	0.3145	0.8227	0.2844	0.3729	0.8374	0.2805	0.3718	0.8336	0.2277	0.3233	0.8401	0.4697
Passive	FakeShield	0.3298	0.4050	0.7549	0.2975	0.3668	0.7443	0.2437	0.3189	0.6961	0.0920	0.1246	0.5678	0.0517	0.0806	0.5303	0.3736
	EditInk	0.6102	0.6283	0.9781	0.6894	0.6971	0.9641	0.8430	0.9037	0.9985	**0.3685**	**0.4564**	0.8187	**0.7246**	**0.8767**	**0.8943**	0.7634
	EditInk*	**0.7318**	**0.7519**	**0.9917**	**0.8045**	**0.8270**	0.9845	**0.8746**	**0.9247**	**0.9986**	–	–	–	–	–	–	0.8766

AdamW (Loshchilov & Hutter, 2019) and the shared ink δ with Adam, following the alternating cooperative learning strategy in Sec. 4.3. Both the head and the ink are optimized with a learning rate of 10^{-3}. Training runs for 5 cooperation rounds, with 5 head-training epochs and 150 ink-update steps per round. The head is trained with a batch size of 32, edited-image caches are generated with a batch size of 8 (1 for SDXL), and ink optimization uses a batch size of 1. The ink is constrained by an L_∞ budget $\epsilon = 8/255 \approx 0.03137$, and the regularization coefficient is $\lambda_{\text{reg}} = 10^{-3}$. **EditInk** directly embeds the learned shared ink without further optimization. **EditInk*** initializes from the same learned ink and applies 200 steps of projected gradient descent (PGD); this refinement is evaluated only with the three open-source editors.

5.2 Comparison with Existing Methods

Baselines. We compare EditInk with three groups of localization methods. Watermark-based proactive baselines include WAM (Sander et al., 2024), EditGuard (Zhang et al., 2023), and OmniGuard (Zhang et al., 2024). Blank Canvas (Song et al., 2025) represents pattern-based proactive localization using prescribed model responses. TruFor (Guillaro et al., 2023) and FakeShield (Xu et al., 2024) serve as passive localization baselines. Passive methods analyze edits of unprotected source images, whereas proactive methods analyze edits of their own protected inputs. Unless otherwise specified, we evaluate the complete editor outputs without background compositing.

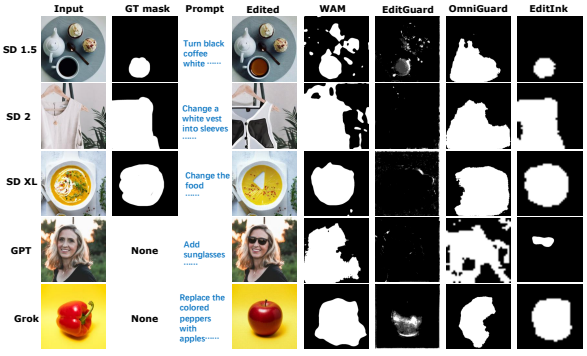

Figure 3: Qualitative comparison of manipulation localization across editors.

Quantitative Comparison. Tables 1 and 2 report pixel-level IoU, F1, and AUC on HumanEdit-Selected and Face Data, respectively. Metrics are computed per image on each method's native prediction grid using resized ground-truth masks, then averaged per editor. Score orientations and de-

cision thresholds are fixed before testing. EditInk directly embeds the shared ink, whereas EditInk* applies 200-step per-image PGD refinement initialized from it and is evaluated only on the three open-source editors.

On HumanEdit-Selected, EditInk achieves the highest IoU, F1, and AUC on both API editors, with F1 gains over WAM of 10.40 and 4.19 percentage points on GPT and Grok, respectively. EditInk* achieves the highest scores across all three metrics on the open-source editors, improving F1 over WAM by 7.48, 8.25, and 3.83 percentage points on SD1.5, SD2, and SDXL, respectively.

On Face Data, EditInk achieves the highest average among methods evaluated on all five editors (76.34 versus 72.84 for WAM) and the best IoU and F1 on both API editors. It also surpasses WAM on all three SDXL metrics, but trails it in F1 on SD1.5 and SD2. EditInk* achieves the best IoU and F1 on all three open-source editors. AUC rankings differ: WAM leads on SD2, while OmniGuard leads on GPT.

Qualitative Comparison. Figure 3 compares EditInk with three watermark-based baselines across five editing examples. In the coffee and clothing examples, WAM and OmniGuard produce responses outside the indicated regions, while EditGuard yields weak or fragmented foreground responses. EditInk captures the main edited regions with fewer off-target responses. WAM also localizes the SDXL food edit well, illustrating that the relative behavior depends on the editing example. For the API examples, EditInk concentrates its response on the added sunglasses and the replacement fruit rather than broadly activating the surrounding content.

Effect of Background Compositing. Figure 4 compares complete editor outputs with region-composited outputs. Scores increase with background compositing for all four methods, with particularly large differences for EditGuard and OmniGuard. For complete editor outputs, EditInk* achieves the highest IoU, F1, and AUC among the compared methods. The differences between these settings show that restoring the protected source background can substantially simplify localization. Performance on region-composited images should therefore not be interpreted as performance on complete editor outputs. Our main comparisons use the latter setting, evaluating localization after the full editing pipeline rather than assuming that the protected background is preserved exactly.

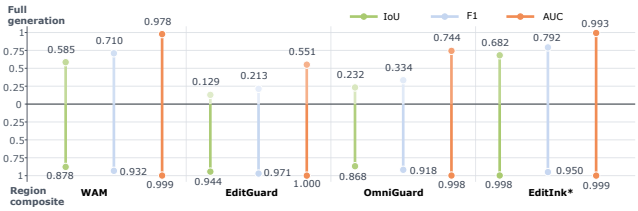

Figure 4: Localization performance in full generation setting and region composite setting.

5.3 Robustness to Image Degradations

Localization after Degradation. We compare EditInk with WAM and OmniGuard on SD2 and SDXL outputs under post-editing image degradations. Each degradation is applied to the complete editor output before localization. We consider JPEG compression with quality factors $\{90, 70, 50\}$, downsampling with factors $\{0.75, 0.50\}$ followed by restoration to the pre-degradation image size, Gaussian blur with standard deviations $\{0.5, 1.0, 1.5\}$, and additive Gaussian noise with standard deviations $\{0.005, 0.01, 0.02\}$ for pixel values in $[0, 1]$. Each curve includes an undegraded reference, and all model parameters and decision rules remain fixed across degradation strengths.

Figure 5 reports the resulting F1 scores. EditInk remains relatively stable under resizing, Gaussian blur, and Gaussian noise for both editors. WAM exhibits larger decreases under resizing and blur, particularly on SD2, widening the performance gap as these degradations strengthen. OmniGuard shows limited variation under several degradations, but its absolute F1 remains substantially lower than that of EditInk. These results are consistent with the goal of preserving localization cues under image degradation.

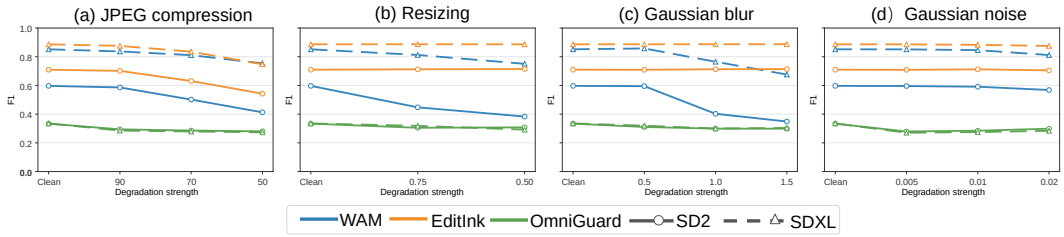

Figure 5: Localization under post-editing image degradations.

Table 3: Ablation results on HumanEdit-Selected (%). The best score is **bolded**.

Setting	SD1.5 IoU	SD2 IoU	SDXL IoU	Mean F1
Ink removed	13.92	13.70	14.08	22.58
Random ink	14.20	13.67	14.76	23.02
VAE → Identity	47.93	48.96	82.01	70.72
Full EditInk	**65.99**	**68.21**	**86.53**	**82.99**

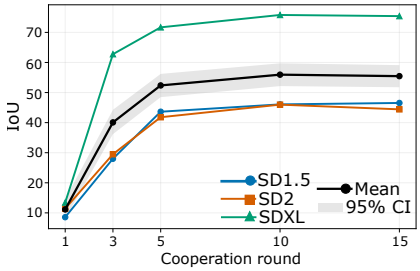

Figure 6: IoU across cooperation rounds.

5.4 Ablation and Cooperation Analysis

We examine the contributions of learned ink, VAE reconstruction, and cooperation rounds. All experiments use the same 400 held-out HumanEdit-Selected source images per editor with paired editing seeds, fixed architecture, data split, and ink budget, and evaluate the shared ink without per-image PGD.

Contribution of Learned Ink. Table 3 compares learned ink with zero ink and random ink under the same perturbation bound, with D_θ frozen. Mean F1 drops from 82.99% with learned ink to 22.58% with zero ink, while random ink achieves only 23.02%. Thus, localization depends on the learned ink–developer pairing rather than perturbation magnitude alone.

VAE Reconstruction during Ink Learning. We retrain the ink–developer pair for five rounds with T_{vae} replaced by the identity mapping during ink learning, while keeping other settings unchanged. Mean F1 decreases from 82.99% to 70.72%, with IoU dropping by 18.06, 19.25, and 4.52 percentage points on SD1.5, SD2, and SDXL, respectively.

Performance across Cooperation Rounds. Figure 6 evaluates matched ink–developer pairs at rounds $r \in \{1, 3, 5, 10, 15\}$. IoU improves rapidly within the first five rounds and changes only modestly thereafter; SDXL reaches a higher plateau, while SD1.5 and SD2 show similar trends. We therefore use five rounds in the final configuration, achieving near-peak localization with fewer training iterations and editor queries.

6 Conclusions

We presented EditInk, a proactive localization framework that learns a shared edit-sensitive ink and an ink developer to identify manipulated regions from the edited image alone. Our alternating cooperative learning strategy trains the developer on actual editor outputs and uses the updated developer to guide ink optimization through differentiable ink-erasure views, without backpropagating through the editors. Experiments on natural-image and face datasets demonstrate effective localization across open-source and API-based editors. In particular, the shared ink achieves higher IoU and F1 than the compared passive and proactive baselines on both API editors without per-image optimization. These results highlight the value of learning the ink and its developer together to make AI manipulation legible.

AI USE STATEMENT

In this work, we used generative AI tools as part of the experimental evaluation to assess the performance of our proposed method and baselines. The generative AI tools were used only within the evaluation and testing pipeline and were not used to develop the core methodology or formulate the main research claims. Additionally, we used generative AI tools for language editing and polishing of the manuscript to improve readability and clarity. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Haonan An, Xiaohui Ye, Guang Hua, Yihang Tao, Hangcheng Cao, Xiangyu Yu, and Yuguang Fang. RecoverMark: Robust watermarking for localization and recovery of manipulated faces. *arXiv preprint arXiv:2602.20618*, 2026. URL https://arxiv.org/abs/2602.20618.

Vishal Asnani, Xi Yin, Tal Hassner, and Xiaoming Liu. MaLP: Manipulation localization using a proactive scheme. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023. URL https://arxiv.org/abs/2303.16976.

Jinbin Bai, Wei Chow, Ling Yang, Xiangtai Li, Juncheng Li, Hanwang Zhang, and Shuicheng Yan. HumanEdit: A high-quality human-rewarded dataset for instruction-based image editing. *arXiv preprint arXiv:2412.04280*, 2024. URL https://arxiv.org/abs/2412.04280.

Tim Brooks, Aleksander Holynski, and Alexei A. Efros. InstructPix2Pix: Learning to follow image editing instructions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 18392–18402, 2023.

Yuzhuo Chen, Zehua Ma, Han Fang, Weiming Zhang, and Nenghai Yu. Tag-wm: Tamper-aware generative image watermarking via diffusion inversion sensitivity. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 16723–16732, 2025.

Zhenliang Gan, Chunya Liu, Yichao Tang, Binghao Wang, Shiwen Cui, Weiqiang Wang, and Xinpeng Zhang. GenPTW: Latent image watermarking for provenance tracing and tamper localization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 4085–4093, 2026. doi: 10.1609/aaai.v40i5.37412. URL https://ojs.aaai.org/index.php/AAAI/article/view/37412.

Fabrizio Guillaro, Davide Cozzolino, Avneesh Sud, Nicholas Dufour, and Luisa Verdoliva. Trufor: Leveraging all-round clues for trustworthy image forgery detection and localization. In *2023 IEEE/CVF conference on computer vision and pattern recognition (CVPR)*, pp. 20606–20615. IEEE, 2023.

Suhyeon Ha, Woo Jae Kim, Joonsung Jeon, Sooel Son, and Sung-eui Yoon. APT: Anchor-aligned perturbations for tamper localization in fully regenerated images. In *European Conference on Computer Vision*, 2026. URL https://arxiv.org/abs/2608.30656.

Qingyu Liu, Yitao Zhang, Zhongjie Ba, Chao Shuai, Peng Cheng, Tianhang Zheng, and Zhibo Wang. Attack-resistant watermarking for aigc image forensics via diffusion-based semantic deflection. In *International Conference on Learning Representations*, 2026.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, Mahmoud Assran, Nicolas Ballas, Wojciech Galuba, Russell Howes, Po-Yao Huang, Shang-Wen Li, Ishan Misra, Michael Rabbat, Vasu Sharma, Gabriel Synnaeve, Hu Xu, Hervé Jégou, Julien Mairal, Patrick Labatut, Armand Joulin, and Piotr Bojanowski. DINOv2: Learning robust visual features without supervision. *arXiv preprint arXiv:2304.07193*, 2023. URL https://arxiv.org/abs/2304.07193.

Dustin Podell, Zion English, Kyle Lacey, Andreas Blattmann, Tim Dockhorn, Jonas Müller, Joe Penna, and Robin Rombach. SDXL: Improving latent diffusion models for high-resolution image synthesis. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/081b08068e4733ae3e7ad019fe8d172f-Abstract-Conference.html.

Tom Sander, Pierre Fernandez, Alain Durmus, Teddy Furon, and Matthijs Douze. Watermark anything with localized messages. *arXiv preprint arXiv:2411.07231*, 2024. URL https://arxiv.org/abs/2411.07231.

Qi Song, Ziyuan Luo, and Renjie Wan. Creating blank canvas against AI-enabled image forgery. *arXiv preprint arXiv:2511.22237*, 2025. URL https://arxiv.org/abs/2511.22237.

Dragoş-Constantin Ţânţaru, Elisabeta Oneaţă, and Dan Oneaţă. Weakly-supervised deepfake localization in diffusion-generated images. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 6258–6268, 2024.

Tianyi Wang, Harry Cheng, Ming-Hui Liu, and Mohan Kankanhalli. FractalForensics: Proactive deepfake detection and localization via fractal watermarks. In *Proceedings of the 33rd ACM International Conference on Multimedia*, pp. 7210–7219, 2025. doi: 10.1145/3746027.3754544. URL https://doi.org/10.1145/3746027.3754544.

Junjiang Wu, Liejun Wang, and Zhiqing Guo. All in one: Unifying deepfake detection, tampering localization, and source tracing with a robust landmark-identity watermark. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 14106–14115, 2026.

Zhipei Xu, Xuanyu Zhang, Runyi Li, Zecheng Tang, Qing Huang, and Jian Zhang. FakeShield: Explainable image forgery detection and localization via multi-modal large language models. *arXiv preprint arXiv:2410.02761*, 2024. URL https://arxiv.org/abs/2410.02761.

Haoxin Yang, Bangzhen Liu, Xuemiao Xu, Cheng Xu, Yuyang Yu, Zikai Huang, Yi Wang, and Shengfeng He. StableGuard: Towards unified copyright protection and tamper localization in latent diffusion models. In *Advances in Neural Information Processing Systems*, 2025. URL https://arxiv.org/abs/2509.17993.

Qifan Yu, Wei Chow, Zhongqi Yue, Kaihang Pan, Yang Wu, Xiaoyang Wan, Juncheng Li, Siliang Tang, Hanwang Zhang, and Yueting Zhuang. AnyEdit: Mastering unified high-quality image editing for any idea. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2025. URL https://arxiv.org/abs/2411.15738.

Xuanyu Zhang, Runyi Li, Jiwen Yu, Youmin Xu, Weiqi Li, and Jian Zhang. EditGuard: Versatile image watermarking for tamper localization and copyright protection. *arXiv preprint arXiv:2312.08883*, 2023. URL https://arxiv.org/abs/2312.08883.

Xuanyu Zhang, Zecheng Tang, Zhipei Xu, Runyi Li, Youmin Xu, Bin Chen, Feng Gao, and Jian Zhang. OmniGuard: Hybrid manipulation localization via augmented versatile deep image watermarking. *arXiv preprint arXiv:2412.01615*, 2024. URL https://arxiv.org/abs/2412.01615.

Jiren Zhu, Russell Kaplan, Justin Johnson, and Li Fei-Fei. HiDDeN: Hiding data with deep networks. In *Proceedings of the European Conference on Computer Vision*, pp. 657–672, 2018. URL https://www.ecva.net/papers/eccv_2018/papers_ECCV/html/Jiren_Zhu_HiDDeN_Hiding_Data_ECCV_2018_paper.php.